MITgcm Documentation

Release 190ac412e

Alistair Adcroft, Jean-Michel Campin, Ed Doddridge, Stephanie Dutkiewicz, Constantinos Evangelinos, David Ferreira, Mick Follows, Gael Forget, Baylor Fox-Kemper, Patrick Heimbach, Chris Hill, Ed Hill, Helen Hill, Oliver Jahn, Jody Klymak, Martin Losch, John Marshall, Guillaume Maze, Matt Mazloff, Dimitris Menemenlis, Andrea Molod, and Jeff Scott

Sep 30, 2019

Contents:

CHAPTER 1

Overview

This document provides the reader with the information necessary to carry out numerical experiments using MITgcm. It gives a comprehensive description of the continuous equations on which the model is based, the numerical algorithms the model employs and a description of the associated program code. Along with the hydrodynamical kernel, physical and biogeochemical parameterizations of key atmospheric and oceanic processes are available. A number of examples illustrating the use of the model in both process and general circulation studies of the atmosphere and ocean are also presented.

1.1 Introduction

MITgcm has a number of novel aspects:

- it can be used to study both atmospheric and oceanic phenomena; one hydrodynamical kernel is used to drive forward both atmospheric and oceanic models - see Figure 1.1
- it has a non-hydrostatic capability and so can be used to study both small-scale and large scale processes - see Figure 1.2
- finite volume techniques are employed yielding an intuitive discretization and support for the treatment of irregular geometries using orthogonal curvilinear grids and shaved cells - see Figure 1.3
- tangent linear and adjoint counterparts are automatically maintained along with the forward model, permitting sensitivity and optimization studies.
- the model is developed to perform efficiently on a wide variety of computational platforms.

Key publications reporting on and charting the development of the model are Hill and Marshall (1995), Marshall et al. (1997a), Marshall et al. (1997b), Adcroft and Marshall (1997), Marshall et al. (1998), Adcroft and Marshall (1999), Hill et al. (1999), Marotzke et al. (1999), Adcroft and Campin (2004), Adcroft et al. (2004b), Marshall et al. (2004) (an overview on the model formulation can also be found in Adcroft et al. (2004c)):

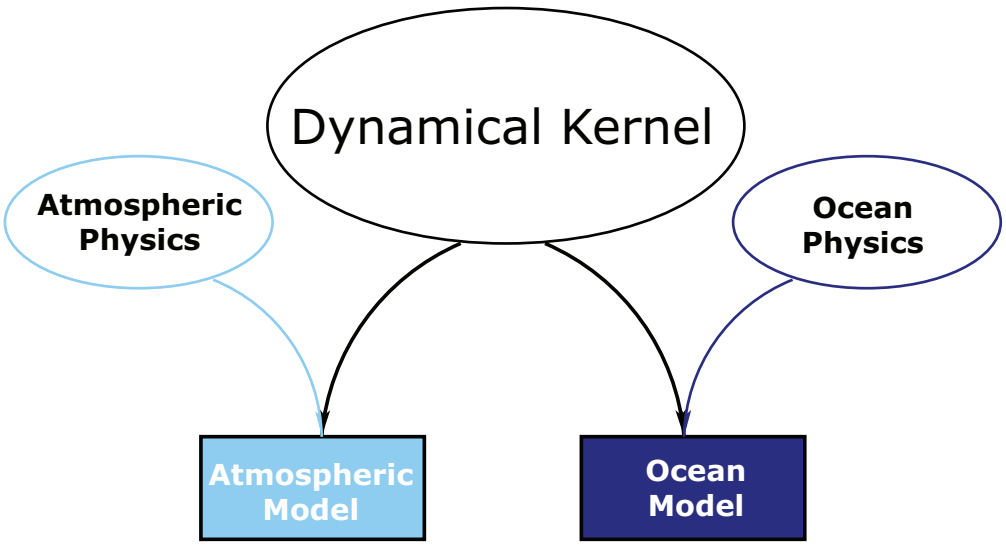

Figure 1.1: MITgcm has a single dynamical kernel that can drive forward either oceanic or atmospheric simulations.

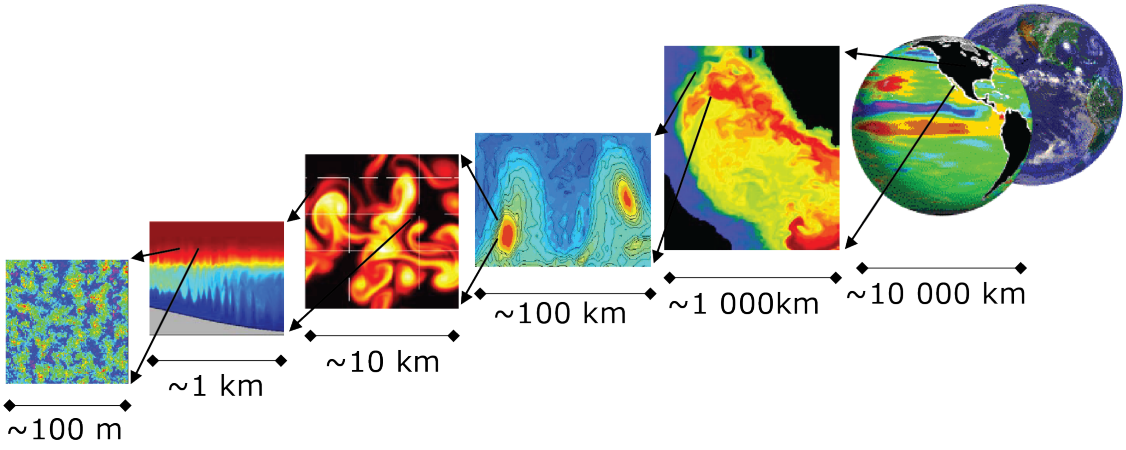

Figure 1.2: MITgcm has non-hydrostatic capabilities, allowing the model to address a wide range of phenomenon - from convection on the left, all the way through to global circulation patterns on the right.

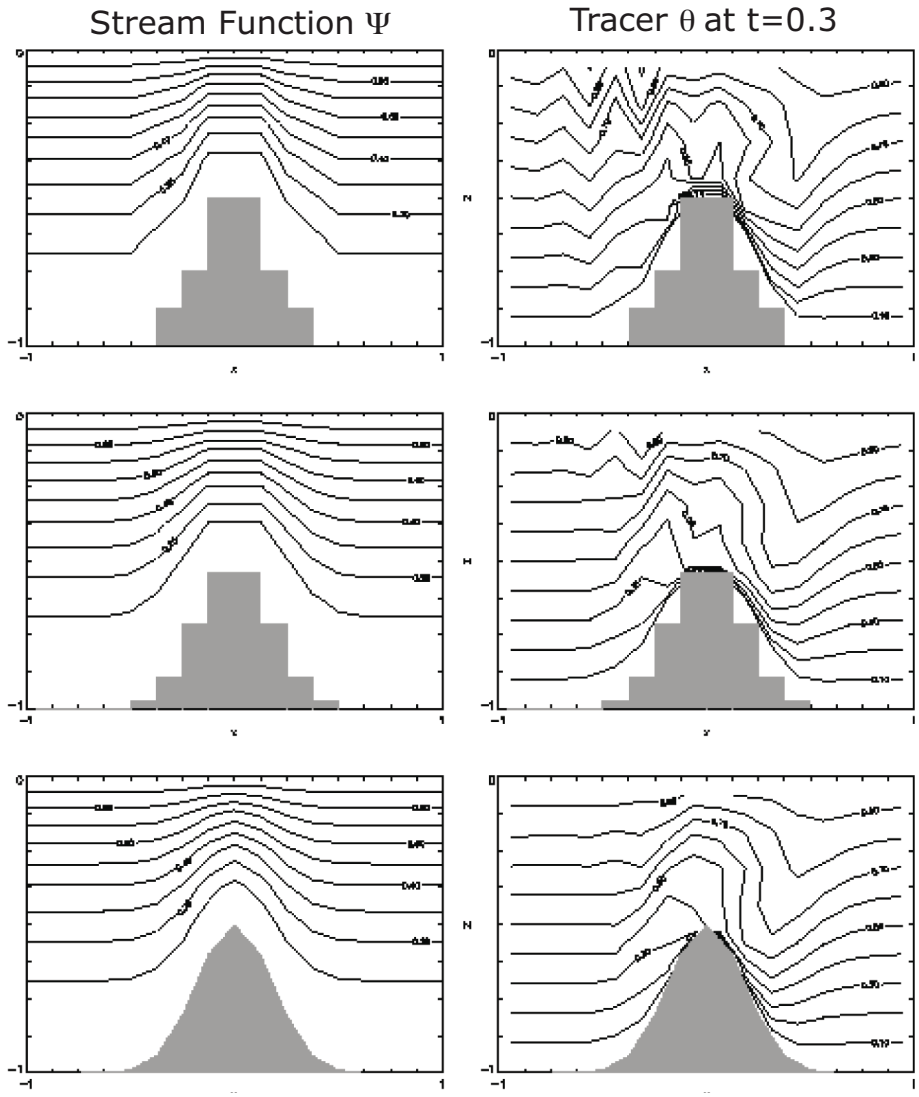

Figure 1.3: Finite volume techniques (bottom panel) are used, permitting a treatment of topography that rivals σ (terrain following) coordinates.

Hill, C. and J. Marshall, (1995) Application of a Parallel Navier-Stokes Model to Ocean Circulation in Parallel Computational Fluid Dynamics, In Proceedings of Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers, 545-552. Elsevier Science B.V.: New York [HM95]

Marshall, J., C. Hill, L. Perelman, and A. Adcroft, (1997a) Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling, J. Geophysical Res., **102(C3)**, 5733-5752 [MHPA97]

Marshall, J., A. Adcroft, C. Hill, L. Perelman, and C. Heisey, (1997b) A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers, J. Geophysical Res., **102(C3)**, 5753-5766 [MAH+97]

Adcroft, A.J., Hill, C.N. and J. Marshall, (1997) Representation of topography by shaved cells in a height coordinate ocean model, Mon Wea Rev, **125**, 2293-2315 [AHM97]

Marshall, J., Jones, H. and C. Hill, (1998) Efficient ocean modeling using non-hydrostatic algorithms, Journal of Marine Systems, **18**, 115-134 [MJH98]

Adcroft, A., Hill C. and J. Marshall: (1999) A new treatment of the Coriolis terms in C-grid models at both high and low resolutions, Mon. Wea. Rev., **127**, 1928-1936 [AHM99]

Hill, C, Adcroft,A., Jamous,D., and J. Marshall, (1999) A Strategy for Terascale Climate Modeling, In Proceedings of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology, 406-425 World Scientific Publishing Co: UK [HAJM99]

Marotzke, J, Giering,R., Zhang, K.Q., Stammer,D., Hill,C., and T.Lee, (1999) Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport variability, J. Geophysical Res., **104(C12)**, 29,529-29,547 [MGZ+99]

A. Adcroft and J.-M. Campin, (2004a) Re-scaled height coordinates for accurate representation of free-surface flows in ocean circulation models, Ocean Modelling, **7**, 269–284 [AC04]

A. Adcroft, J.-M. Campin, C. Hill, and J. Marshall, (2004b) Implementation of an atmosphere-ocean general circulation model on the expanded spherical cube, Mon Wea Rev , **132**, 2845–2863 [ACHM04]

J. Marshall, A. Adcroft, J.-M. Campin, C. Hill, and A. White, (2004) Atmosphere-ocean modeling exploiting fluid isomorphisms, Mon. Wea. Rev., **132**, 2882–2894 [MAC+04]

A. Adcroft, C. Hill, J.-M. Campin, J. Marshall, and P. Heimbach, (2004c) Overview of the formulation and numerics of the MITgcm, In Proceedings of the ECMWF seminar series on Numerical Methods, Recent developments in numerical methods for atmosphere and ocean modelling, 139–149. URL: http://mitgcm.org/pdfs/ECMWF2004-Adcroft.pdf [AHJMC+04]

We begin by briefly showing some of the results of the model in action to give a feel for the wide range of problems that can be addressed using it.

1.2 Illustrations of the model in action

MITgcm has been designed and used to model a wide range of phenomena, from convection on the scale of meters in the ocean to the global pattern of atmospheric winds - see Figure 1.2. To give a flavor of the kinds of problems the model has been used to study, we briefly describe some of them here. A more detailed description of the underlying formulation, numerical algorithm and implementation that lie behind these calculations is given later. Indeed many of the illustrative examples shown below can be easily reproduced: simply download the model (the minimum you need is a PC running Linux, together with a FORTRAN77 compiler) and follow the examples described in detail in the documentation.

1.2.1 Global atmosphere: 'Held-Suarez' benchmark

A novel feature of MITgcm is its ability to simulate, using one basic algorithm, both atmospheric and oceanographic flows at both small and large scales.

Figure 1.4 shows an instantaneous plot of the 500 mb temperature field obtained using the atmospheric isomorph of MITgcm run at 2.8° resolution on the cubed sphere. We see cold air over the pole (blue) and warm air along an equatorial band (red). Fully developed baroclinic eddies spawned in the northern hemisphere storm track are evident. There are no mountains or land-sea contrast in this calculation, but you can easily put them in. The model is driven by relaxation to a radiative-convective equilibrium profile, following the description set out in Held and Suarez (1994) [HS94] designed to test atmospheric hydrodynamical cores - there are no mountains or land-sea contrast.

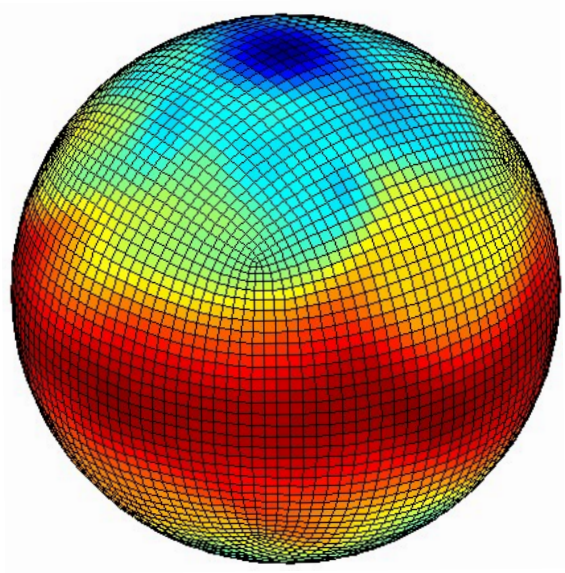

Figure 1.4: Instantaneous plot of the temperature field at 500 mb obtained using the atmospheric isomorph of MITgcm

As described in Adcroft et al. (2004) [ACHM04], a 'cubed sphere' is used to discretize the globe permitting a uniform griding and obviated the need to Fourier filter. The 'vector-invariant' form of MITgcm supports any orthogonal curvilinear grid, of which the cubed sphere is just one of many choices.

Figure 1.5 shows the 5-year mean, zonally averaged zonal wind from a 20-level configuration of the model. It compares favorable with more conventional spatial discretization approaches. The two plots show the field calculated using the cube-sphere grid and the flow calculated using a regular, spherical polar latitude-longitude grid. Both grids are supported within the model.

1.2.2 Ocean gyres

Baroclinic instability is a ubiquitous process in the ocean, as well as the atmosphere. Ocean eddies play an important role in modifying the hydrographic structure and current systems of the oceans. Coarse resolution models of the oceans cannot resolve the eddy field and yield rather broad, diffusive patterns of ocean currents. But if the resolution of our models is increased until the baroclinic instability process is resolved, numerical solutions of a different and much more realistic kind, can be obtained.

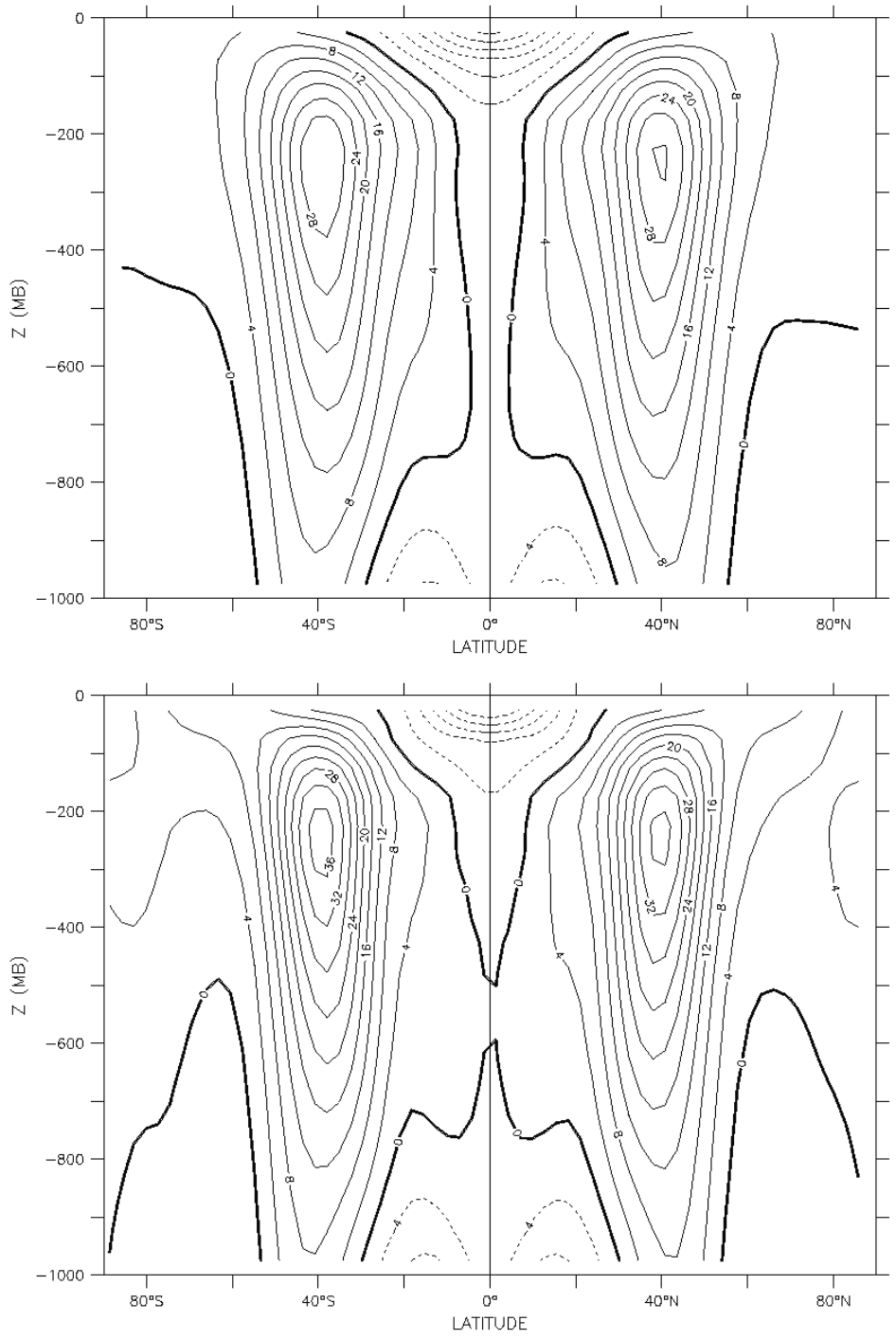

Figure 1.5: Five year mean, zonally averaged zonal flow for cube-sphere simulation (top) and latitude-longitude simulation (bottom) and using Held-Suarez forcing. Note the difference in the solutions over the pole — the cubed sphere is superior.

Figure 1.6 shows the surface temperature and velocity field obtained from MITgcm run at $\frac{1}{6}^\circ$ horizontal resolution on a *lat-lon* grid in which the pole has been rotated by 90° on to the equator (to avoid the converging of meridian in northern latitudes). 21 vertical levels are used in the vertical with a 'lopped cell' representation of topography. The development and propagation of anomalously warm and cold eddies can be clearly seen in the Gulf Stream region. The transport of warm water northward by the mean flow of the Gulf Stream is also clearly visible.

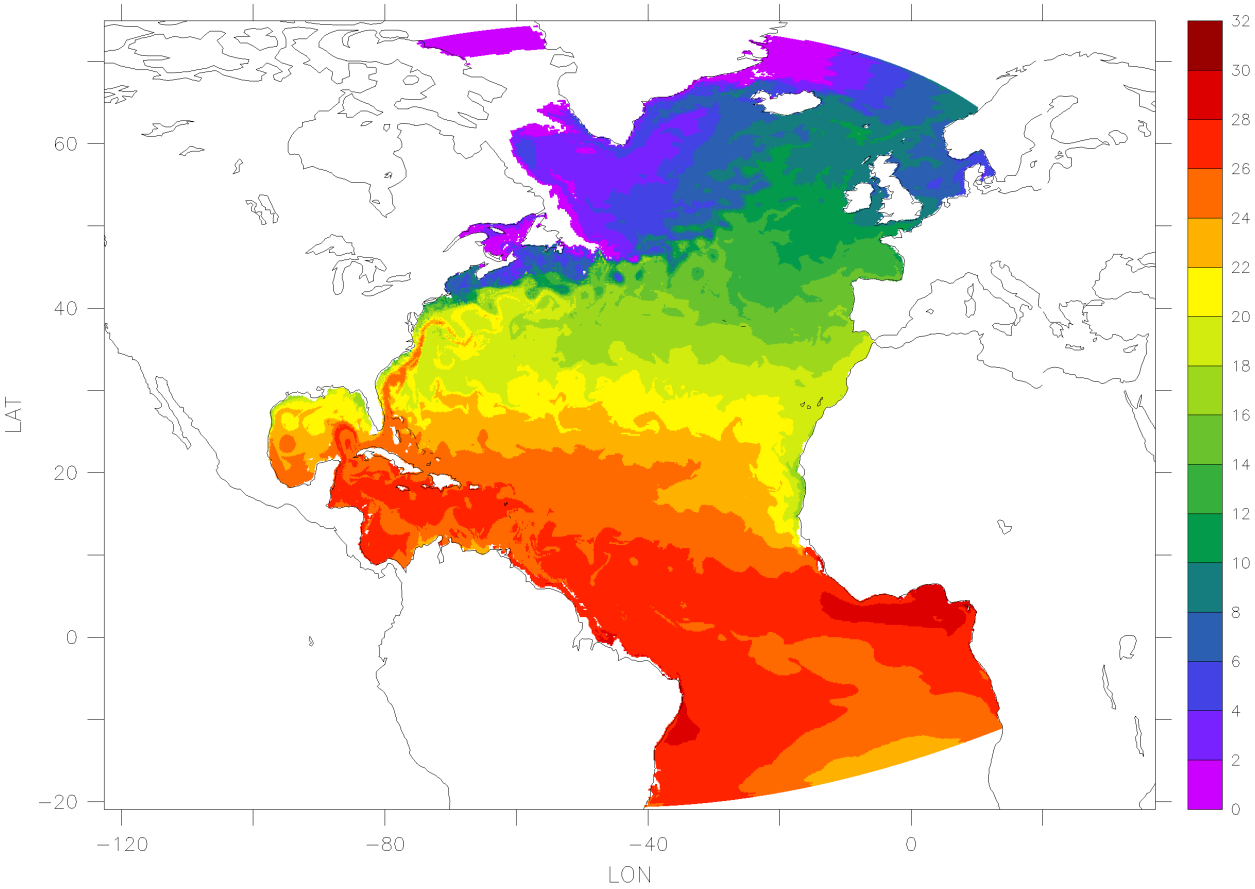

Figure 1.6: Instantaneous temperature map from a $\frac{1}{6}^\circ$ simulation of the North Atlantic. The figure shows the temperature in the second layer (37.5 m deep).

1.2.3 Global ocean circulation

Figure 1.7 shows the pattern of ocean currents at the surface of a 4° global ocean model run with 15 vertical levels. Lopped cells are used to represent topography on a regular *lat-lon* grid extending from 70°N to 70°S. The model is driven using monthly-mean winds with mixed boundary conditions on temperature and salinity at the surface. The transfer properties of ocean eddies, convection and mixing is parameterized in this model.

Figure 1.8 shows the meridional overturning circulation of the global ocean in Sverdrups.

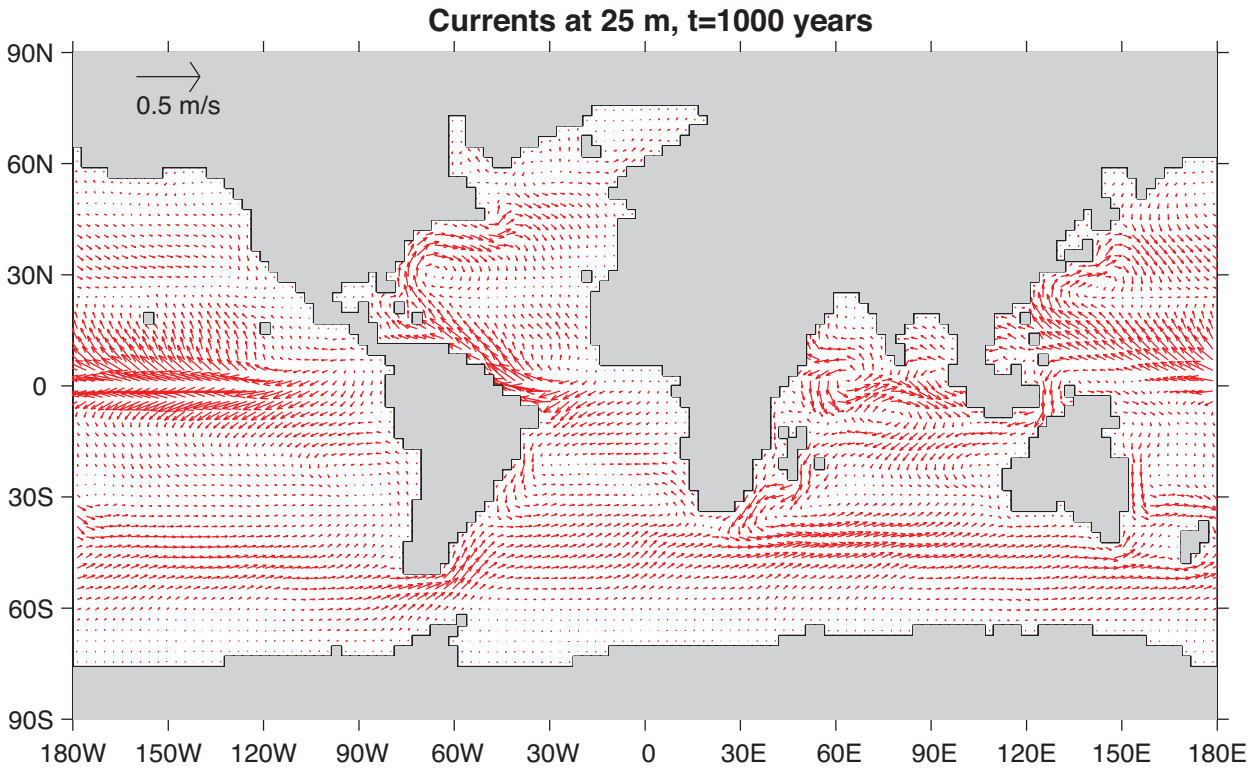

Figure 1.7: Pattern of surface ocean currents from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

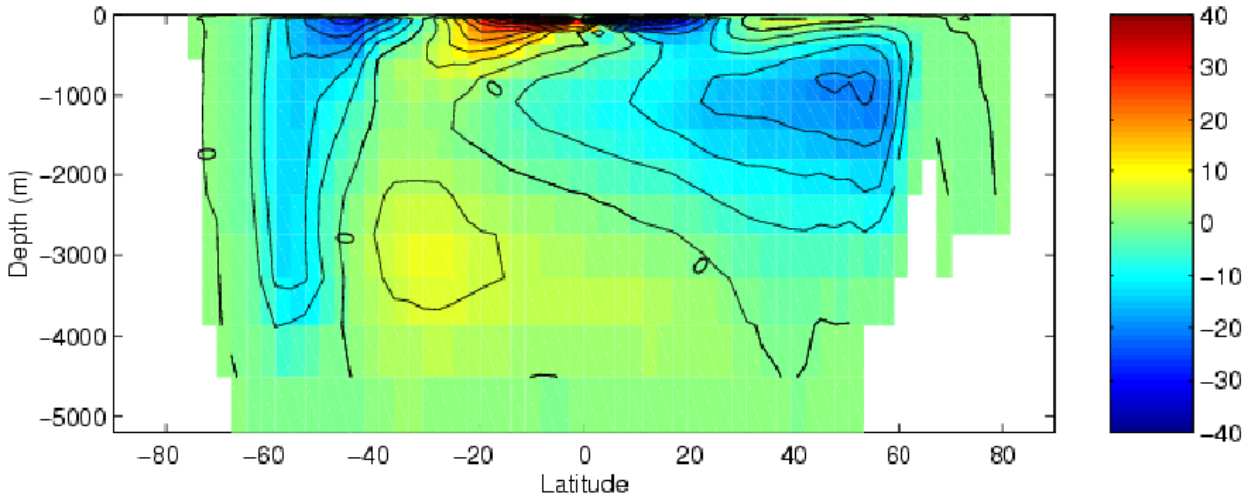

Figure 1.8: Meridional overturning stream function (in Sverdrups) from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

1.2.4 Convection and mixing over topography

Dense plumes generated by localized cooling on the continental shelf of the ocean may be influenced by rotation when the deformation radius is smaller than the width of the cooling region. Rather than gravity plumes, the mechanism for moving dense fluid down the shelf is then through geostrophic eddies. The simulation shown in Figure 1.9 (blue is cold dense fluid, red is warmer, lighter fluid) employs the non-hydrostatic capability of MITgcm to trigger convection by surface cooling. The cold, dense water falls down the slope but is deflected along the slope by rotation. It is found that entrainment in the vertical plane is reduced when rotational control is strong, and replaced by lateral entrainment due to the baroclinic instability of the along-slope current.

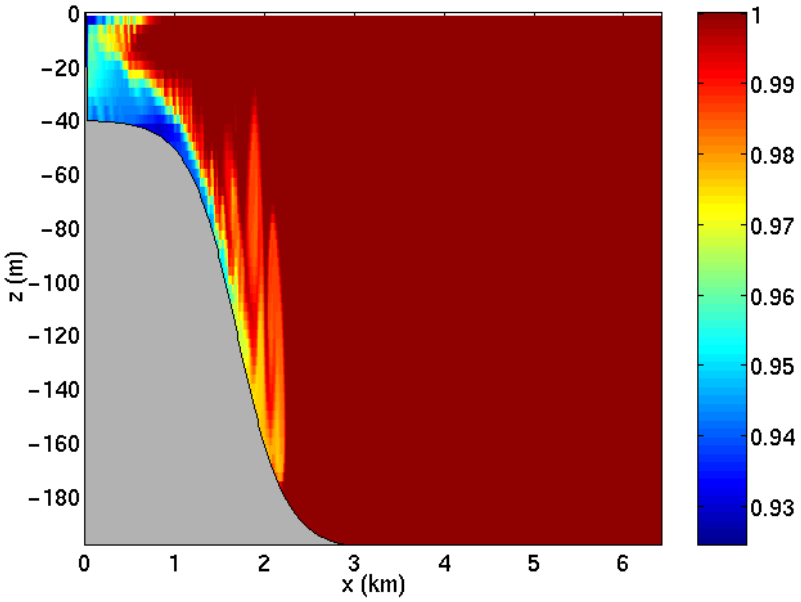

Figure 1.9: MITgcm run in a non-hydrostatic configuration to study convection over a slope.

1.2.5 Boundary forced internal waves

The unique ability of MITgcm to treat non-hydrostatic dynamics in the presence of complex geometry makes it an ideal tool to study internal wave dynamics and mixing in oceanic canyons and ridges driven by large amplitude barotropic tidal currents imposed through open boundary conditions.

Figure 1.10 shows the influence of cross-slope topographic variations on internal wave breaking - the cross-slope velocity is in color, the density contoured. The internal waves are excited by application of open boundary conditions on the left. They propagate to the sloping boundary (represented using MITgcm's finite volume spatial discretization) where they break under non-hydrostatic dynamics.

1.2.6 Parameter sensitivity using the adjoint of MITgcm

Forward and tangent linear counterparts of MITgcm are supported using an 'automatic adjoint compiler'. These can be used in parameter sensitivity and data assimilation studies.

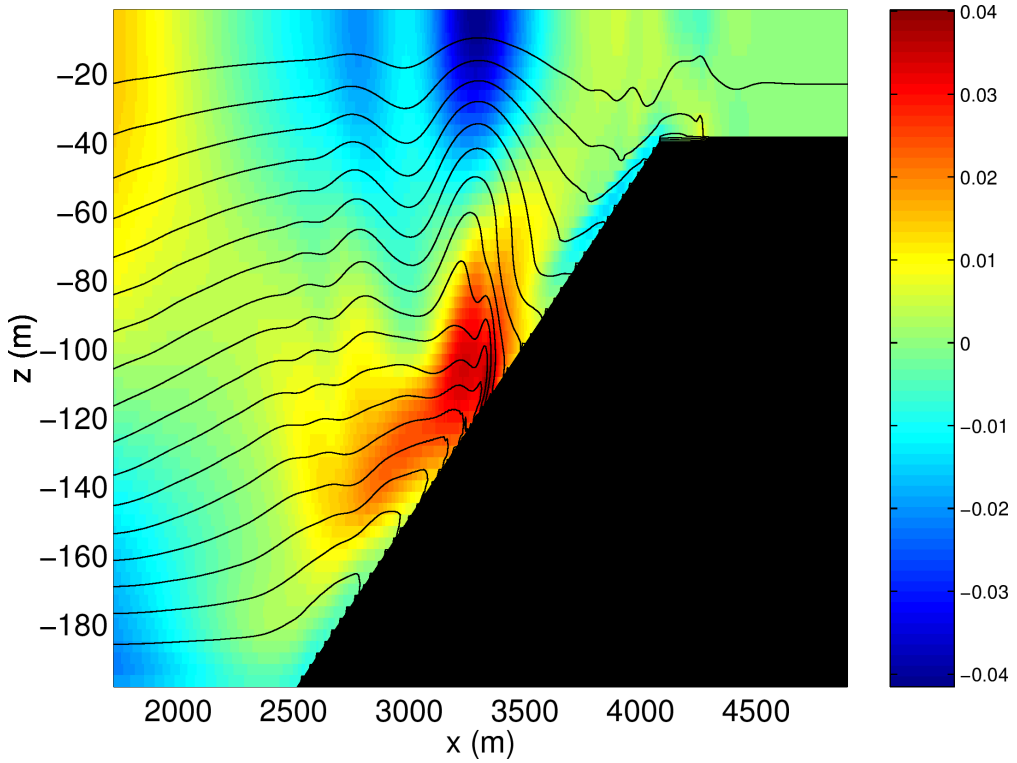

Figure 1.10: Simulation of internal waves forced at an open boundary (on the left) impacting a sloping shelf. The along slope velocity is shown colored, contour lines show density surfaces. The slope is represented with high-fidelity using lopped cells.

As one example of application of the MITgcm adjoint, Figure 1.11 maps the gradient $\frac{\partial J}{\partial \mathcal{H}}$ where J is the magnitude of the overturning stream-function shown in Figure 1.8 at 60°N and $\mathcal{H}(\lambda, \varphi)$ is the mean, local air-sea heat flux over a 100 year period. We see that J is sensitive to heat fluxes over the Labrador Sea, one of the important sources of deep water for the thermohaline circulations. This calculation also yields sensitivities to all other model parameters.

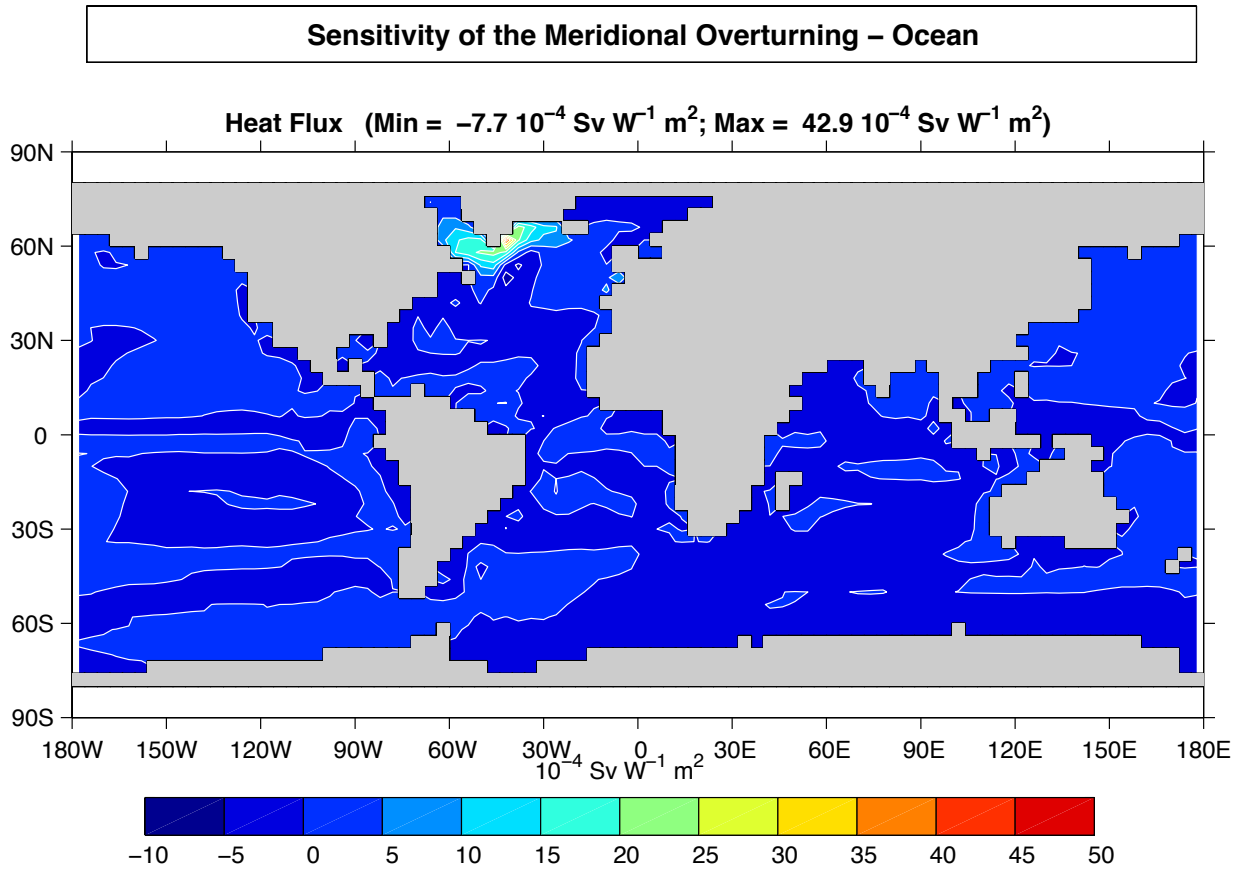

Figure 1.11: Sensitivity of meridional overturning strength to surface heat flux changes. Contours show the magnitude of the response (in Sv x 10^{-4}) that a persistent +1 Wm^{-2} heat flux anomaly at a given grid point would produce.

1.2.7 Global state estimation of the ocean

An important application of MITgcm is in state estimation of the global ocean circulation. An appropriately defined 'cost function', which measures the departure of the model from observations (both remotely sensed and in-situ) over an interval of time, is minimized by adjusting 'control parameters' such as air-sea fluxes, the wind field, the initial conditions etc. Figure 1.12 and Figure 1.13 show the large scale planetary circulation and a Hopf-Muller plot of Equatorial sea-surface height. Both are obtained from assimilation bringing the model in to consistency with altimetric and in-situ observations over the period 1992-1997.

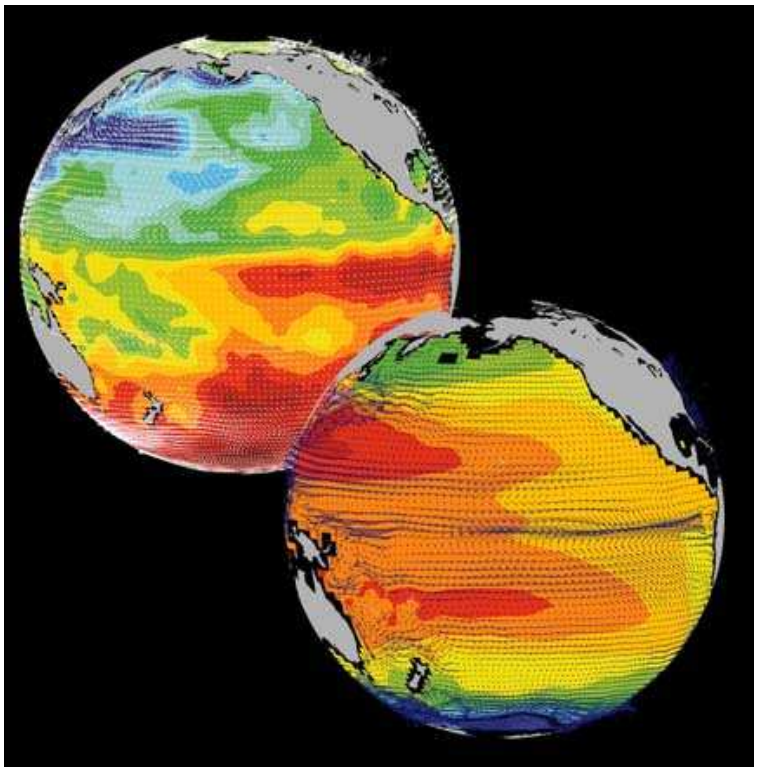

Figure 1.12: Circulation patterns from a multi-year, global circulation simulation constrained by Topex altimeter data and WOCE cruise observations. This output is from a higher resolution, shorter duration experiment with equatorially enhanced grid spacing.

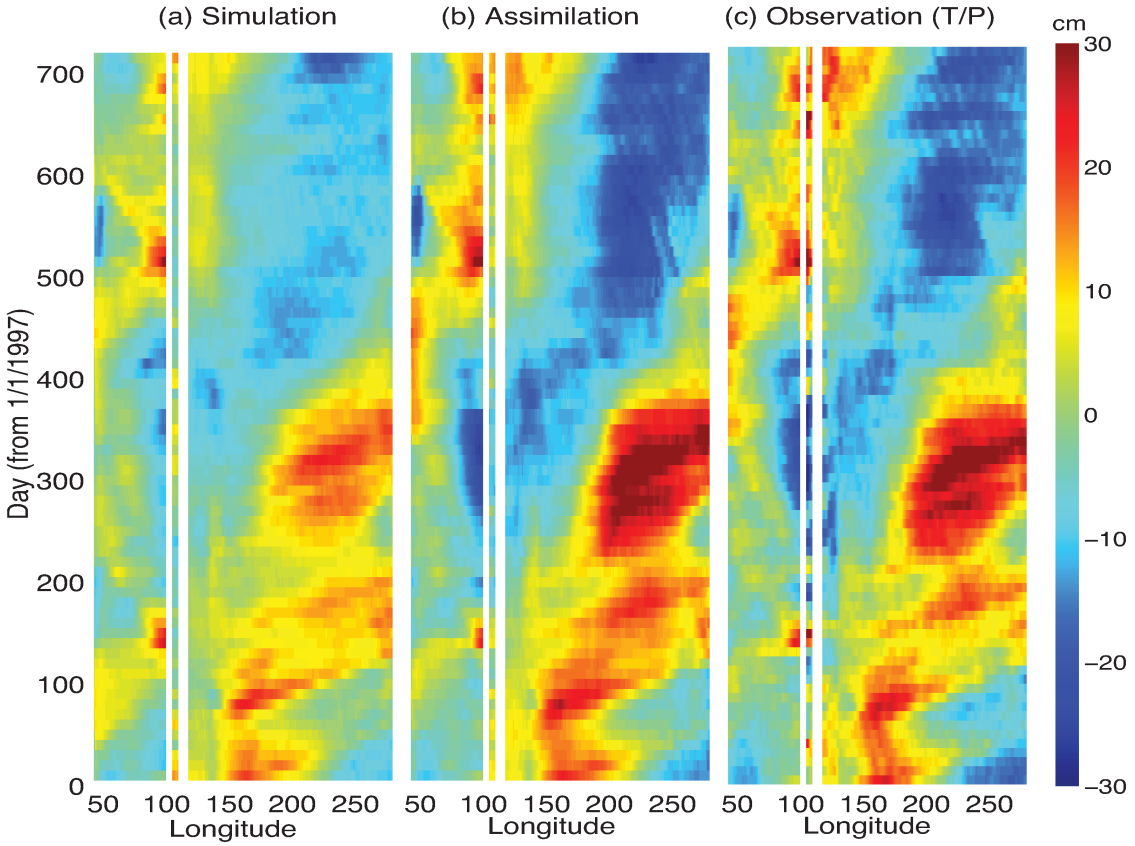

Figure 1.13: Equatorial sea-surface height in unconstrained (left), constrained (middle) simulations and in observations (right).

1.2.8 Ocean biogeochemical cycles

MITgcm is being used to study global biogeochemical cycles in the ocean. For example one can study the effects of interannual changes in meteorological forcing and upper ocean circulation on the fluxes of carbon dioxide and oxygen between the ocean and atmosphere. Figure 1.14 shows the annual air-sea flux of oxygen and its relation to density outcrops in the southern oceans from a single year of a global, interannually varying simulation. The simulation is run at 1°x1° resolution telescoping to $\frac{1}{3}^\circ$ x $\frac{1}{3}^\circ$ in the tropics (not shown).

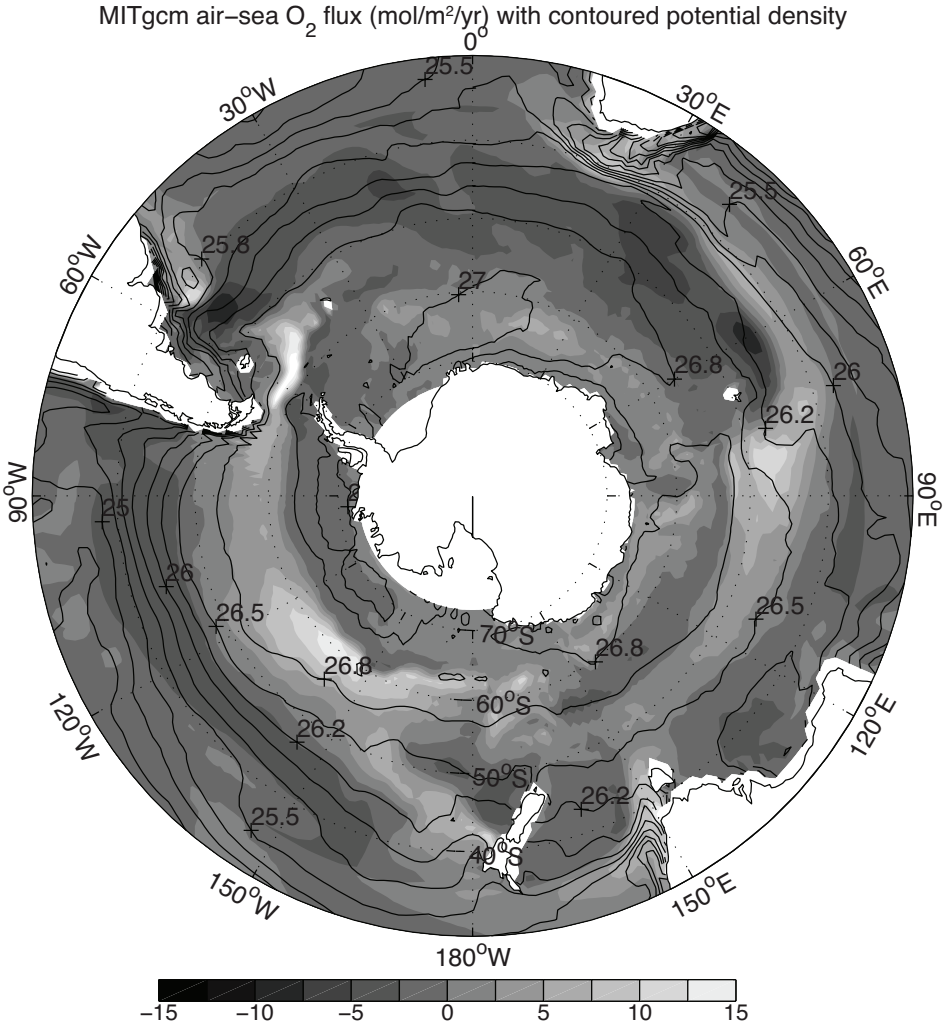

Figure 1.14: Annual air-sea flux of oxygen (shaded) plotted along with potential density outcrops of the surface of the southern ocean from a global 1°x1° integration with a telescoping grid (to $\frac{1}{3}^\circ$) at the equator.

1.2.9 Simulations of laboratory experiments

Figure 1.16 shows MITgcm being used to simulate a laboratory experiment (Figure 1.15) inquiring into the dynamics of the Antarctic Circumpolar Current (ACC). An initially homogeneous tank of water (1 m in diameter) is driven from its free surface by a rotating heated disk. The combined action of mechanical and thermal forcing creates a lens of fluid which becomes baroclinically unstable. The stratification and depth of penetration of the lens is arrested by its instability in a process analogous to that which sets the stratification of the ACC.

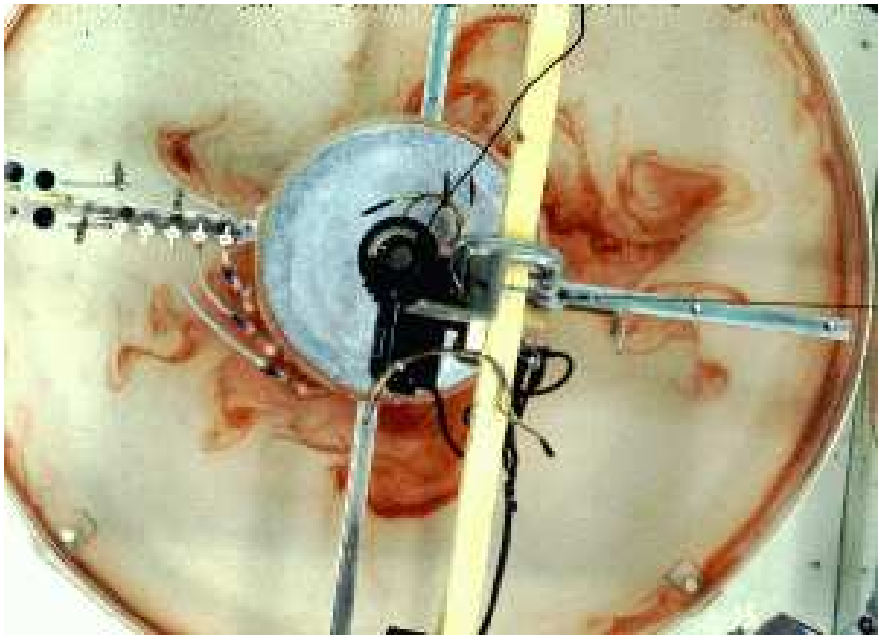

Figure 1.15: A 1 m diameter laboratory experiment simulating the dynamics of the Antarctic Circumpolar Current.

1.3 Continuous equations in 'r' coordinates

To render atmosphere and ocean models from one dynamical core we exploit 'isomorphisms' between equation sets that govern the evolution of the respective fluids - see Figure 1.17. One system of hydrodynamical equations is written down and encoded. The model variables have different interpretations depending on whether the atmosphere or ocean is being studied. Thus, for example, the vertical coordinate 'r' is interpreted as pressure, p, if we are modeling the atmosphere (right hand side of Figure 1.17) and height, z, if we are modeling the ocean (left hand side of Figure 1.17).

The state of the fluid at any time is characterized by the distribution of velocity $\vec{\mathbf{v}}$, active tracers θ and S, a 'geopotential' ϕ and density $\rho = \rho(\theta, S, p)$ which may depend on θ, S, and p. The equations that govern the evolution of these fields, obtained by applying the laws of classical mechanics and thermodynamics to a Boussinesq, Navier-Stokes fluid are, written in terms of a generic vertical coordinate, r, so that the appropriate kinematic boundary conditions can be applied isomorphically see Figure 1.18.

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + \left(2\vec{\Omega} \times \vec{\mathbf{v}}\right)_h + \nabla_h \phi = \mathcal{F}_{\vec{\mathbf{v}}_h} \text{ horizontal momentum} \tag{1.1}$$

$$\frac{D\dot{r}}{Dt} + \widehat{k} \cdot \left(2\vec{\Omega} \times \vec{\mathbf{v}}\right) + \frac{\partial \phi}{\partial r} + b = \mathcal{F}_{\dot{r}} \text{ vertical momentum} \tag{1.2}$$

$$\nabla_h \cdot \vec{\mathbf{v}}_h + \frac{\partial \dot{r}}{\partial r} = 0 \text{ continuity} \tag{1.3}$$

$$b = b(\theta, S, r) \text{ equation of state} \tag{1.4}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \text{ potential temperature} \tag{1.5}$$

$$\frac{DS}{Dt} = \mathcal{Q}_S \text{ humidity/salinity} \tag{1.6}$$

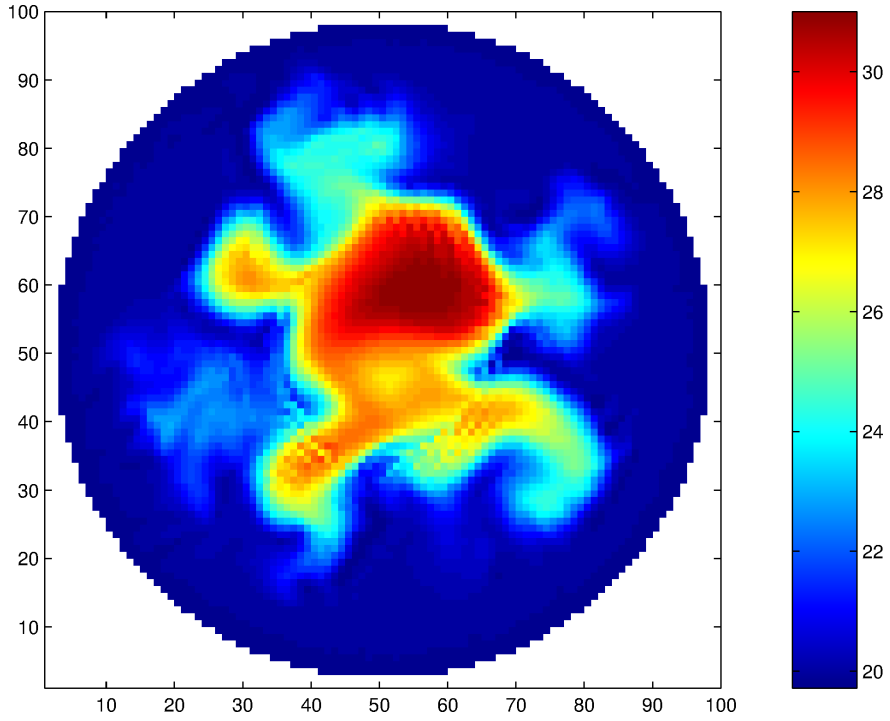

Figure 1.16: A numerical simulation of the laboratory experiment using MITgcm.

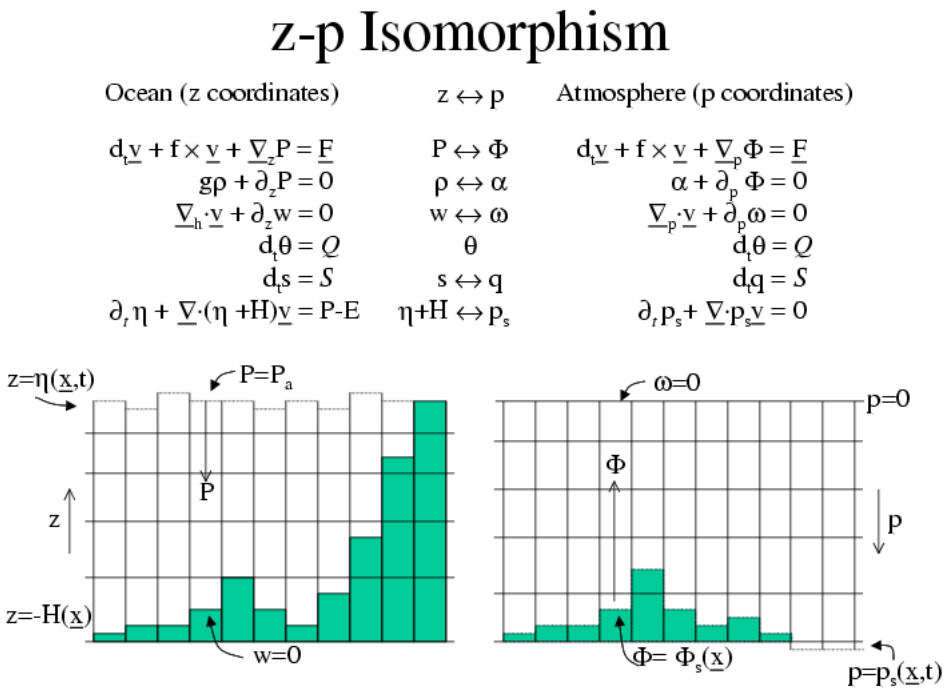

Figure 1.17: Isomorphic equation sets used for atmosphere (right) and ocean (left).

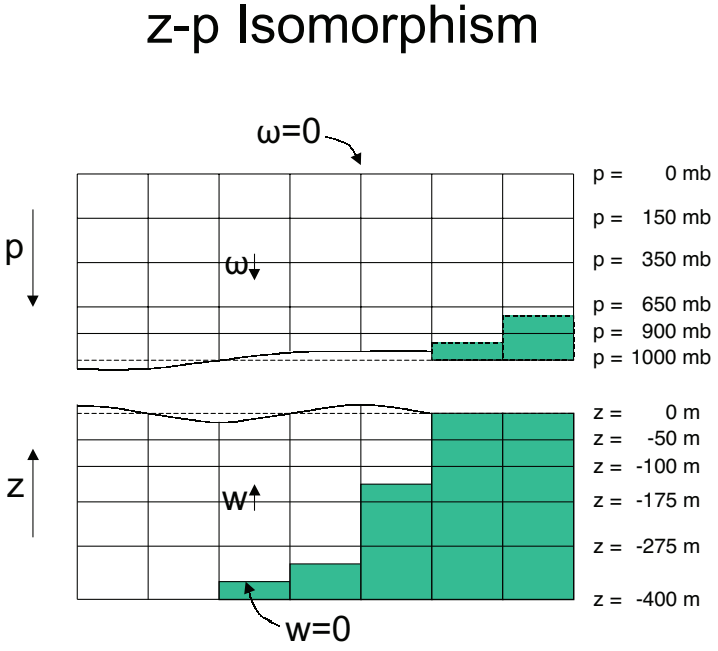

Figure 1.18: Vertical coordinates and kinematic boundary conditions for atmosphere (top) and ocean (bottom).

Here:

r is the vertical coordinate

$\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{\mathbf{v}} \cdot \nabla$ is the total derivative

$\nabla = \nabla_h + \widehat{k}\frac{\partial}{\partial r}$ is the 'grad' operator

with ∇_h operating in the horizontal and $\widehat{k}\frac{\partial}{\partial r}$ operating in the vertical, where $\widehat{k}$ is a unit vector in the vertical

t is time

$\vec{\mathbf{v}} = (u, v, \dot{r}) = (\vec{\mathbf{v}}_h, \dot{r})$ is the velocity

ϕ is the 'pressure'/'geopotential'

$\vec{\Omega}$ is the Earth's rotation

b is the 'buoyancy'

θ is potential temperature

S is specific humidity in the atmosphere; salinity in the ocean

$\mathcal{F}_{\vec{\mathbf{v}}}$ are forcing and dissipation of $\vec{\mathbf{v}}$

$\mathcal{Q}_\theta$ are forcing and dissipation of θ

$\mathcal{Q}_S$ are forcing and dissipation of S

The $\mathcal{F}'s$ and $\mathcal{Q}'s$ are provided by 'physics' and forcing packages for atmosphere and ocean. These are described in later chapters.

1.3.1 Kinematic Boundary conditions

1.3.1.1 Vertical

at fixed and moving r surfaces we set (see Figure 1.18):

$$\dot{r} = 0 \text{ at } r = R_{fixed}(x, y) \text{ (ocean bottom, top of the atmosphere)} \tag{1.7}$$

$$\dot{r} = \frac{Dr}{Dt} \text{ at } r = R_{moving}(x, y) \text{ (ocean surface, bottom of the atmosphere)} \tag{1.8}$$

Here

$$R_{moving} = R_o + \eta$$

where $R_o(x, y)$ is the '$r-$value' (height or pressure, depending on whether we are in the atmosphere or ocean) of the 'moving surface' in the resting fluid and η is the departure from $R_o(x, y)$ in the presence of motion.

1.3.1.2 Horizontal

$$\vec{\mathbf{v}} \cdot \vec{\mathbf{n}} = 0 \tag{1.9}$$

where $\vec{\mathbf{n}}$ is the normal to a solid boundary.

1.3.2 Atmosphere

In the atmosphere, (see Figure 1.18), we interpret:

$$r = p \text{ is the pressure} \tag{1.10}$$

$$\dot{r} = \frac{Dp}{Dt} = \omega \text{ is the vertical velocity in p coordinates} \tag{1.11}$$

$$\phi = g\, z \text{ is the geopotential height} \tag{1.12}$$

$$b = \frac{\partial \Pi}{\partial p}\theta \text{ is the buoyancy} \tag{1.13}$$

$$\theta = T(\frac{p_c}{p})^{\kappa} \text{ is potential temperature} \tag{1.14}$$

$$S = q \text{ is the specific humidity} \tag{1.15}$$

where

T is absolute temperature

p is the pressure

z is the height of the pressure surface

g is the acceleration due to gravity

In the above the ideal gas law, $p = \rho RT$, has been expressed in terms of the Exner function $\Pi(p)$ given by (1.16) (see also Section 1.4.1)

$$\Pi(p) = c_p(\frac{p}{p_c})^{\kappa} \tag{1.16}$$

where p_c is a reference pressure and $\kappa = R/c_p$ with R the gas constant and c_p the specific heat of air at constant pressure.

At the top of the atmosphere (which is 'fixed' in our r coordinate):

$$R_{fixed} = p_{top} = 0$$

In a resting atmosphere the elevation of the mountains at the bottom is given by

$$R_{moving} = R_o(x, y) = p_o(x, y)$$

i.e. the (hydrostatic) pressure at the top of the mountains in a resting atmosphere.

The boundary conditions at top and bottom are given by:

$$\omega = 0 \text{ at } r = R_{fixed} \text{ (top of the atmosphere)} \tag{1.17}$$

$$\omega = \frac{Dp_s}{Dt} \text{ at } r = R_{moving} \text{ (bottom of the atmosphere)} \tag{1.18}$$

Then the (hydrostatic form of) equations (1.1)-(1.6) yields a consistent set of atmospheric equations which, for convenience, are written out in $p-$coordinates in Section 1.4.1 - see eqs. (1.59)-(1.63).

1.3.3 Ocean

In the ocean we interpret:

$$r = z \text{ is the height} \tag{1.19}$$

$$\dot{r} = \frac{Dz}{Dt} = w \text{ is the vertical velocity} \tag{1.20}$$

$$\phi = \frac{p}{\rho_c} \text{ is the pressure} \tag{1.21}$$

$$b(\theta, S, r) = \frac{g}{\rho_c}\left(\rho(\theta, S, r) - \rho_c\right) \text{ is the buoyancy} \tag{1.22}$$

where ρ_c is a fixed reference density of water and g is the acceleration due to gravity.

In the above:

At the bottom of the ocean: $R_{fixed}(x, y) = -H(x, y)$.

The surface of the ocean is given by: $R_{moving} = \eta$

The position of the resting free surface of the ocean is given by $R_o = Z_o = 0$.

Boundary conditions are:

$$w = 0 \text{ at } r = R_{fixed} \text{ (ocean bottom)} \tag{1.23}$$

$$w = \frac{D\eta}{Dt} \text{ at } r = R_{moving} = \eta \text{ (ocean surface)} \tag{1.24}$$

where η is the elevation of the free surface.

Then equations (1.1)- (1.6) yield a consistent set of oceanic equations which, for convenience, are written out in $z-$coordinates in Section 1.5.1 - see eqs. (1.98) to (1.103).

1.3.4 Hydrostatic, Quasi-hydrostatic, Quasi-nonhydrostatic and Non-hydrostatic forms

Let us separate ϕ in to surface, hydrostatic and non-hydrostatic terms:

$$\phi(x,y,r) = \phi_s(x,y) + \phi_{hyd}(x,y,r) + \phi_{nh}(x,y,r) \tag{1.25}$$

and write (1.1) in the form:

$$\frac{\partial \vec{\mathbf{v}_h}}{\partial t} + \nabla_h \phi_s + \nabla_h \phi_{hyd} + \epsilon_{nh} \nabla_h \phi_{nh} = \vec{\mathbf{G}}_{\vec{v}_h} \tag{1.26}$$

$$\frac{\partial \phi_{hyd}}{\partial r} = -b \tag{1.27}$$

$$\epsilon_{nh} \frac{\partial \dot{r}}{\partial t} + \frac{\partial \phi_{nh}}{\partial r} = G_{\dot{r}} \tag{1.28}$$

Here ϵ_{nh} is a non-hydrostatic parameter.

The $\left(\vec{\mathbf{G}}_{\vec{v}}, G_{\dot{r}}\right)$ in (1.26) and (1.28) represent advective, metric and Coriolis terms in the momentum equations. In spherical coordinates they take the form[1] - see Marshall et al. (1997a) [MHPA97] for a full discussion:

$$\begin{aligned}
G_u = & -\vec{\mathbf{v}}.\nabla u && \text{advection} \\
& - \left\{ \underline{\frac{u\dot{r}}{r}} - \frac{uv\tan\varphi}{r} \right\} && \text{metric} \\
& - \left\{ -2\Omega v \sin\varphi + \underline{2\Omega \dot{r} \cos\varphi} \right\} && \text{Coriolis} \\
& + \mathcal{F}_u && \text{forcing/dissipation}
\end{aligned} \tag{1.29}$$

$$\begin{aligned}
G_v = & -\vec{\mathbf{v}}.\nabla v && \text{advection} \\
& - \left\{ \underline{\frac{v\dot{r}}{r}} - \frac{u^2\tan\varphi}{r} \right\} && \text{metric} \\
& - \{2\Omega u \sin\varphi\} && \text{Coriolis} \\
& + \mathcal{F}_v && \text{forcing/dissipation}
\end{aligned} \tag{1.30}$$

$$\begin{aligned}
G_{\dot{r}} = & -\underline{\underline{\vec{\mathbf{v}}.\nabla \dot{r}}} && \text{advection} \\
& - \left\{ \underline{\frac{u^2+v^2}{r}} \right\} && \text{metric} \\
& + \underline{2\Omega u \cos\varphi} && \text{Coriolis} \\
& + \underline{\underline{\mathcal{F}_{\dot{r}}}} && \text{forcing/dissipation}
\end{aligned} \tag{1.31}$$

In the above 'r' is the distance from the center of the earth and 'φ ' is latitude (see Figure 1.20).

Grad and div operators in spherical coordinates are defined in *Coordinate systems*.

1.3.4.1 Shallow atmosphere approximation

Most models are based on the 'hydrostatic primitive equations' (**HPE**'s) in which the vertical momentum equation is reduced to a statement of hydrostatic balance and the 'traditional approximation' is made in which the Coriolis force is treated approximately and the shallow atmosphere approximation is made. MITgcm need not make the 'traditional approximation'. To be able to support consistent non-hydrostatic forms the shallow atmosphere approximation can be relaxed - when dividing through by r in, for example, (1.29), we do not replace r by a, the radius of the earth.

[1] In the hydrostatic primitive equations (**HPE**) all underlined terms in (1.29), (1.30) and (1.31) are omitted; the singly-underlined terms are included in the quasi-hydrostatic model (**QH**). The fully non-hydrostatic model (**NH**) includes all terms.

1.3.4.2 Hydrostatic and quasi-hydrostatic forms

These are discussed at length in Marshall et al. (1997a) [MHPA97].

In the 'hydrostatic primitive equations' (**HPE**) all the underlined terms in Eqs. (1.29) → (1.31) are neglected and 'r' is replaced by 'a', the mean radius of the earth. Once the pressure is found at one level - e.g. by inverting a 2-d Elliptic equation for ϕ_s at $r = R_{moving}$ - the pressure can be computed at all other levels by integration of the hydrostatic relation, eq (1.27).

In the 'quasi-hydrostatic' equations (**QH**) strict balance between gravity and vertical pressure gradients is not imposed. The $2\Omega u \cos\varphi$ Coriolis term are not neglected and are balanced by a non-hydrostatic contribution to the pressure field: only the terms underlined twice in Eqs. (1.29) → (1.31) are set to zero and, simultaneously, the shallow atmosphere approximation is relaxed. In **QH** *all* the metric terms are retained and the full variation of the radial position of a particle monitored. The **QH** vertical momentum equation (1.28) becomes:

$$\frac{\partial \phi_{nh}}{\partial r} = 2\Omega u \cos\varphi$$

making a small correction to the hydrostatic pressure.

QH has good energetic credentials - they are the same as for **HPE**. Importantly, however, it has the same angular momentum principle as the full non-hydrostatic model (**NH**) - see Marshall et.al. (1997a) [MHPA97]. As in **HPE** only a 2-d elliptic problem need be solved.

1.3.4.3 Non-hydrostatic and quasi-nonhydrostatic forms

MITgcm presently supports a full non-hydrostatic ocean isomorph, but only a quasi-non-hydrostatic atmospheric isomorph.

Non-hydrostatic Ocean

In the non-hydrostatic ocean model all terms in equations Eqs. (1.29) → (1.31) are retained. A three dimensional elliptic equation must be solved subject to Neumann boundary conditions (see below). It is important to note that use of the full **NH** does not admit any new 'fast' waves in to the system - the incompressible condition (1.3) has already filtered out acoustic modes. It does, however, ensure that the gravity waves are treated accurately with an exact dispersion relation. The **NH** set has a complete angular momentum principle and consistent energetics - see White and Bromley (1995) [WB95]; Marshall et al. (1997a) [MHPA97].

Quasi-nonhydrostatic Atmosphere

In the non-hydrostatic version of our atmospheric model we approximate $\dot{r}$ in the vertical momentum eqs. (1.28) and (1.30) (but only here) by:

$$\dot{r} = \frac{Dp}{Dt} = \frac{1}{g}\frac{D\phi}{Dt} \tag{1.32}$$

where p_{hy} is the hydrostatic pressure.

1.3.4.4 Summary of equation sets supported by model

Atmosphere

Hydrostatic, and quasi-hydrostatic and quasi non-hydrostatic forms of the compressible non-Boussinesq equations in $p-$coordinates are supported.

Hydrostatic and quasi-hydrostatic

The hydrostatic set is written out in $p-$coordinates in *Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates* - see eqs. (1.59) to (1.63).

Quasi-nonhydrostatic

A quasi-nonhydrostatic form is also supported.

Ocean

Hydrostatic and quasi-hydrostatic

Hydrostatic, and quasi-hydrostatic forms of the incompressible Boussinesq equations in $z-$coordinates are supported.

Non-hydrostatic

Non-hydrostatic forms of the incompressible Boussinesq equations in $z-$ coordinates are supported - see eqs. (1.98) to (1.103).

1.3.5 Solution strategy

The method of solution employed in the **HPE**, **QH** and **NH** models is summarized in Figure 1.19. Under all dynamics, a 2-d elliptic equation is first solved to find the surface pressure and the hydrostatic pressure at any level computed from the weight of fluid above. Under **HPE** and **QH** dynamics, the horizontal momentum equations are then stepped forward and $\dot{r}$ found from continuity. Under **NH** dynamics a 3-d elliptic equation must be solved for the non-hydrostatic pressure before stepping forward the horizontal momentum equations; $\dot{r}$ is found by stepping forward the vertical momentum equation.

There is no penalty in implementing **QH** over **HPE** except, of course, some complication that goes with the inclusion of $\cos\varphi$ Coriolis terms and the relaxation of the shallow atmosphere approximation. But this leads to negligible increase in computation. In **NH**, in contrast, one additional elliptic equation - a three-dimensional one - must be inverted for p_{nh}. However the 'overhead' of the **NH** model is essentially negligible in the hydrostatic limit (see detailed discussion in Marshall et al. (1997) [MHPA97] resulting in a non-hydrostatic algorithm that, in the hydrostatic limit, is as computationally economic as the **HPEs**.

1.3.6 Finding the pressure field

Unlike the prognostic variables u, v, w, θ and S, the pressure field must be obtained diagnostically. We proceed, as before, by dividing the total (pressure/geo) potential in to three parts, a surface part, $\phi_s(x,y)$, a hydrostatic part $\phi_{hyd}(x,y,r)$ and a non-hydrostatic part $\phi_{nh}(x,y,r)$, as in (1.25), and writing the momentum equation as in (1.26).

1.3.6.1 Hydrostatic pressure

Hydrostatic pressure is obtained by integrating (1.27) vertically from $r = R_o$ where $\phi_{hyd}(r = R_o) = 0$, to yield:

$$\int_r^{R_o} \frac{\partial \phi_{hyd}}{\partial r} dr = [\phi_{hyd}]_r^{R_o} = \int_r^{R_o} -b dr$$

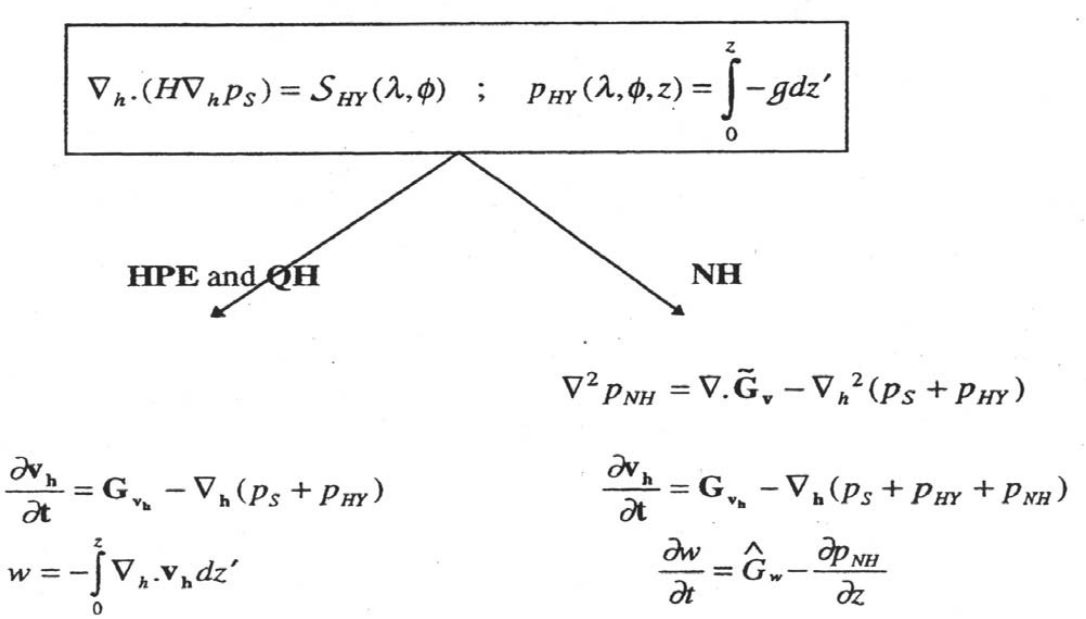

Figure 1.19: Basic solution strategy in MITgcm. **HPE** and **QH** forms diagnose the vertical velocity, in **NH** a prognostic equation for the vertical velocity is integrated.

and so

$$\phi_{hyd}(x,y,r) = \int_r^{R_o} b dr \tag{1.33}$$

The model can be easily modified to accommodate a loading term (e.g atmospheric pressure pushing down on the ocean's surface) by setting:

$$\phi_{hyd}(r = R_o) = loading \tag{1.34}$$

1.3.6.2 Surface pressure

The surface pressure equation can be obtained by integrating continuity, (1.3), vertically from $r = R_{fixed}$ to $r = R_{moving}$

$$\int_{R_{fixed}}^{R_{moving}} (\nabla_h \cdot \vec{\mathbf{v}}_h + \partial_r \dot{r})\, dr = 0$$

Thus:

$$\frac{\partial \eta}{\partial t} + \vec{\mathbf{v}}.\nabla\eta + \int_{R_{fixed}}^{R_{moving}} \nabla_h \cdot \vec{\mathbf{v}}_h dr = 0$$

where $\eta = R_{moving} - R_o$ is the free-surface r-anomaly in units of r. The above can be rearranged to yield, using Leibnitz's theorem:

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \int_{R_{fixed}}^{R_{moving}} \vec{\mathbf{v}}_h dr = \text{source} \tag{1.35}$$

where we have incorporated a source term.

Whether ϕ is pressure (ocean model, p/ρ_c) or geopotential (atmospheric model), in (1.26), the horizontal gradient term can be written

$$\nabla_h \phi_s = \nabla_h \left(b_s \eta \right) \tag{1.36}$$

where b_s is the buoyancy at the surface.

In the hydrostatic limit ($\epsilon_{nh} = 0$), equations (1.26), (1.35) and (1.36) can be solved by inverting a 2-d elliptic equation for ϕ_s as described in Chapter 2. Both 'free surface' and 'rigid lid' approaches are available.

1.3.6.3 Non-hydrostatic pressure

Taking the horizontal divergence of (1.26) and adding $\frac{\partial}{\partial r}$ of (1.28), invoking the continuity equation (1.3), we deduce that:

$$\nabla^2_3 \phi_{nh} = \nabla . \vec{\mathbf{G}}_{\vec{v}} - \left(\nabla^2_h \phi_s + \nabla^2 \phi_{hyd} \right) = \nabla . \vec{\mathbf{F}} \tag{1.37}$$

For a given rhs this 3-d elliptic equation must be inverted for ϕ_{nh} subject to appropriate choice of boundary conditions. This method is usually called *The Pressure Method* [Harlow and Welch (1965) [HW65]; Williams (1969) [Wil69]; Potter (1973) [Pot73]. In the hydrostatic primitive equations case (**HPE**), the 3-d problem does not need to be solved.

Boundary Conditions

We apply the condition of no normal flow through all solid boundaries - the coasts (in the ocean) and the bottom:

$$\vec{\mathbf{v}} . \widehat{n} = 0 \tag{1.38}$$

where $\widehat{n}$ is a vector of unit length normal to the boundary. The kinematic condition (1.38) is also applied to the vertical velocity at $r = R_{moving}$. No-slip ($v_T = 0$) or slip ($\partial v_T / \partial n = 0$) conditions are employed on the tangential component of velocity, v_T, at all solid boundaries, depending on the form chosen for the dissipative terms in the momentum equations - see below.

Eq. (1.38) implies, making use of (1.26), that:

$$\widehat{n} . \nabla \phi_{nh} = \widehat{n} . \vec{\mathbf{F}} \tag{1.39}$$

where

$$\vec{\mathbf{F}} = \vec{\mathbf{G}}_{\vec{v}} - \left(\nabla_h \phi_s + \nabla \phi_{hyd} \right)$$

presenting inhomogeneous Neumann boundary conditions to the Elliptic problem (1.37). As shown, for example, by Williams (1969) [Wil69], one can exploit classical 3D potential theory and, by introducing an appropriately chosen δ-function sheet of 'source-charge', replace the inhomogeneous boundary condition on pressure by a homogeneous one. The source term rhs in (1.37) is the divergence of the vector $\vec{\mathbf{F}}$. By simultaneously setting $\widehat{n} . \vec{\mathbf{F}} = 0$ and $\widehat{n} . \nabla \phi_{nh} = 0$ on the boundary the following self-consistent but simpler homogenized Elliptic problem is obtained:

$$\nabla^2 \phi_{nh} = \nabla . \widetilde{\vec{\mathbf{F}}}$$

where $\widetilde{\vec{\mathbf{F}}}$ is a modified $\vec{\mathbf{F}}$ such that $\widetilde{\vec{\mathbf{F}}} . \widehat{n} = 0$. As is implied by (1.39) the modified boundary condition becomes:

$$\widehat{n} . \nabla \phi_{nh} = 0 \tag{1.40}$$

If the flow is 'close' to hydrostatic balance then the 3-d inversion converges rapidly because ϕ_{nh} is then only a small correction to the hydrostatic pressure field (see the discussion in Marshall et al. (1997a,b) [MHPA97] [MAH+97].

The solution ϕ_{nh} to (1.37) and (1.39) does not vanish at $r = R_{moving}$, and so refines the pressure there.

1.3.7 Forcing/dissipation

1.3.7.1 Forcing

The forcing terms $\mathcal{F}$ on the rhs of the equations are provided by 'physics packages' and forcing packages. These are described later on.

1.3.7.2 Dissipation

Momentum

Many forms of momentum dissipation are available in the model. Laplacian and biharmonic frictions are commonly used:

$$D_V = A_h \nabla_h^2 v + A_v \frac{\partial^2 v}{\partial z^2} + A_4 \nabla_h^4 v \tag{1.41}$$

where A_h and A_v are (constant) horizontal and vertical viscosity coefficients and A_4 is the horizontal coefficient for biharmonic friction. These coefficients are the same for all velocity components.

Tracers

The mixing terms for the temperature and salinity equations have a similar form to that of momentum except that the diffusion tensor can be non-diagonal and have varying coefficients.

$$D_{T,S} = \nabla.[\underline{\underline{K}} \nabla(T,S)] + K_4 \nabla_h^4 (T,S) \tag{1.42}$$

where $\underline{\underline{K}}$ is the diffusion tensor and the K_4 horizontal coefficient for biharmonic diffusion. In the simplest case where the subgrid-scale fluxes of heat and salt are parameterized with constant horizontal and vertical diffusion coefficients, $\underline{\underline{K}}$, reduces to a diagonal matrix with constant coefficients:

$$K = \begin{pmatrix} K_h & 0 & 0 \\ 0 & K_h & 0 \\ 0 & 0 & K_v \end{pmatrix} \tag{1.43}$$

where K_h and K_v are the horizontal and vertical diffusion coefficients. These coefficients are the same for all tracers (temperature, salinity ...).

1.3.8 Vector invariant form

For some purposes it is advantageous to write momentum advection in eq (1.1) and (1.2) in the (so-called) 'vector invariant' form:

$$\frac{D\vec{\mathbf{v}}}{Dt} = \frac{\partial \vec{\mathbf{v}}}{\partial t} + (\nabla \times \vec{\mathbf{v}}) \times \vec{\mathbf{v}} + \nabla \left[\frac{1}{2} (\vec{\mathbf{v}} \cdot \vec{\mathbf{v}}) \right] \tag{1.44}$$

This permits alternative numerical treatments of the non-linear terms based on their representation as a vorticity flux. Because gradients of coordinate vectors no longer appear on the rhs of (1.44), explicit representation of the metric terms in (1.29), (1.30) and (1.31), can be avoided: information about the geometry is contained in the areas and lengths of the volumes used to discretize the model.

1.3.9 Adjoint

Tangent linear and adjoint counterparts of the forward model are described in Section 7.

1.4 Appendix ATMOSPHERE

1.4.1 Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates

The hydrostatic primitive equations (**HPE**'s) in $p-$coordinates are:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \nabla_p \phi = \vec{\mathcal{F}} \tag{1.45}$$

$$\frac{\partial \phi}{\partial p} + \alpha = 0 \tag{1.46}$$

$$\nabla_p \cdot \vec{\mathbf{v}}_h + \frac{\partial \omega}{\partial p} = 0 \tag{1.47}$$

$$p\alpha = RT \tag{1.48}$$

$$c_v \frac{DT}{Dt} + p\frac{D\alpha}{Dt} = \mathcal{Q} \tag{1.49}$$

where $\vec{\mathbf{v}}_h = (u, v, 0)$ is the 'horizontal' (on pressure surfaces) component of velocity, $\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{\mathbf{v}}_h \cdot \nabla_p + \omega \frac{\partial}{\partial p}$ is the total derivative, $f = 2\Omega \sin\varphi$ is the Coriolis parameter, $\phi = gz$ is the geopotential, $\alpha = 1/\rho$ is the specific volume, $\omega = \frac{Dp}{Dt}$ is the vertical velocity in the $p-$coordinate. Equation (1.49) is the first law of thermodynamics where internal energy $e = c_v T$, T is temperature, Q is the rate of heating per unit mass and $p\frac{D\alpha}{Dt}$ is the work done by the fluid in compressing.

It is convenient to cast the heat equation in terms of potential temperature θ so that it looks more like a generic conservation law. Differentiating (1.48) we get:

$$p\frac{D\alpha}{Dt} + \alpha\frac{Dp}{Dt} = R\frac{DT}{Dt}$$

which, when added to the heat equation (1.49) and using $c_p = c_v + R$, gives:

$$c_p \frac{DT}{Dt} - \alpha\frac{Dp}{Dt} = \mathcal{Q} \tag{1.50}$$

Potential temperature is defined:

$$\theta = T(\frac{p_c}{p})^\kappa \tag{1.51}$$

where p_c is a reference pressure and $\kappa = R/c_p$. For convenience we will make use of the Exner function $\Pi(p)$ which is defined by:

$$\Pi(p) = c_p(\frac{p}{p_c})^\kappa \tag{1.52}$$

The following relations will be useful and are easily expressed in terms of the Exner function:

$$c_p T = \Pi\theta \;\; ; \;\; \frac{\partial \Pi}{\partial p} = \frac{\kappa\Pi}{p} \;\; ; \;\; \alpha = \frac{\kappa\Pi\theta}{p} = \frac{\partial\, \Pi}{\partial p}\theta \;\; ; \;\; \frac{D\Pi}{Dt} = \frac{\partial \Pi}{\partial p}\frac{Dp}{Dt}$$

where $b = \frac{\partial\, \Pi}{\partial p}\theta$ is the buoyancy.

The heat equation is obtained by noting that

$$c_p \frac{DT}{Dt} = \frac{D(\Pi\theta)}{Dt} = \Pi\frac{D\theta}{Dt} + \theta\frac{D\Pi}{Dt} = \Pi\frac{D\theta}{Dt} + \alpha\frac{Dp}{Dt}$$

and on substituting into (1.50) gives:

$$\Pi\frac{D\theta}{Dt} = \mathcal{Q} \tag{1.53}$$

which is in conservative form.

For convenience in the model we prefer to step forward (1.53) rather than (1.49).

1.4.1.1 Boundary conditions

The upper and lower boundary conditions are:

$$\text{at the top: } p = 0, \omega = \frac{Dp}{Dt} = 0 \tag{1.54}$$

$$\text{at the surface: } p = p_s, \phi = \phi_{topo} = g\, Z_{topo} \tag{1.55}$$

In $p-$coordinates, the upper boundary acts like a solid boundary ($\omega = 0$); in $z-$coordinates the lower boundary is analogous to a free surface (ϕ is imposed and $\omega \neq 0$).

1.4.1.2 Splitting the geopotential

For the purposes of initialization and reducing round-off errors, the model deals with perturbations from reference (or 'standard') profiles. For example, the hydrostatic geopotential associated with the resting atmosphere is not dynamically relevant and can therefore be subtracted from the equations. The equations written in terms of perturbations are obtained by substituting the following definitions into the previous model equations:

$$\theta = \theta_o + \theta' \tag{1.56}$$

$$\alpha = \alpha_o + \alpha' \tag{1.57}$$

$$\phi = \phi_o + \phi' \tag{1.58}$$

The reference state (indicated by subscript 'o') corresponds to horizontally homogeneous atmosphere at rest $(\theta_o, \alpha_o, \phi_o)$ with surface pressure $p_o(x, y)$ that satisfies $\phi_o(p_o) = g\, Z_{topo}$, defined:

$$\theta_o(p) = f^n(p)$$

$$\alpha_o(p) = \Pi_p \theta_o$$

$$\phi_o(p) = \phi_{topo} - \int_{p_0}^{p} \alpha_o dp$$

The final form of the **HPE**'s in $p-$coordinates is then:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \nabla_p \phi' = \vec{\mathcal{F}} \tag{1.59}$$

$$\frac{\partial \phi'}{\partial p} + \alpha' = 0 \tag{1.60}$$

$$\nabla_p \cdot \vec{\mathbf{v}}_h + \frac{\partial \omega}{\partial p} = 0 \tag{1.61}$$

$$\frac{\partial \Pi}{\partial p}\theta' = \alpha' \tag{1.62}$$

$$\frac{D\theta}{Dt} = \frac{\mathcal{Q}}{\Pi} \tag{1.63}$$

1.5 Appendix OCEAN

1.5.1 Equations of Motion for the Ocean

We review here the method by which the standard (Boussinesq, incompressible) HPE's for the ocean written in $z-$coordinates are obtained. The non-Boussinesq equations for oceanic motion are:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho}\nabla_z p = \vec{\mathcal{F}} \tag{1.64}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + g + \frac{1}{\rho}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.65}$$

$$\frac{1}{\rho}\frac{D\rho}{Dt} + \nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.66}$$

$$\rho = \rho(\theta, S, p) \tag{1.67}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.68}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.69}$$

These equations permit acoustics modes, inertia-gravity waves, non-hydrostatic motions, a geostrophic (Rossby) mode and a thermohaline mode. As written, they cannot be integrated forward consistently - if we step ρ forward in (1.66), the answer will not be consistent with that obtained by stepping (1.68) and (1.69) and then using (1.67) to yield ρ. It is therefore necessary to manipulate the system as follows. Differentiating the EOS (equation of state) gives:

$$\frac{D\rho}{Dt} = \left.\frac{\partial\rho}{\partial\theta}\right|_{S,p}\frac{D\theta}{Dt} + \left.\frac{\partial\rho}{\partial S}\right|_{\theta,p}\frac{DS}{Dt} + \left.\frac{\partial\rho}{\partial p}\right|_{\theta,S}\frac{Dp}{Dt} \tag{1.70}$$

Note that $\frac{\partial\rho}{\partial p} = \frac{1}{c_s^2}$ is the reciprocal of the sound speed (c_s) squared. Substituting into (1.66) gives:

$$\frac{1}{\rho c_s^2}\frac{Dp}{Dt} + \nabla_z \cdot \vec{\mathbf{v}} + \partial_z w \approx 0 \tag{1.71}$$

where we have used an approximation sign to indicate that we have assumed adiabatic motion, dropping the $\frac{D\theta}{Dt}$ and $\frac{DS}{Dt}$. Replacing (1.66) with (1.71) yields a system that can be explicitly integrated forward:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho}\nabla_z p = \vec{\mathcal{F}} \tag{1.72}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + g + \frac{1}{\rho}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.73}$$

$$\frac{1}{\rho c_s^2}\frac{Dp}{Dt} + \nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.74}$$

$$\rho = \rho(\theta, S, p) \tag{1.75}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.76}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.77}$$

1.5.1.1 Compressible z-coordinate equations

Here we linearize the acoustic modes by replacing ρ with $\rho_o(z)$ wherever it appears in a product (ie. non-linear term) - this is the 'Boussinesq assumption'. The only term that then retains the full variation in ρ is the gravitational acceleration:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_o}\nabla_z p = \vec{\mathcal{F}} \tag{1.78}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho}{\rho_o} + \frac{1}{\rho_o}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.79}$$

$$\frac{1}{\rho_o c_s^2}\frac{Dp}{Dt} + \nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.80}$$

$$\rho = \rho(\theta, S, p) \tag{1.81}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.82}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.83}$$

These equations still retain acoustic modes. But, because the "compressible" terms are linearized, the pressure equation (1.80) can be integrated implicitly with ease (the time-dependent term appears as a Helmholtz term in the non-hydrostatic pressure equation). These are the *truly* compressible Boussinesq equations. Note that the EOS must have the same pressure dependency as the linearized pressure term, ie. $\left.\frac{\partial\rho}{\partial p}\right|_{\theta,S} = \frac{1}{c_s^2}$, for consistency.

1.5.1.2 'Anelastic' z-coordinate equations

The anelastic approximation filters the acoustic mode by removing the time-dependency in the continuity (now pressure-) equation (1.80). This could be done simply by noting that $\frac{Dp}{Dt} \approx -g\rho_o\frac{Dz}{Dt} = -g\rho_o w$, but this leads to an inconsistency between continuity and EOS. A better solution is to change the dependency on pressure in the EOS by splitting the pressure into a reference function of height and a perturbation:

$$\rho = \rho(\theta, S, p_o(z) + \epsilon_s p')$$

Remembering that the term $\frac{Dp}{Dt}$ in continuity comes from differentiating the EOS, the continuity equation then becomes:

$$\frac{1}{\rho_o c_s^2}\left(\frac{Dp_o}{Dt} + \epsilon_s\frac{Dp'}{Dt}\right) + \nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0$$

If the time- and space-scales of the motions of interest are longer than those of acoustic modes, then $\frac{Dp'}{Dt} << (\frac{Dp_o}{Dt}, \nabla\cdot\vec{\mathbf{v}}_h)$ in the continuity equations and $\left.\frac{\partial\rho}{\partial p}\right|_{\theta,S}\frac{Dp'}{Dt} << \left.\frac{\partial\rho}{\partial p}\right|_{\theta,S}\frac{Dp_o}{Dt}$ in the EOS (1.70). Thus we set $\epsilon_s = 0$, removing the dependency on p' in the continuity equation and EOS. Expanding $\frac{Dp_o(z)}{Dt} = -g\rho_o w$ then leads to the anelastic continuity equation:

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} - \frac{g}{c_s^2}w = 0 \tag{1.84}$$

A slightly different route leads to the quasi-Boussinesq continuity equation where we use the scaling $\frac{\partial\rho'}{\partial t} + \nabla_3\cdot\rho'\vec{\mathbf{v}} << \nabla_3 \cdot \rho_o\vec{\mathbf{v}}$ yielding:

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{1}{\rho_o}\frac{\partial\left(\rho_o w\right)}{\partial z} = 0 \tag{1.85}$$

Equations (1.84) and (1.85) are in fact the same equation if:

$$\frac{1}{\rho_o}\frac{\partial \rho_o}{\partial z} = \frac{-g}{c_s^2}$$

Again, note that if ρ_o is evaluated from prescribed θ_o and S_o profiles, then the EOS dependency on p_o and the term $\frac{g}{c_s^2}$ in continuity should be referred to those same profiles. The full set of 'quasi-Boussinesq' or 'anelastic' equations for the ocean are then:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_o}\nabla_z p = \vec{\mathcal{F}} \tag{1.86}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho}{\rho_o} + \frac{1}{\rho_o}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.87}$$

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{1}{\rho_o}\frac{\partial\left(\rho_o w\right)}{\partial z} = 0 \tag{1.88}$$

$$\rho = \rho(\theta, S, p_o(z)) \tag{1.89}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.90}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.91}$$

1.5.1.3 Incompressible z-coordinate equations

Here, the objective is to drop the depth dependence of ρ_o and so, technically, to also remove the dependence of ρ on p_o. This would yield the "truly" incompressible Boussinesq equations:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_c}\nabla_z p = \vec{\mathcal{F}} \tag{1.92}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho}{\rho_c} + \frac{1}{\rho_c}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.93}$$

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.94}$$

$$\rho = \rho(\theta, S) \tag{1.95}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.96}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.97}$$

where ρ_c is a constant reference density of water.

1.5.1.4 Compressible non-divergent equations

The above "incompressible" equations are incompressible in both the flow and the density. In many oceanic applications, however, it is important to retain compressibility effects in the density. To do this we must split the density thus:

$$\rho = \rho_o + \rho'$$

We then assert that variations with depth of ρ_o are unimportant while the compressible effects in ρ' are:

$$\rho_o = \rho_c$$

$$\rho' = \rho(\theta, S, p_o(z)) - \rho_o$$

This then yields what we can call the semi-compressible Boussinesq equations:

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_c}\nabla_z p' = \vec{\mathcal{F}} \tag{1.98}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho'}{\rho_c} + \frac{1}{\rho_c}\frac{\partial p'}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.99}$$

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.100}$$

$$\rho' = \rho(\theta, S, p_o(z)) - \rho_c \tag{1.101}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.102}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.103}$$

Note that the hydrostatic pressure of the resting fluid, including that associated with ρ_c, is subtracted out since it has no effect on the dynamics.

Though necessary, the assumptions that go into these equations are messy since we essentially assume a different EOS for the reference density and the perturbation density. Nevertheless, it is the hydrostatic ($\epsilon_{nh} = 0$) form of these equations that are used throughout the ocean modeling community and referred to as the primitive equations (**HPE**'s).

1.6 Appendix OPERATORS

1.6.1 Coordinate systems

1.6.1.1 Spherical coordinates

In spherical coordinates, the velocity components in the zonal, meridional and vertical direction respectively, are given by:

$$u = r\cos\varphi\frac{D\lambda}{Dt}$$

$$v = r\frac{D\varphi}{Dt}$$

$$\dot{r} = \frac{Dr}{Dt}$$

(see Figure 1.20) Here φ is the latitude, λ the longitude, r the radial distance of the particle from the center of the earth, Ω is the angular speed of rotation of the Earth and D/Dt is the total derivative.

The 'grad' (∇) and 'div' ($\nabla\cdot$) operators are defined by, in spherical coordinates:

$$\nabla \equiv \left(\frac{1}{r\cos\varphi}\frac{\partial}{\partial\lambda}, \frac{1}{r}\frac{\partial}{\partial\varphi}, \frac{\partial}{\partial r}\right)$$

$$\nabla \cdot v \equiv \frac{1}{r\cos\varphi}\left\{\frac{\partial u}{\partial\lambda} + \frac{\partial}{\partial\varphi}\left(v\cos\varphi\right)\right\} + \frac{1}{r^2}\frac{\partial\left(r^2\dot{r}\right)}{\partial r}$$

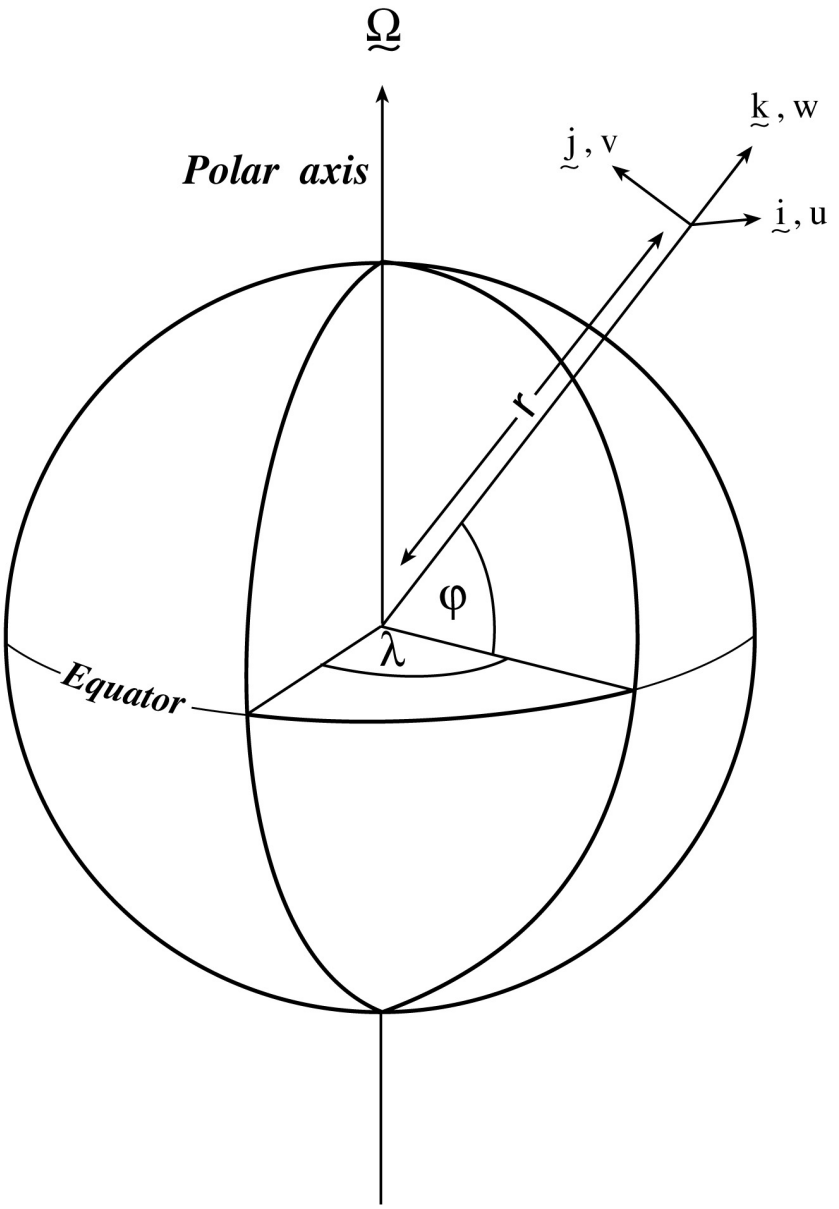

Figure 1.20: Spherical polar coordinates: longitude λ, latitude φ and r the distance from the center.

CHAPTER 2

Discretization and Algorithm

This chapter lays out the numerical schemes that are employed in the core MITgcm algorithm. Whenever possible links are made to actual program code in the MITgcm implementation. The chapter begins with a discussion of the temporal discretization used in MITgcm. This discussion is followed by sections that describe the spatial discretization. The schemes employed for momentum terms are described first, afterwards the schemes that apply to passive and dynamically active tracers are described.

2.1 Notation

Because of the particularity of the vertical direction in stratified fluid context, in this chapter, the vector notations are mostly used for the horizontal component: the horizontal part of a vector is simply written $\vec{\mathbf{v}}$ (instead of $\mathbf{v_h}$ or $\vec{\mathbf{v}}_h$ in chapter 1) and a 3D vector is simply written $\vec{v}$ (instead of $\vec{\mathbf{v}}$ in chapter 1).

The notations we use to describe the discrete formulation of the model are summarized as follows.

General notation:

$\Delta x, \Delta y, \Delta r$ grid spacing in X, Y, R directions

A_c, A_w, A_s, A_ζ : horizontal area of a grid cell surrounding θ, u, v, ζ point

$\mathcal{V}_u, \mathcal{V}_v, \mathcal{V}_w, \mathcal{V}_\theta$: Volume of the grid box surrounding u, v, w, θ point

i, j, k : current index relative to X, Y, R directions

Basic operators:

$\delta_i : \delta_i \Phi = \Phi_{i+1/2} - \Phi_{i-1/2}$

$^{-i} : \overline{\Phi}^i = (\Phi_{i+1/2} + \Phi_{i-1/2})/2$

$\delta_x : \delta_x \Phi = \frac{1}{\Delta x} \delta_i \Phi$

$\overline{\nabla}$ = horizontal gradient operator : $\overline{\nabla}\Phi = \{\delta_x \Phi, \delta_y \Phi\}$

$\overline{\nabla}\cdot$ = horizontal divergence operator : $\overline{\nabla} \cdot \vec{\mathrm{f}} = \frac{1}{\mathcal{A}} \{\delta_i \Delta y \, \mathrm{f}_x + \delta_j \Delta x \, \mathrm{f}_y\}$

$\overline{\nabla}^2$ = horizontal Laplacian operator : $\overline{\nabla}^2 \Phi = \overline{\nabla} \cdot \overline{\nabla} \Phi$

2.2 Time-stepping

The equations of motion integrated by the model involve four prognostic equations for flow, u and v, temperature, θ, and salt/moisture, S, and three diagnostic equations for vertical flow, w, density/buoyancy, ρ/b, and pressure/geopotential, ϕ_{hyd}. In addition, the surface pressure or height may by described by either a prognostic or diagnostic equation and if non-hydrostatics terms are included then a diagnostic equation for non-hydrostatic pressure is also solved. The combination of prognostic and diagnostic equations requires a model algorithm that can march forward prognostic variables while satisfying constraints imposed by diagnostic equations.

Since the model comes in several flavors and formulation, it would be confusing to present the model algorithm exactly as written into code along with all the switches and optional terms. Instead, we present the algorithm for each of the basic formulations which are:

1. the semi-implicit pressure method for hydrostatic equations with a rigid-lid, variables co-located in time and with Adams-Bashforth time-stepping;
2. as 1 but with an implicit linear free-surface;
3. as 1 or 2 but with variables staggered in time;
4. as 1 or 2 but with non-hydrostatic terms included;
5. as 2 or 3 but with non-linear free-surface.

In all the above configurations it is also possible to substitute the Adams-Bashforth with an alternative time-stepping scheme for terms evaluated explicitly in time. Since the over-arching algorithm is independent of the particular time-stepping scheme chosen we will describe first the over-arching algorithm, known as the pressure method, with a rigid-lid model in Section 2.3. This algorithm is essentially unchanged, apart for some coefficients, when the rigid lid assumption is replaced with a linearized implicit free-surface, described in Section 2.4. These two flavors of the pressure-method encompass all formulations of the model as it exists today. The integration of explicit in time terms is out-lined in Section 2.5 and put into the context of the overall algorithm in Section 2.7 and Section 2.8. Inclusion of non-hydrostatic terms requires applying the pressure method in three dimensions instead of two and this algorithm modification is described in Section 2.9. Finally, the free-surface equation may be treated more exactly, including non-linear terms, and this is described in Section 2.10.2.

2.3 Pressure method with rigid-lid

The horizontal momentum and continuity equations for the ocean ((1.98) and (1.100)), or for the atmosphere ((1.45) and (1.47)), can be summarized by:

$$\begin{aligned} \partial_t u + g\partial_x \eta = &\quad G_u \\ \partial_t v + g\partial_y \eta = &\quad G_v \\ \partial_x u + \partial_y v + \partial_z w = &\quad 0 \end{aligned}$$

where we are adopting the oceanic notation for brevity. All terms in the momentum equations, except for surface pressure gradient, are encapsulated in the G vector. The continuity equation, when integrated over the fluid depth, H, and with the rigid-lid/no normal flow boundary conditions applied, becomes:

$$\partial_x H\widehat{u} + \partial_y H\widehat{v} = 0 \tag{2.1}$$

Here, $H\widehat{u} = \int_H u dz$ is the depth integral of u, similarly for $H\widehat{v}$. The rigid-lid approximation sets $w = 0$ at the lid so that it does not move but allows a pressure to be exerted on the fluid by the lid. The horizontal momentum equations and vertically integrated continuity equation are be discretized in time and space as follows:

$$u^{n+1} + \Delta t g \partial_x \eta^{n+1} = u^n + \Delta t G_u^{(n+1/2)} \tag{2.2}$$

$$v^{n+1} + \Delta t g \partial_y \eta^{n+1} = v^n + \Delta t G_v^{(n+1/2)} \tag{2.3}$$

$$\partial_x H\widehat{u^{n+1}} + \partial_y H\widehat{v^{n+1}} = 0 \tag{2.4}$$

As written here, terms on the LHS all involve time level $n+1$ and are referred to as implicit; the implicit backward time stepping scheme is being used. All other terms in the RHS are explicit in time. The thermodynamic quantities are integrated forward in time in parallel with the flow and will be discussed later. For the purposes of describing the pressure method it suffices to say that the hydrostatic pressure gradient is explicit and so can be included in the vector G.

Substituting the two momentum equations into the depth integrated continuity equation eliminates u^{n+1} and v^{n+1} yielding an elliptic equation for η^{n+1}. Equations (2.2), (2.3) and (2.4) can then be re-arranged as follows:

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \tag{2.5}$$

$$v^* = v^n + \Delta t G_v^{(n+1/2)} \tag{2.6}$$

$$\partial_x \Delta t g H \partial_x \eta^{n+1} + \partial_y \Delta t g H \partial_y \eta^{n+1} = \partial_x H\widehat{u^*} + \partial_y H\widehat{v^*} \tag{2.7}$$

$$u^{n+1} = u^* - \Delta t g \partial_x \eta^{n+1} \tag{2.8}$$

$$v^{n+1} = v^* - \Delta t g \partial_y \eta^{n+1} \tag{2.9}$$

Equations (2.5) to (2.9), solved sequentially, represent the pressure method algorithm used in the model. The essence of the pressure method lies in the fact that any explicit prediction for the flow would lead to a divergence flow field so a pressure field must be found that keeps the flow non-divergent over each step of the integration. The particular location in time of the pressure field is somewhat ambiguous; in Figure 2.1 we depicted as co-located with the future flow field (time level $n+1$) but it could equally have been drawn as staggered in time with the flow.

The correspondence to the code is as follows:

- the prognostic phase, equations (2.5) and (2.6), stepping forward u^n and v^n to u^* and v^* is coded in timestep.F
- the vertical integration, $H\widehat{u^*}$ and $H\widehat{v^*}$, divergence and inversion of the elliptic operator in equation (2.7) is coded in solve_for_pressure.F

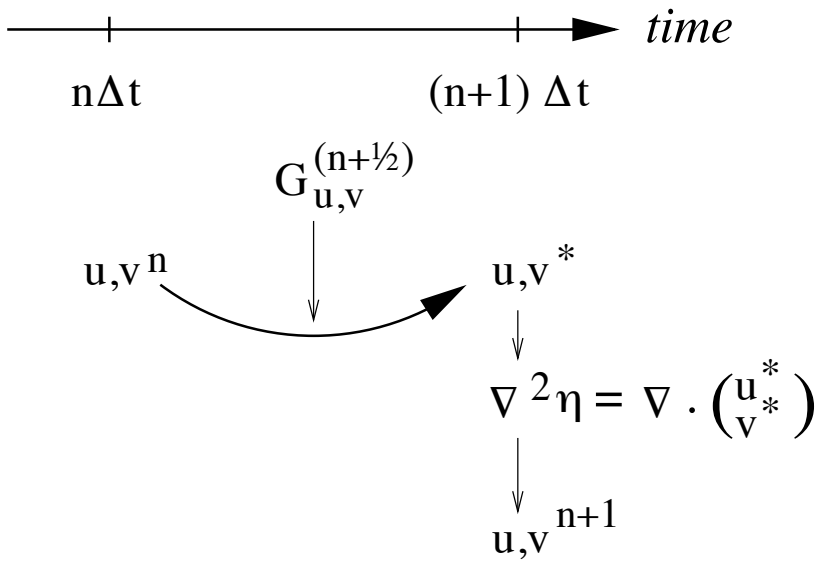

Figure 2.1: A schematic of the evolution in time of the pressure method algorithm. A prediction for the flow variables at time level $n+1$ is made based only on the explicit terms, $G^{(n+1/2)}$, and denoted u^*, v^*. Next, a pressure field is found such that u^{n+1}, v^{n+1} will be non-divergent. Conceptually, the $*$ quantities exist at time level $n+1$ but they are intermediate and only temporary.

- finally, the new flow field at time level $n+1$ given by equations (2.8) and (2.9) is calculated in correction_step.F

The calling tree for these routines is as follows:

Pressure method calling tree

FORWARD_STEP	
DYNAMICS	
TIMESTEP	u^*, v^* (2.5) , (2.6)
SOLVE_FOR_PRESSURE	
CALC_DIV_GHAT	$H\widehat{u^*}, H\widehat{v^*}$ (2.7)
CG2D	η^{n+1} (2.7)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla\eta^{n+1}$
CORRECTION_STEP	u^{n+1}, v^{n+1} (2.8) , (2.9)

In general, the horizontal momentum time-stepping can contain some terms that are treated implicitly in time, such as the vertical viscosity when using the backward time-stepping scheme (implicitViscosity =.TRUE.). The method used to solve those implicit terms is provided in Section 2.6, and modifies equations (2.2) and (2.3) to give:

$$
\begin{aligned}
u^{n+1} - \Delta t \partial_z A_v \partial_z u^{n+1} + \Delta t g \partial_x \eta^{n+1} &= u^n + \Delta t G_u^{(n+1/2)} \\
v^{n+1} - \Delta t \partial_z A_v \partial_z v^{n+1} + \Delta t g \partial_y \eta^{n+1} &= v^n + \Delta t G_v^{(n+1/2)}
\end{aligned}
$$

2.4 Pressure method with implicit linear free-surface

The rigid-lid approximation filters out external gravity waves subsequently modifying the dispersion relation of barotropic Rossby waves. The discrete form of the elliptic equation has some zero eigenvalues which makes it a potentially tricky or inefficient problem to solve.

The rigid-lid approximation can be easily replaced by a linearization of the free-surface equation which can be written:

$$\partial_t \eta + \partial_x H\widehat{u} + \partial_y H\widehat{v} = P - E + R \tag{2.10}$$

which differs from the depth integrated continuity equation with rigid-lid ((2.1)) by the time-dependent term and fresh-water source term.

Equation (2.4) in the rigid-lid pressure method is then replaced by the time discretization of (2.10) which is:

$$\eta^{n+1} + \Delta t \partial_x H\widehat{u^{n+1}} + \Delta t \partial_y H\widehat{v^{n+1}} = \eta^n + \Delta t(P - E) \tag{2.11}$$

where the use of flow at time level $n+1$ makes the method implicit and backward in time. This is the preferred scheme since it still filters the fast unresolved wave motions by damping them. A centered scheme, such as Crank-Nicholson (see Section 2.10.1), would alias the energy of the fast modes onto slower modes of motion.

As for the rigid-lid pressure method, equations (2.2), (2.3) and (2.11) can be re-arranged as follows:

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \tag{2.12}$$

$$v^* = v^n + \Delta t G_v^{(n+1/2)} \tag{2.13}$$

$$\eta^* = \epsilon_{fs}(\eta^n + \Delta t(P - E)) - \Delta t(\partial_x H\widehat{u^*} + \partial_y H\widehat{v^*}) \tag{2.14}$$

$$\partial_x g H \partial_x \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} - \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \tag{2.15}$$

$$u^{n+1} = u^* - \Delta t g \partial_x \eta^{n+1} \tag{2.16}$$

$$v^{n+1} = v^* - \Delta t g \partial_y \eta^{n+1} \tag{2.17}$$

Equations (2.12) to (2.17), solved sequentially, represent the pressure method algorithm with a backward implicit, linearized free surface. The method is still formerly a pressure method because in the limit of large Δt the rigid-lid method is recovered. However, the implicit treatment of the free-surface allows the flow to be divergent and for the surface pressure/elevation to respond on a finite time-scale (as opposed to instantly). To recover the rigid-lid formulation, we use a switch-like variable, ϵ_{fs} (freesurfFac), which selects between the free-surface and rigid-lid; $\epsilon_{fs} = 1$ allows the free-surface to evolve; $\epsilon_{fs} = 0$ imposes the rigid-lid. The evolution in time and location of variables is exactly as it was for the rigid-lid model so that Figure 2.1 is still applicable. Similarly, the calling sequence, given *here*, is as for the pressure-method.

2.5 Explicit time-stepping: Adams-Bashforth

In describing the the pressure method above we deferred describing the time discretization of the explicit terms. We have historically used the quasi-second order Adams-Bashforth method for all explicit terms in both the momentum and tracer equations. This is still the default mode of operation but it is now possible to use alternate schemes for tracers (see Section 2.16). In the previous sections, we summarized an explicit scheme as:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.18}$$

where τ could be any prognostic variable (u, v, θ or S) and τ^* is an explicit estimate of τ^{n+1} and would be exact if not for implicit-in-time terms. The parenthesis about $n+1/2$ indicates that the term is explicit and extrapolated forward in time and for this we use the quasi-second order Adams-Bashforth method:

$$G_\tau^{(n+1/2)} = (3/2 + \epsilon_{AB})G_\tau^n - (1/2 + \epsilon_{AB})G_\tau^{n-1} \tag{2.19}$$

This is a linear extrapolation, forward in time, to $t = (n + 1/2 + \epsilon_{AB})\Delta t$. An extrapolation to the mid-point in time, $t = (n + 1/2)\Delta t$, corresponding to $\epsilon_{AB} = 0$, would be second order accurate but is weakly unstable for oscillatory terms. A small but finite value for ϵ_{AB} stabilizes the method. Strictly speaking, damping terms such as diffusion and dissipation, and fixed terms (forcing), do not need to be inside the Adams-Bashforth extrapolation. However, in the current code, it is simpler to include these terms and this can be justified if the flow and forcing evolves smoothly. Problems can, and do, arise when forcing or motions are high frequency and this corresponds to a reduced stability compared to a simple forward time-stepping of such terms. The model offers the possibility to leave terms outside the Adams-Bashforth extrapolation, by turning off the logical flag forcing_In_AB (parameter file `data`, namelist `PARM01`, default value = TRUE) and then setting tracForcingOutAB (default=0), momForcingOutAB (default=0), and momDissip_In_AB (parameter file `data`, namelist `PARM01`, default value = TRUE), respectively for the tracer terms, momentum forcing terms, and the dissipation terms.

A stability analysis for an oscillation equation should be given at this point.

A stability analysis for a relaxation equation should be given at this point.

2.6 Implicit time-stepping: backward method

Vertical diffusion and viscosity can be treated implicitly in time using the backward method which is an intrinsic scheme. Recently, the option to treat the vertical advection implicitly has been added, but not yet tested; therefore, the description hereafter is limited to diffusion and viscosity. For tracers, the time discretized equation is:

$$\tau^{n+1} - \Delta t \partial_r \kappa_v \partial_r \tau^{n+1} = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.20}$$

where $G_\tau^{(n+1/2)}$ is the remaining explicit terms extrapolated using the Adams-Bashforth method as described above. Equation (2.20) can be split split into:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.21}$$

$$\tau^{n+1} = \mathcal{L}_\tau^{-1}(\tau^*) \tag{2.22}$$

where $\mathcal{L}_\tau^{-1}$ is the inverse of the operator

$$\mathcal{L}_\tau = [1 + \Delta t \partial_r \kappa_v \partial_r]$$

Equation (2.21) looks exactly as (2.18) while (2.22) involves an operator or matrix inversion. By re-arranging (2.20) in this way we have cast the method as an explicit prediction step and an implicit step allowing the latter to be inserted into the over all algorithm with minimal interference.

The calling sequence for stepping forward a tracer variable such as temperature with implicit diffusion is as follows:

Adams-Bashforth calling tree

FORWARD_STEP	
THERMODYNAMICS	
TEMP_INTEGRATE	
GAD_CALC_RHS	$G_\theta^n = G_\theta(u, \theta^n)$

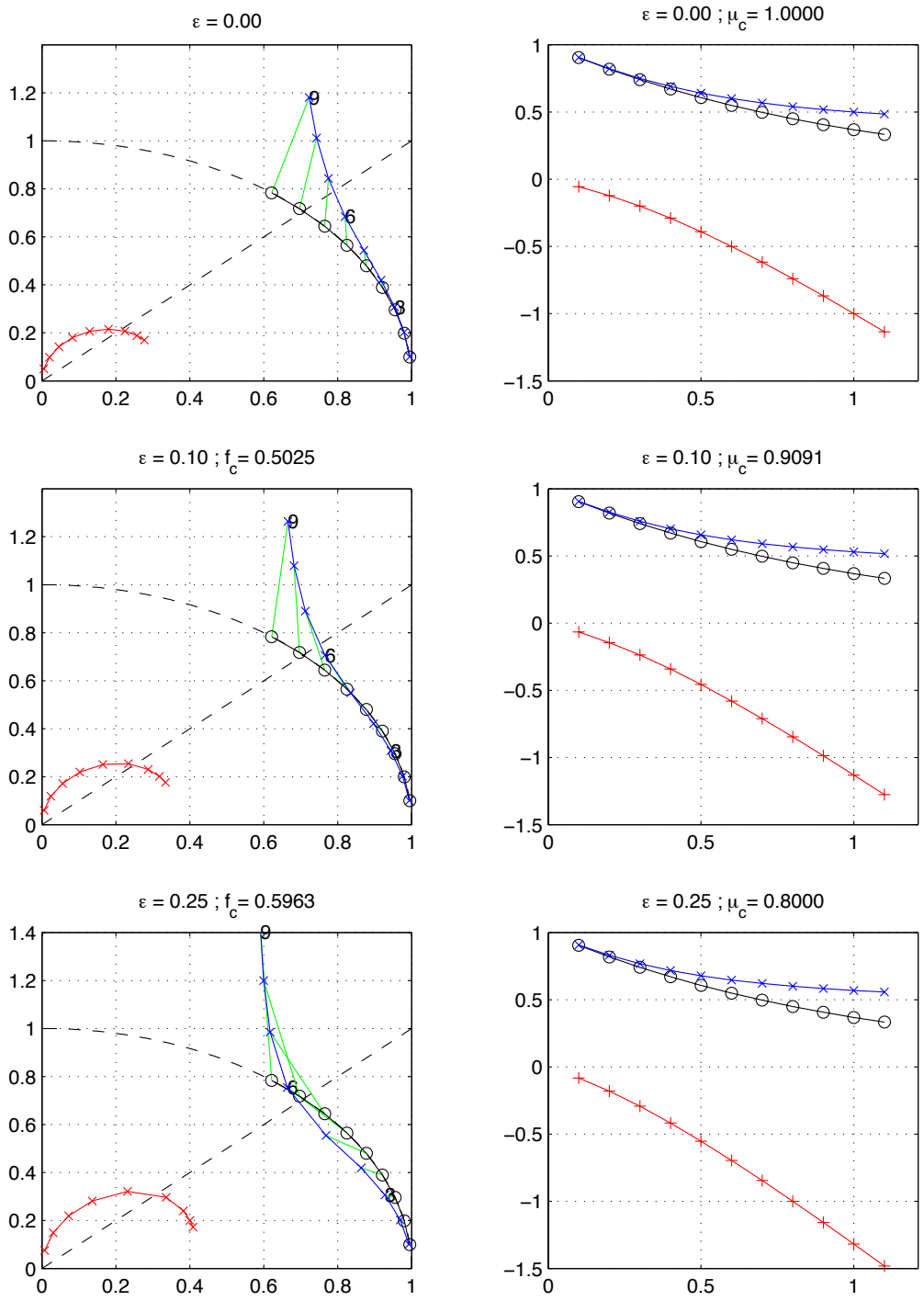

Figure 2.2: Oscillatory and damping response of quasi-second order Adams-Bashforth scheme for different values of the ϵ_{AB} parameter (0.0, 0.1, 0.25, from top to bottom) The analytical solution (in black), the physical mode (in blue) and the numerical mode (in red) are represented with a CFL step of 0.1. The left column represents the oscillatory response on the complex plane for CFL ranging from 0.1 up to 0.9. The right column represents the damping response amplitude (y-axis) function of the CFL (x-axis).

either	
EXTERNAL_FORCING	$G_\theta^n = G_\theta^n + \mathcal{Q}$
ADAMS_BASHFORTH2	$G_\theta^{(n+1/2)}$ (2.19)
or	
EXTERNAL_FORCING	$G_\theta^{(n+1/2)} = G_\theta^{(n+1/2)} + \mathcal{Q}$
TIMESTEP_TRACER	τ^* (2.18)
IMPLDIFF	$\tau^{(n+1)}$ (2.22)

In order to fit within the pressure method, the implicit viscosity must not alter the barotropic flow. In other words, it can only redistribute momentum in the vertical. The upshot of this is that although vertical viscosity may be backward implicit and unconditionally stable, no-slip boundary conditions may not be made implicit and are thus cast as a an explicit drag term.

2.7 Synchronous time-stepping: variables co-located in time

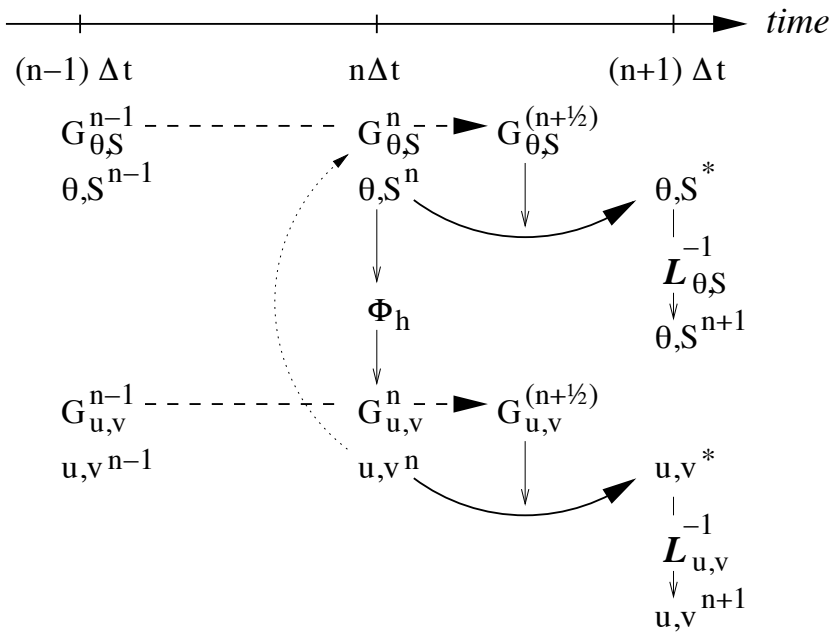

Figure 2.3: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm. All prognostic variables are co-located in time. Explicit tendencies are evaluated at time level n as a function of the state at that time level (dotted arrow). The explicit tendency from the previous time level, $n-1$, is used to extrapolate tendencies to $n+1/2$ (dashed arrow). This extrapolated tendency allows variables to be stably integrated forward-in-time to render an estimate ($*$ -variables) at the $n+1$ time level (solid arc-arrow). The operator $\mathcal{L}$ formed from implicit-in-time terms is solved to yield the state variables at time level $n+1$.

The Adams-Bashforth extrapolation of explicit tendencies fits neatly into the pressure method algorithm when all state variables are co-located in time. The algorithm can be represented by the sequential solution of the follow equations:

$$G_{\theta,S}^n = G_{\theta,S}(u^n, \theta^n, S^n) \tag{2.23}$$

$$G_{\theta,S}^{(n+1/2)} = (3/2 + \epsilon_{AB})G_{\theta,S}^n - (1/2 + \epsilon_{AB})G_{\theta,S}^{n-1} \tag{2.24}$$

$$(\theta^*, S^*) = (\theta^n, S^n) + \Delta t G_{\theta,S}^{(n+1/2)} \tag{2.25}$$

$$(\theta^{n+1}, S^{n+1}) = \mathcal{L}_{\theta,S}^{-1}(\theta^*, S^*) \tag{2.26}$$

$$\phi_{hyd}^n = \int b(\theta^n, S^n) dr \tag{2.27}$$

$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^n = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^n, \phi_{hyd}^n) \tag{2.28}$$

$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n+1/2)} = (3/2 + \epsilon_{AB})\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^n - (1/2 + \epsilon_{AB})\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1} \tag{2.29}$$

$$\vec{\mathbf{v}}^* = \vec{\mathbf{v}}^n + \Delta t \vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n+1/2)} \tag{2.30}$$

$$\vec{\mathbf{v}}^{**} = \mathcal{L}_{\vec{\mathbf{v}}}^{-1}(\vec{\mathbf{v}}^*) \tag{2.31}$$

$$\eta^* = \epsilon_{fs}\left(\eta^n + \Delta t(P - E)\right) - \Delta t \nabla \cdot H\widehat{\vec{\mathbf{v}}^{**}} \tag{2.32}$$

$$\nabla \cdot gH\nabla\eta^{n+1} - \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \tag{2.33}$$

$$\vec{\mathbf{v}}^{n+1} = \vec{\mathbf{v}}^{**} - \Delta t g \nabla \eta^{n+1} \tag{2.34}$$

Figure 2.3 illustrates the location of variables in time and evolution of the algorithm with time. The Adams-Bashforth extrapolation of the tracer tendencies is illustrated by the dashed arrow, the prediction at $n+1$ is indicated by the solid arc. Inversion of the implicit terms, $\mathcal{L}_{\theta,S}^{-1}$, then yields the new tracer fields at $n+1$. All these operations are carried out in subroutine THERMODYNAMICS and subsidiaries, which correspond to equations (2.23) to (2.26). Similarly illustrated is the Adams-Bashforth extrapolation of accelerations, stepping forward and solving of implicit viscosity and surface pressure gradient terms, corresponding to equations (2.28) to (2.34). These operations are carried out in subroutines DYNAMICS, SOLVE_FOR_PRESSURE and MOMENTUM_CORRECTION_STEP. This, then, represents an entire algorithm for stepping forward the model one time-step. The corresponding calling tree for the overall synchronous algorithm using Adams-Bashforth time-stepping is given below. The place where the model geometry hFac factors) is updated is added here but is only relevant for the non-linear free-surface algorithm. For completeness, the external forcing, ocean and atmospheric physics have been added, although they are mainly optional.

Synchronous Adams-Bashforth calling tree

- FORWARD_STEP
  - EXTERNAL_FIELDS_LOAD
  - DO_ATMOSPHERIC_PHYS
  - DO_OCEANIC_PHYS
  - THERMODYNAMICS
    - CALC_GT
      - GAD_CALC_RHS — $G_\theta^n = G_\theta(u, \theta^n)$ (2.23)
      - EXTERNAL_FORCING — $G_\theta^n = G_\theta^n + \mathcal{Q}$
      - ADAMS_BASHFORTH2 — $G_\theta^{(n+1/2)}$ (2.24)
    - TIMESTEP_TRACER — θ^* (2.25)
    - IMPLDIFF — $\theta^{(n+1)}$ (2.26)
  - DYNAMICS
    - CALC_PHI_HYD — ϕ_{hyd}^n (2.27)
    - MOM_FLUXFORM or MOM_VECINV — $G_{\vec{\mathbf{v}}}^n$ (2.28)
    - TIMESTEP — $\vec{\mathbf{v}}^*$ (2.29), (2.30)
    - IMPLDIFF — $\vec{\mathbf{v}}^{**}$ (2.31)
  - UPDATE_R_STAR or UPDATE_SURF_DR (NonLin-FS only)
  - SOLVE_FOR_PRESSURE
    - CALC_DIV_GHAT — η^* (2.32)

CG2D	η^{n+1} (2.33)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla\eta^{n+1}$
CORRECTION_STEP	u^{n+1}, v^{n+1} (2.34)
TRACERS_CORRECTION_STEP	
CYCLE_TRACER	θ^{n+1}
SHAP_FILT_APPLY_TS or ZONAL_FILT_APPLY_TS	
CONVECTIVE_ADJUSTMENT	

2.8 Staggered baroclinic time-stepping

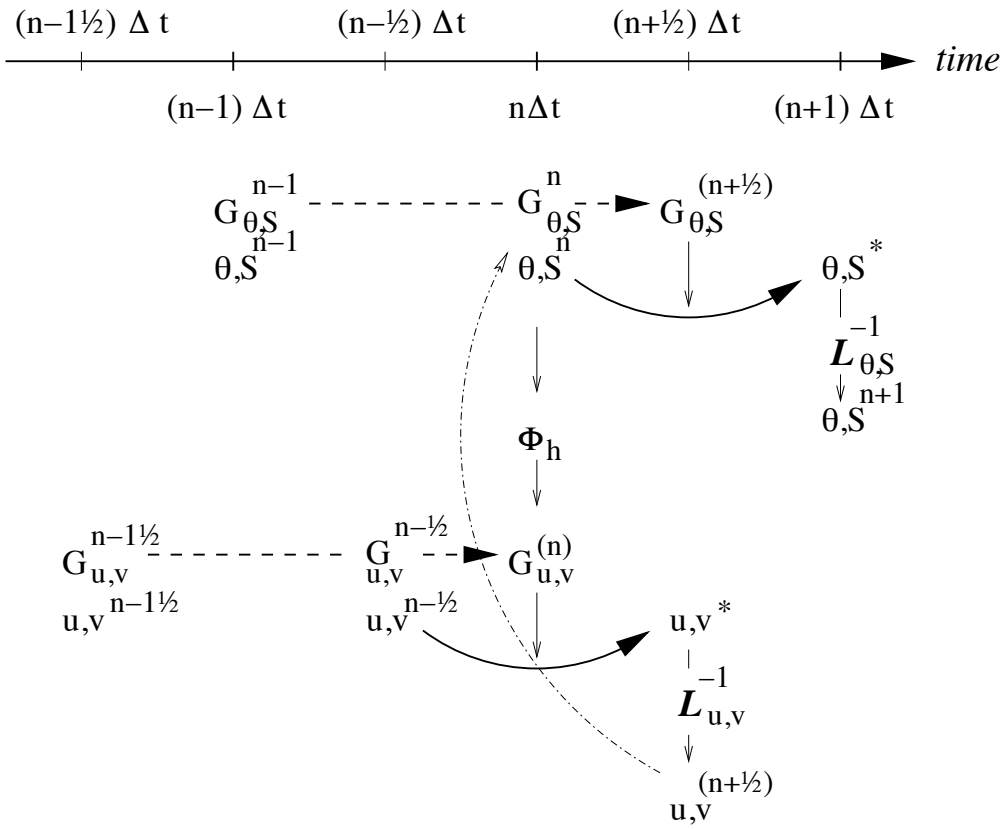

Figure 2.4: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm but with staggering in time of thermodynamic variables with the flow. Explicit momentum tendencies are evaluated at time level $n - 1/2$ as a function of the flow field at that time level $n - 1/2$. The explicit tendency from the previous time level, $n - 3/2$, is used to extrapolate tendencies to n (dashed arrow). The hydrostatic pressure/geo-potential ϕ_{hyd} is evaluated directly at time level n (vertical arrows) and used with the extrapolated tendencies to step forward the flow variables from $n - 1/2$ to $n + 1/2$ (solid arc-arrow). The implicit-in-time operator $\mathcal{L}_{\mathbf{u,v}}$ (vertical arrows) is then applied to the previous estimation of the the flow field ($*$ -variables) and yields to the two velocity components u, v at time level $n + 1/2$. These are then used to calculate the advection term (dashed arc-arrow) of the thermo-dynamics tendencies at time step n. The extrapolated thermodynamics tendency, from time level $n - 1$ and n to $n + 1/2$, allows thermodynamic variables to be stably integrated forward-in-time (solid arc-arrow) up to time level $n + 1$.

For well-stratified problems, internal gravity waves may be the limiting process for determining a stable time-step. In the circumstance, it is more efficient to stagger in time the thermodynamic variables with the flow variables. Figure 2.4 illustrates the staggering and algorithm. The key difference between this and Figure 2.3 is that the thermodynamic variables are solved after the dynamics, using the recently updated flow field. This essentially allows the gravity wave terms to leap-frog in time giving second order accuracy and more stability.

The essential change in the staggered algorithm is that the thermodynamics solver is delayed from half a time step, allowing the use of the most recent velocities to compute the advection terms. Once the thermodynamics fields are updated, the hydrostatic pressure is computed to step forward the dynamics. Note that the pressure gradient must also be taken out of the Adams-Bashforth extrapolation. Also, retaining the integer time-levels, n and $n+1$, does not give a user the sense of where variables are located in time. Instead, we re-write the entire algorithm, (2.23) to (2.34), annotating the position in time of variables appropriately:

$$\phi^n_{hyd} = \int b(\theta^n, S^n) dr \tag{2.35}$$

$$\vec{\mathbf{G}}^{n-1/2}_{\vec{\mathbf{v}}} = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^{n-1/2}) \tag{2.36}$$

$$\vec{\mathbf{G}}^{(n)}_{\vec{\mathbf{v}}} = (3/2 + \epsilon_{AB})\vec{\mathbf{G}}^{n-1/2}_{\vec{\mathbf{v}}} - (1/2 + \epsilon_{AB})\vec{\mathbf{G}}^{n-3/2}_{\vec{\mathbf{v}}} \tag{2.37}$$

$$\vec{\mathbf{v}}^* = \vec{\mathbf{v}}^{n-1/2} + \Delta t \left(\vec{\mathbf{G}}^{(n)}_{\vec{\mathbf{v}}} - \nabla \phi^n_{hyd} \right) \tag{2.38}$$

$$\vec{\mathbf{v}}^{**} = \mathcal{L}^{-1}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^*) \tag{2.39}$$

$$\eta^* = \epsilon_{fs} \left(\eta^{n-1/2} + \Delta t (P-E)^n \right) - \Delta t \nabla \cdot H\widehat{\vec{\mathbf{v}}^{**}} \tag{2.40}$$

$$\nabla \cdot gH\nabla \eta^{n+1/2} - \frac{\epsilon_{fs}\eta^{n+1/2}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \tag{2.41}$$

$$\vec{\mathbf{v}}^{n+1/2} = \vec{\mathbf{v}}^{**} - \Delta t g \nabla \eta^{n+1/2} \tag{2.42}$$

$$G^n_{\theta,S} = G_{\theta,S}(u^{n+1/2}, \theta^n, S^n) \tag{2.43}$$

$$G^{(n+1/2)}_{\theta,S} = (3/2 + \epsilon_{AB})G^n_{\theta,S} - (1/2 + \epsilon_{AB})G^{n-1}_{\theta,S} \tag{2.44}$$

$$(\theta^*, S^*) = (\theta^n, S^n) + \Delta t G^{(n+1/2)}_{\theta,S} \tag{2.45}$$

$$(\theta^{n+1}, S^{n+1}) = \mathcal{L}^{-1}_{\theta,S}(\theta^*, S^*) \tag{2.46}$$

The corresponding calling tree is given below. The staggered algorithm is activated with the run-time flag staggerTimeStep =.TRUE. in parameter file `data`, namelist `PARM01`.

Staggered Adams-Bashforth calling tree

FORWARD_STEP
- EXTERNAL_FIELDS_LOAD
- DO_ATMOSPHERIC_PHYS
- DO_OCEANIC_PHYS

DYNAMICS
- CALC_PHI_HYD — ϕ^n_{hyd} (2.35)
- MOM_FLUXFORM or MOM_VECINV — $G^{n-1/2}_{\vec{\mathbf{v}}}$ (2.36)
- TIMESTEP — $\vec{\mathbf{v}}^*$ (2.37), (2.38)
- IMPLDIFF — $\vec{\mathbf{v}}^{**}$ (2.39)

UPDATE_R_STAR or UPDATE_SURF_DR (NonLin-FS only)

SOLVE_FOR_PRESSURE
- CALC_DIV_GHAT — η^* (2.40)

CG2D	$\eta^{n+1/2}$ (2.41)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla\eta^{n+1/2}$
CORRECTION_STEP	$u^{n+1/2}, v^{n+1/2}$ (2.42)
THERMODYNAMICS	
CALC_GT	
GAD_CALC_RHS	$G_\theta^n = G_\theta(u, \theta^n)$ (2.43)
EXTERNAL_FORCING	$G_\theta^n = G_\theta^n + \mathcal{Q}$
ADAMS_BASHFORTH2	$G_\theta^{(n+1/2)}$ (2.44)
TIMESTEP_TRACER	θ^* (2.45)
IMPLDIFF	$\theta^{(n+1)}$ (2.46)
TRACERS_CORRECTION_STEP	
CYCLE_TRACER	θ^{n+1}
SHAP_FILT_APPLY_TS or ZONAL_FILT_APPLY_TS	
CONVECTIVE_ADJUSTMENT	

The only difficulty with this approach is apparent in equation (2.43) and illustrated by the dotted arrow connecting $u, v^{n+1/2}$ with G_θ^n. The flow used to advect tracers around is not naturally located in time. This could be avoided by applying the Adams-Bashforth extrapolation to the tracer field itself and advecting that around but this approach is not yet available. We're not aware of any detrimental effect of this feature. The difficulty lies mainly in interpretation of what time-level variables and terms correspond to.

2.9 Non-hydrostatic formulation

The non-hydrostatic formulation re-introduces the full vertical momentum equation and requires the solution of a 3-D elliptic equations for non-hydrostatic pressure perturbation. We still integrate vertically for the hydrostatic pressure and solve a 2-D elliptic equation for the surface pressure/elevation for this reduces the amount of work needed to solve for the non-hydrostatic pressure.

The momentum equations are discretized in time as follows:

$$\frac{1}{\Delta t}u^{n+1} + g\partial_x\eta^{n+1} + \partial_x\phi_{nh}^{n+1} = \frac{1}{\Delta t}u^n + G_u^{(n+1/2)} \tag{2.47}$$

$$\frac{1}{\Delta t}v^{n+1} + g\partial_y\eta^{n+1} + \partial_y\phi_{nh}^{n+1} = \frac{1}{\Delta t}v^n + G_v^{(n+1/2)} \tag{2.48}$$

$$\frac{1}{\Delta t}w^{n+1} + \partial_r\phi_{nh}^{n+1} = \frac{1}{\Delta t}w^n + G_w^{(n+1/2)} \tag{2.49}$$

which must satisfy the discrete-in-time depth integrated continuity, equation (2.11) and the local continuity equation

$$\partial_x u^{n+1} + \partial_y v^{n+1} + \partial_r w^{n+1} = 0 \tag{2.50}$$

As before, the explicit predictions for momentum are consolidated as:

$$\begin{aligned} u^* &= u^n + \Delta t G_u^{(n+1/2)} \\ v^* &= v^n + \Delta t G_v^{(n+1/2)} \\ w^* &= w^n + \Delta t G_w^{(n+1/2)} \end{aligned}$$

but this time we introduce an intermediate step by splitting the tendency of the flow as follows:

$$\begin{aligned} u^{n+1} &= u^{**} - \Delta t\partial_x\phi_{nh}^{n+1} \quad u^{**} = u^* - \Delta t g\partial_x\eta^{n+1} \\ v^{n+1} &= v^{**} - \Delta t\partial_y\phi_{nh}^{n+1} \quad v^{**} = v^* - \Delta t g\partial_y\eta^{n+1} \end{aligned}$$

Substituting into the depth integrated continuity (equation (2.11)) gives

$$\partial_x H \partial_x \left(g\eta^{n+1} + \widehat{\phi}_{nh}^{n+1} \right) + \partial_y H \partial_y \left(g\eta^{n+1} + \widehat{\phi}_{nh}^{n+1} \right) - \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \tag{2.51}$$

which is approximated by equation (2.15) on the basis that i) ϕ_{nh}^{n+1} is not yet known and ii) $\nabla\widehat{\phi}_{nh} << g\nabla\eta$. If (2.15) is solved accurately then the implication is that $\widehat{\phi}_{nh} \approx 0$ so that the non-hydrostatic pressure field does not drive barotropic motion.

The flow must satisfy non-divergence (equation (2.50)) locally, as well as depth integrated, and this constraint is used to form a 3-D elliptic equations for ϕ_{nh}^{n+1}:

$$\partial_{xx}\phi_{nh}^{n+1} + \partial_{yy}\phi_{nh}^{n+1} + \partial_{rr}\phi_{nh}^{n+1} = \partial_x u^{**} + \partial_y v^{**} + \partial_r w^* \tag{2.52}$$

The entire algorithm can be summarized as the sequential solution of the following equations:

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \tag{2.53}$$

$$v^* = v^n + \Delta t G_v^{(n+1/2)} \tag{2.54}$$

$$w^* = w^n + \Delta t G_w^{(n+1/2)} \tag{2.55}$$

$$\eta^* \; = \; \epsilon_{fs}\left(\eta^n + \Delta t(P-E)\right) - \Delta t\left(\partial_x H\widehat{u^*} + \partial_y H\widehat{v^*}\right) \tag{2.56}$$

$$\partial_x gH\partial_x\eta^{n+1} + \partial_y gH\partial_y\eta^{n+1} - \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} \; = \; -\frac{\eta^*}{\Delta t^2} \tag{2.57}$$

$$u^{**} = u^* - \Delta t g \partial_x \eta^{n+1} \tag{2.58}$$

$$v^{**} = v^* - \Delta t g \partial_y \eta^{n+1} \tag{2.59}$$

$$\partial_{xx}\phi_{nh}^{n+1} + \partial_{yy}\phi_{nh}^{n+1} + \partial_{rr}\phi_{nh}^{n+1} = \partial_x u^{**} + \partial_y v^{**} + \partial_r w^* \tag{2.60}$$

$$u^{n+1} = u^{**} - \Delta t \partial_x \phi_{nh}^{n+1} \tag{2.61}$$

$$v^{n+1} = v^{**} - \Delta t \partial_y \phi_{nh}^{n+1} \tag{2.62}$$

$$\partial_r w^{n+1} = -\partial_x u^{n+1} - \partial_y v^{n+1} \tag{2.63}$$

where the last equation is solved by vertically integrating for w^{n+1}.

2.10 Variants on the Free Surface

We now describe the various formulations of the free-surface that include non-linear forms, implicit in time using Crank-Nicholson, explicit and [one day] split-explicit. First, we'll reiterate the underlying algorithm but this time using the notation consistent with the more general vertical coordinate r. The elliptic equation for free-surface coordinate (units of r), corresponding to (2.11), and assuming no non-hydrostatic effects ($\epsilon_{nh} = 0$) is:

$$\epsilon_{fs}\eta^{n+1} - \nabla_h \cdot \Delta t^2 (R_o - R_{fixed}) \nabla_h b_s \eta^{n+1} = \eta^* \tag{2.64}$$

where

$$\eta^* = \epsilon_{fs}\,\eta^n - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o} \vec{\mathbf{v}}^* dr \; + \; \epsilon_{fw}\Delta t (P-E)^n \tag{2.65}$$

S/R SOLVE_FOR_PRESSURE

u^* : gU (DYNVARS.h)
v^* : gV (DYNVARS.h)
η^* : cg2d_b (SOLVE_FOR_PRESSURE.h)
η^{n+1} : etaN (DYNVARS.h)

Once η^{n+1} has been found, substituting into (2.2), (2.3) yields $\vec{\mathbf{v}}^{n+1}$ if the model is hydrostatic ($\epsilon_{nh} = 0$):

$$\vec{\mathbf{v}}^{n+1} = \vec{\mathbf{v}}^* - \Delta t \nabla_h b_s \eta^{n+1}$$

This is known as the correction step. However, when the model is non-hydrostatic ($\epsilon_{nh} = 1$) we need an additional step and an additional equation for ϕ'_{nh}. This is obtained by substituting (2.47), (2.48) and (2.49) into continuity:

$$[\nabla_h^2 + \partial_{rr}]\phi'_{nh}{}^{n+1} = \frac{1}{\Delta t}\nabla_h \cdot \vec{\mathbf{v}}^{**} + \partial_r \dot{r}^* \tag{2.66}$$

where

$$\vec{\mathbf{v}}^{**} = \vec{\mathbf{v}}^* - \Delta t \nabla_h b_s \eta^{n+1}$$

Note that η^{n+1} is also used to update the second RHS term $\partial_r \dot{r}^*$ since the vertical velocity at the surface ($\dot{r}_{surf}$) is evaluated as $(\eta^{n+1} - \eta^n)/\Delta t$.

Finally, the horizontal velocities at the new time level are found by:

$$\vec{\mathbf{v}}^{n+1} = \vec{\mathbf{v}}^{**} - \epsilon_{nh} \Delta t \nabla_h \phi'_{nh}{}^{n+1} \tag{2.67}$$

and the vertical velocity is found by integrating the continuity equation vertically. Note that, for the convenience of the restart procedure, the vertical integration of the continuity equation has been moved to the beginning of the time step (instead of at the end), without any consequence on the solution.

S/R CORRECTION_STEP

η^{n+1} : etaN (DYNVARS.h)
ϕ_{nh}^{n+1} : phi_nh (NH_VARS.h)
u^* : gU (DYNVARS.h)
v^* : gV (DYNVARS.h)
u^{n+1} : uVel (DYNVARS.h)
v^{n+1} : vVel (DYNVARS.h)

Regarding the implementation of the surface pressure solver, all computation are done within the routine SOLVE_FOR_PRESSURE and its dependent calls. The standard method to solve the 2D elliptic problem (2.64) uses the conjugate gradient method (routine CG2D); the solver matrix and conjugate gradient operator are only function of the discretized domain and are therefore evaluated separately, before the time iteration loop, within INI_CG2D. The computation of the RHS η^* is partly done in CALC_DIV_GHAT and in SOLVE_FOR_PRESSURE.

The same method is applied for the non hydrostatic part, using a conjugate gradient 3D solver (CG3D) that is initialized in INI_CG3D. The RHS terms of 2D and 3D problems are computed together at the same point in the code.

2.10.1 Crank-Nicolson barotropic time stepping

The full implicit time stepping described previously is unconditionally stable but damps the fast gravity waves, resulting in a loss of potential energy. The modification presented now allows one to combine an implicit part (γ, β) and an explicit part $(1-\gamma, 1-\beta)$ for the surface pressure gradient (γ) and for the barotropic flow divergence (β). For

instance, $\gamma = \beta = 1$ is the previous fully implicit scheme; $\gamma = \beta = 1/2$ is the non damping (energy conserving), unconditionally stable, Crank-Nicolson scheme; $(\gamma, \beta) = (1, 0)$ or $= (0, 1)$ corresponds to the forward - backward scheme that conserves energy but is only stable for small time steps. In the code, γ, β are defined as parameters, respectively implicSurfPress, implicDiv2DFlow. They are read from the main parameter file `data` (namelist `PARM01`) and are set by default to 1,1.

Equations (2.12) – (2.17) are modified as follows:

$$\frac{\vec{\mathbf{v}}^{n+1}}{\Delta t} + \nabla_h b_s[\gamma\eta^{n+1} + (1-\gamma)\eta^n] + \epsilon_{nh}\nabla_h {\phi'_{nh}}^{n+1} = \frac{\vec{\mathbf{v}}^n}{\Delta t} + \vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n+1/2)} + \nabla_h {\phi'_{hyd}}^{(n+1/2)}$$

$$\epsilon_{fs}\frac{\eta^{n+1} - \eta^n}{\Delta t} + \nabla_h \cdot \int_{R_{fixed}}^{R_o} [\beta\vec{\mathbf{v}}^{n+1} + (1-\beta)\vec{\mathbf{v}}^n] dr = \epsilon_{fw}(P - E) \tag{2.68}$$

We set

$$\begin{aligned}
\vec{\mathbf{v}}^* = &\quad \vec{\mathbf{v}}^n + \Delta t\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n+1/2)} + (\gamma - 1)\Delta t\nabla_h b_s\eta^n + \Delta t\nabla_h {\phi'_{hyd}}^{(n+1/2)} \\
\eta^* = &\quad \epsilon_{fs}\eta^n + \epsilon_{fw}\Delta t(P - E) - \Delta t\nabla_h \cdot \int_{R_{fixed}}^{R_o} [\beta\vec{\mathbf{v}}^* + (1-\beta)\vec{\mathbf{v}}^n] dr
\end{aligned}$$

In the hydrostatic case $\epsilon_{nh} = 0$, allowing us to find η^{n+1}, thus:

$$\epsilon_{fs}\eta^{n+1} - \nabla_h \cdot \gamma\beta\Delta t^2 b_s(R_o - R_{fixed})\nabla_h\eta^{n+1} = \eta^*$$

and then to compute (CORRECTION_STEP):

$$\vec{\mathbf{v}}^{n+1} = \vec{\mathbf{v}}^* - \gamma\Delta t\nabla_h b_s\eta^{n+1}$$

Notes:

1. The RHS term of equation (2.68) corresponds the contribution of fresh water flux (P-E) to the free-surface variations ($\epsilon_{fw} = 1$, useRealFreshWaterFlux =.TRUE. in parameter file `data`). In order to remain consistent with the tracer equation, specially in the non-linear free-surface formulation, this term is also affected by the Crank-Nicolson time stepping. The RHS reads: $\epsilon_{fw}(\beta(P-E)^{n+1/2} + (1-\beta)(P-E)^{n-1/2})$
2. The stability criteria with Crank-Nicolson time stepping for the pure linear gravity wave problem in cartesian coordinates is:
   - $\gamma + \beta < 1$: unstable
   - $\gamma \geq 1/2$ and $\beta \geq 1/2$: stable
   - $\gamma + \beta \geq 1$: stable if $c_{max}^2(\gamma - 1/2)(\beta - 1/2) + 1 \geq 0$ with $c_{max} = 2\Delta t\sqrt{gH}\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}}$
3. A similar mixed forward/backward time-stepping is also available for the non-hydrostatic algorithm, with a fraction γ_{nh} ($0 < \gamma_{nh} \leq 1$) of the non-hydrostatic pressure gradient being evaluated at time step $n+1$ (backward in time) and the remaining part $(1 - \gamma_{nh})$ being evaluated at time step n (forward in time). The run-time parameter implicitNHPress corresponding to the implicit fraction γ_{nh} of the non-hydrostatic pressure is set by default to the implicit fraction γ of surface pressure (implicSurfPress), but can also be specified independently (in main parameter file `data`, namelist `PARM01`).

2.10.2 Non-linear free-surface

Options have been added to the model that concern the free surface formulation.

2.10.2.1 Pressure/geo-potential and free surface

For the atmosphere, since $\phi = \phi_{topo} - \int_{p_s}^{p} \alpha dp$, subtracting the reference state defined in section Section 1.4.1.2 :

$$\phi_o = \phi_{topo} - \int_{p_o}^{p} \alpha_o dp \quad \text{with} \quad \phi_o(p_o) = \phi_{topo}$$

we get:

$$\phi' = \phi - \phi_o = \int_{p}^{p_s} \alpha dp - \int_{p}^{p_o} \alpha_o dp$$

For the ocean, the reference state is simpler since ρ_c does not dependent on z ($b_o = g$) and the surface reference position is uniformly $z = 0$ ($R_o = 0$), and the same subtraction leads to a similar relation. For both fluids, using the isomorphic notations, we can write:

$$\phi' = \int_{r}^{r_{surf}} b\, dr - \int_{r}^{R_o} b_o dr$$

and re-write as:

$$\phi' = \int_{R_o}^{r_{surf}} b\, dr + \int_{r}^{R_o} (b - b_o) dr \tag{2.69}$$

or:

$$\phi' = \int_{R_o}^{r_{surf}} b_o dr + \int_{r}^{r_{surf}} (b - b_o) dr \tag{2.70}$$

In section Section 1.3.6, following eq. (2.69), the pressure/geo-potential ϕ' has been separated into surface (ϕ_s), and hydrostatic anomaly (ϕ'_{hyd}). In this section, the split between ϕ_s and ϕ'_{hyd} is made according to equation (2.70). This slightly different definition reflects the actual implementation in the code and is valid for both linear and non-linear free-surface formulation, in both r-coordinate and r*-coordinate.

Because the linear free-surface approximation ignores the tracer content of the fluid parcel between R_o and $r_{surf} = R_o + \eta$, for consistency reasons, this part is also neglected in ϕ'_{hyd} :

$$\phi'_{hyd} = \int_{r}^{r_{surf}} (b - b_o) dr \simeq \int_{r}^{R_o} (b - b_o) dr$$

Note that in this case, the two definitions of ϕ_s and ϕ'_{hyd} from equations (2.69) and (2.70) converge toward the same (approximated) expressions: $\phi_s = \int_{R_o}^{r_{surf}} b_o dr$ and $\phi'_{hyd} = \int_{r}^{R_o} b' dr$. On the contrary, the unapproximated formulation (see Section 2.10.2.2) retains the full expression: $\phi'_{hyd} = \int_{r}^{r_{surf}} (b - b_o) dr$. This is obtained by selecting nonlinFreeSurf =4 in parameter file `data`. Regarding the surface potential:

$$\phi_s = \int_{R_o}^{R_o+\eta} b_o dr = b_s \eta \quad \text{with} \quad b_s = \frac{1}{\eta} \int_{R_o}^{R_o+\eta} b_o dr$$

$b_s \simeq b_o(R_o)$ is an excellent approximation (better than the usual numerical truncation, since generally $|\eta|$ is smaller than the vertical grid increment).

For the ocean, $\phi_s = g\eta$ and $b_s = g$ is uniform. For the atmosphere, however, because of topographic effects, the reference surface pressure $R_o = p_o$ has large spatial variations that are responsible for significant b_s variations (from 0.8 to 1.2 $[m^3/kg]$). For this reason, when uniformLin_PhiSurf =.FALSE. (parameter file `data`, namelist `PARAM01`) a non-uniform linear coefficient b_s is used and computed (INI_LINEAR_PHISURF) according to the reference surface pressure p_o: $b_s = b_o(R_o) = c_p \kappa (p_o/P^o_{SL})^{(\kappa-1)} \theta_{ref}(p_o)$, with P^o_{SL} the mean sea-level pressure.

2.10.2.2 Free surface effect on column total thickness (Non-linear free-surface)

The total thickness of the fluid column is $r_{surf} - R_{fixed} = \eta + R_o - R_{fixed}$. In most applications, the free surface displacements are small compared to the total thickness $\eta \ll H_o = R_o - R_{fixed}$. In the previous sections and in older version of the model, the linearized free-surface approximation was made, assuming $r_{surf} - R_{fixed} \simeq H_o$ when computing horizontal transports, either in the continuity equation or in tracer and momentum advection terms. This approximation is dropped when using the non-linear free-surface formulation and the total thickness, including the time varying part η, is considered when computing horizontal transports. Implications for the barotropic part are presented hereafter. In section Section 2.10.2.3 consequences for tracer conservation is briefly discussed (more details can be found in Campin et al. (2004) [CAHM04]) ; the general time-stepping is presented in section Section 2.10.2.4 with some limitations regarding the vertical resolution in section Section 2.10.2.5.

In the non-linear formulation, the continuous form of the model equations remains unchanged, except for the 2D continuity equation (2.11) which is now integrated from $R_{fixed}(x,y)$ up to $r_{surf} = R_o + \eta$:

$$\epsilon_{fs}\partial_t \eta = \dot{r}|_{r=r_{surf}} + \epsilon_{fw}(P-E) = -\nabla_h \cdot \int_{R_{fixed}}^{R_o+\eta} \vec{\mathbf{v}} dr + \epsilon_{fw}(P-E)$$

Since η has a direct effect on the horizontal velocity (through $\nabla_h \Phi_{surf}$), this adds a non-linear term to the free surface equation. Several options for the time discretization of this non-linear part can be considered, as detailed below.

If the column thickness is evaluated at time step n, and with implicit treatment of the surface potential gradient, equations (2.64) and (2.65) become:

$$\epsilon_{fs}\eta^{n+1} - \nabla_h \cdot \Delta t^2 (\eta^n + R_o - R_{fixed}) \nabla_h b_s \eta^{n+1} = \eta^*$$

where

$$\eta^* = \epsilon_{fs}\, \eta^n - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o+\eta^n} \vec{\mathbf{v}}^* dr \; + \; \epsilon_{fw} \Delta_t (P-E)^n$$

This method requires us to update the solver matrix at each time step.

Alternatively, the non-linear contribution can be evaluated fully explicitly:

$$\epsilon_{fs}\eta^{n+1} - \nabla_h \cdot \Delta t^2 (R_o - R_{fixed}) \nabla_h b_s \eta^{n+1} = \eta^* + \nabla_h \cdot \Delta t^2 (\eta^n) \nabla_h b_s \eta^n$$

This formulation allows one to keep the initial solver matrix unchanged though throughout the integration, since the non-linear free surface only affects the RHS.

Finally, another option is a "linearized" formulation where the total column thickness appears only in the integral term of the RHS (2.65) but not directly in the equation (2.64).

Those different options (see Table 2.1) have been tested and show little differences. However, we recommend the use of the most precise method (nonlinFreeSurf =4) since the computation cost involved in the solver matrix update is negligible.

Table 2.1: Non-linear free-surface flags

Parameter	Value	Description
nonlinFreeSurf	-1	linear free-surface, restart from a pickup file produced with #undef EXACT_CONSERV code
	0	linear free-surface (= default)
	4	full non-linear free-surface
	3	same as 4 but neglecting $\int_{R_o}^{R_o+\eta} b' dr$ in Φ'_{hyd}
	2	same as 3 but do not update cg2d solver matrix
	1	same as 2 but treat momentum as in linear free-surface
select_rStar	0	do not use r^* vertical coordinate (= default)
	2	use r^* vertical coordinate
	1	same as 2 but without the contribution of the slope of the coordinate in $\nabla\Phi$

2.10.2.3 Tracer conservation with non-linear free-surface

To ensure global tracer conservation (i.e., the total amount) as well as local conservation, the change in the surface level thickness must be consistent with the way the continuity equation is integrated, both in the barotropic part (to find η) and baroclinic part (to find $w = \dot{r}$).

To illustrate this, consider the shallow water model, with a source of fresh water (P):

$$\partial_t h + \nabla \cdot h\vec{\mathbf{v}} = P$$

where h is the total thickness of the water column. To conserve the tracer θ we have to discretize:

$$\partial_t(h\theta) + \nabla \cdot (h\theta\vec{\mathbf{v}}) = P\theta_{\rm rain}$$

Using the implicit (non-linear) free surface described above (Section 2.4) we have:

$$h^{n+1} = h^n - \Delta t \nabla \cdot (h^n\, \vec{\mathbf{v}}^{n+1}) + \Delta t P$$

The discretized form of the tracer equation must adopt the same "form" in the computation of tracer fluxes, that is, the same value of h, as used in the continuity equation:

$$h^{n+1}\,\theta^{n+1} = h^n\,\theta^n - \Delta t \nabla \cdot (h^n\,\theta^n\,\vec{\mathbf{v}}^{n+1}) + \Delta t P \theta_{rain}$$

The use of a 3 time-levels time-stepping scheme such as the Adams-Bashforth make the conservation sightly tricky. The current implementation with the Adams-Bashforth time-stepping provides an exact local conservation and prevents any drift in the global tracer content (Campin et al. (2004) [CAHM04]). Compared to the linear free-surface method, an additional step is required: the variation of the water column thickness (from h^n to h^{n+1}) is not incorporated directly into the tracer equation. Instead, the model uses the G_θ terms (first step) as in the linear free surface formulation (with the "*surface correction*" turned "on", see tracer section):

$$G_\theta^n = \left(-\nabla \cdot (h^n\,\theta^n\,\vec{\mathbf{v}}^{n+1}) - \dot{r}_{surf}^{n+1}\theta^n\right)/h^n$$

Then, in a second step, the thickness variation (expansion/reduction) is taken into account:

$$\theta^{n+1} = \theta^n + \Delta t \frac{h^n}{h^{n+1}}\left(G_\theta^{(n+1/2)} + P(\theta_{\rm rain} - \theta^n)/h^n\right)$$

Note that with a simple forward time step (no Adams-Bashforth), these two formulations are equivalent, since $(h^{n+1} - h^n)/\Delta t = P - \nabla \cdot (h^n\,\vec{\mathbf{v}}^{n+1}) = P + \dot{r}_{surf}^{n+1}$

2.10.2.4 Time stepping implementation of the non-linear free-surface

The grid cell thickness was hold constant with the linear free-surface; with the non-linear free-surface, it is now varying in time, at least at the surface level. This implies some modifications of the general algorithm described earlier in sections Section 2.7 and Section 2.8.

A simplified version of the staggered in time, non-linear free-surface algorithm is detailed hereafter, and can be compared to the equivalent linear free-surface case (eq. (2.36) to (2.46)) and can also be easily transposed to the synchronous time-stepping case. Among the simplifications, salinity equation, implicit operator and detailed elliptic equation are omitted. Surface forcing is explicitly written as fluxes of temperature, fresh water and momentum, $Q^{n+1/2}, P^{n+1/2}, F^n_{\mathbf{v}}$ respectively. h^n and dh^n are the column and grid box thickness in r-coordinate.

$$\phi^n_{hyd} = \int b(\theta^n, S^n, r) dr \tag{2.71}$$

$$\vec{\mathbf{G}}^{n-1/2}_{\vec{\mathbf{v}}} = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(dh^{n-1}, \vec{\mathbf{v}}^{n-1/2}) \;\; ; \;\; \vec{\mathbf{G}}^{(n)}_{\vec{\mathbf{v}}} = \frac{3}{2}\vec{\mathbf{G}}^{n-1/2}_{\vec{\mathbf{v}}} - \frac{1}{2}\vec{\mathbf{G}}^{n-3/2}_{\vec{\mathbf{v}}} \tag{2.72}$$

$$\vec{\mathbf{v}}^* = \vec{\mathbf{v}}^{n-1/2} + \Delta t \frac{dh^{n-1}}{dh^n}\left(\vec{\mathbf{G}}^{(n)}_{\vec{\mathbf{v}}} + F^n_{\vec{\mathbf{v}}}/dh^{n-1}\right) - \Delta t \nabla \phi^n_{hyd} \tag{2.73}$$

$$\longrightarrow \; update \;\; model \;\; geometry \; : \; \mathbf{hFac}(dh^n)$$

$$\begin{aligned} \eta^{n+1/2} &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{\mathbf{v}}^{n+1/2} dh^n \\ &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \left(\vec{\mathbf{v}}^* - g\Delta t \nabla \eta^{n+1/2}\right) dh^n \end{aligned} \tag{2.74}$$

$$\vec{\mathbf{v}}^{n+1/2} = \vec{\mathbf{v}}^* - g\Delta t \nabla \eta^{n+1/2} \tag{2.75}$$

$$h^{n+1} = h^n + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{\mathbf{v}}^{n+1/2} dh^n \tag{2.76}$$

$$G^n_\theta = G_\theta(dh^n, u^{n+1/2}, \theta^n) \;\; ; \;\; G^{(n+1/2)}_\theta = \frac{3}{2}G^n_\theta - \frac{1}{2}G^{n-1}_\theta \tag{2.77}$$

$$\theta^{n+1} = \theta^n + \Delta t \frac{dh^n}{dh^{n+1}}\left(G^{(n+1/2)}_\theta + (P^{n+1/2}(\theta_{\rm rain} - \theta^n) + Q^{n+1/2})/dh^n\right)$$

Two steps have been added to linear free-surface algorithm (eq. (2.36) to (2.46)): Firstly, the model "geometry" (here the **hFacC,W,S**) is updated just before entering SOLVE_FOR_PRESSURE, using the current dh^n field. Secondly, the vertically integrated continuity equation (2.76) has been added (exactConserv =.TRUE., in parameter file `data`, namelist `PARM01`) just before computing the vertical velocity, in subroutine INTEGR_CONTINUITY. Although this equation might appear redundant with (2.74), the integrated column thickness h^{n+1} will be different from $\eta^{n+1/2} + H$ in the following cases:

- when Crank-Nicolson time-stepping is used (see Section 2.10.1).
- when filters are applied to the flow field, after (2.75), and alter the divergence of the flow.
- when the solver does not iterate until convergence; for example, because a too large residual target was set (cg2dTargetResidual, parameter file `data`, namelist `PARM02`).

In this staggered time-stepping algorithm, the momentum tendencies are computed using dh^{n-1} geometry factors (2.72) and then rescaled in subroutine TIMESTEP, (2.73), similarly to tracer tendencies (see Section 2.10.2.3). The tracers are stepped forward later, using the recently updated flow field $\mathbf{v}^{n+1/2}$ and the corresponding model geometry dh^n to compute the tendencies (2.77); then the tendencies are rescaled by dh^n/dh^{n+1} to derive the new tracers values $(\theta, S)^{n+1}$ ((2.78), in subroutines CALC_GT, CALC_GS).

Note that the fresh-water input is added in a consistent way in the continuity equation and in the tracer equation, taking into account the fresh-water temperature $\theta_{\rm rain}$.

Regarding the restart procedure, two 2D fields h^{n-1} and $(h^n - h^{n-1})/\Delta t$ in addition to the standard state variables and tendencies ($\eta^{n-1/2}$, $\mathbf{v}^{n-1/2}$, θ^n, S^n, $\mathbf{G}_{\mathbf{v}}^{n-3/2}$, $G_{\theta,S}^{n-1}$) are stored in a "*pickup*" file. The model restarts reading this pickup file, then updates the model geometry according to h^{n-1}, and compute h^n and the vertical velocity before starting the main calling sequence (eq. (2.71) to (2.78), FORWARD_STEP).

S/R INTEGR_CONTINUITY

$h^{n+1} - H_o$: etaH (DYNVARS.h)
$h^n - H_o$: etaHnm1 (SURFACE.h)
$(h^{n+1} - h^n)/\Delta t$: dEtaHdt (SURFACE.h)

2.10.2.5 Non-linear free-surface and vertical resolution

When the amplitude of the free-surface variations becomes as large as the vertical resolution near the surface, the surface layer thickness can decrease to nearly zero or can even vanish completely. This later possibility has not been implemented, and a minimum relative thickness is imposed (hFacInf, parameter file `data`, namelist `PARM01`) to prevent numerical instabilities caused by very thin surface level.

A better alternative to the vanishing level problem relies on a different vertical coordinate r^* : The time variation of the total column thickness becomes part of the r^* coordinate motion, as in a σ_z, σ_p model, but the fixed part related to topography is treated as in a height or pressure coordinate model. A complete description is given in Adcroft and Campin (2004) [AC04].

The time-stepping implementation of the r^* coordinate is identical to the non-linear free-surface in r coordinate, and differences appear only in the spacial discretization.

2.11 Spatial discretization of the dynamical equations

Spatial discretization is carried out using the finite volume method. This amounts to a grid-point method (namely second-order centered finite difference) in the fluid interior but allows boundaries to intersect a regular grid allowing a more accurate representation of the position of the boundary. We treat the horizontal and vertical directions as separable and differently.

2.11.1 The finite volume method: finite volumes versus finite difference

The finite volume method is used to discretize the equations in space. The expression "finite volume" actually has two meanings; one is the method of embedded or intersecting boundaries (shaved or lopped cells in our terminology) and the other is non-linear interpolation methods that can deal with non-smooth solutions such as shocks (i.e. flux limiters for advection). Both make use of the integral form of the conservation laws to which the *weak solution* is a solution on each finite volume of (sub-domain). The weak solution can be constructed out of piece-wise constant elements or be differentiable. The differentiable equations can not be satisfied by piece-wise constant functions.

As an example, the 1-D constant coefficient advection-diffusion equation:

$$\partial_t \theta + \partial_x (u\theta - \kappa \partial_x \theta) = 0$$

can be discretized by integrating over finite sub-domains, i.e. the lengths Δx_i:

$$\Delta x \partial_t \theta + \delta_i(F) = 0$$

is exact if $\theta(x)$ is piece-wise constant over the interval Δx_i or more generally if θ_i is defined as the average over the interval Δx_i.

The flux, $F_{i-1/2}$, must be approximated:

$$F = u\overline{\theta} - \frac{\kappa}{\Delta x_c}\partial_i\theta$$

and this is where truncation errors can enter the solution. The method for obtaining $\overline{\theta}$ is unspecified and a wide range of possibilities exist including centered and upwind interpolation, polynomial fits based on the the volume average definitions of quantities and non-linear interpolation such as flux-limiters.

Choosing simple centered second-order interpolation and differencing recovers the same ODE's resulting from finite differencing for the interior of a fluid. Differences arise at boundaries where a boundary is not positioned on a regular or smoothly varying grid. This method is used to represent the topography using lopped cell, see Adcroft et al. (1997) [AHM97]. Subtle difference also appear in more than one dimension away from boundaries. This happens because each direction is discretized independently in the finite difference method while the integrating over finite volume implicitly treats all directions simultaneously.

2.11.2 C grid staggering of variables

The basic algorithm employed for stepping forward the momentum equations is based on retaining non-divergence of the flow at all times. This is most naturally done if the components of flow are staggered in space in the form of an Arakawa C grid (Arakawa and Lamb, 1977 [AL77]).

Figure 2.5 shows the components of flow (u,v,w) staggered in space such that the zonal component falls on the interface between continuity cells in the zonal direction. Similarly for the meridional and vertical directions. The continuity cell is synonymous with tracer cells (they are one and the same).

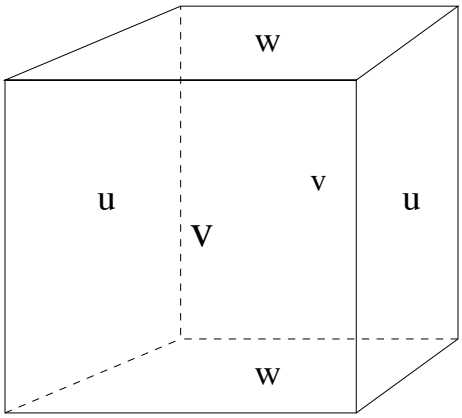

Figure 2.5: Three dimensional staggering of velocity components. This facilitates the natural discretization of the continuity and tracer equations.

2.11.3 Grid initialization and data

Initialization of grid data is controlled by subroutine INI_GRID which in calls INI_VERTICAL_GRID to initialize the vertical grid, and then either of INI_CARTESIAN_GRID, INI_SPHERICAL_POLAR_GRID or INI_CURVILINEAR_GRID to initialize the horizontal grid for cartesian, spherical-polar or curvilinear coordinates respectively.

The reciprocals of all grid quantities are pre-calculated and this is done in subroutine INI_MASKS_ETC which is called later by subroutine INITIALISE_FIXED.

All grid descriptors are global arrays and stored in common blocks in GRID.h and a generally declared as _RS.

2.11.4 Horizontal grid

The model domain is decomposed into tiles and within each tile a quasi-regular grid is used. A tile is the basic unit of domain decomposition for parallelization but may be used whether parallelized or not; see section [sec:domain_decomposition] for more details. Although the tiles may be patched together in an unstructured manner (i.e. irregular or non-tessilating pattern), the interior of tiles is a structured grid of quadrilateral cells. The horizontal coordinate system is orthogonal curvilinear meaning we can not necessarily treat the two horizontal directions as separable. Instead, each cell in the horizontal grid is described by the length of it's sides and it's area.

The grid information is quite general and describes any of the available coordinates systems, cartesian, spherical-polar or curvilinear. All that is necessary to distinguish between the coordinate systems is to initialize the grid data (descriptors) appropriately.

In the following, we refer to the orientation of quantities on the computational grid using geographic terminology such as points of the compass. This is purely for convenience but should not be confused with the actual geographic orientation of model quantities.

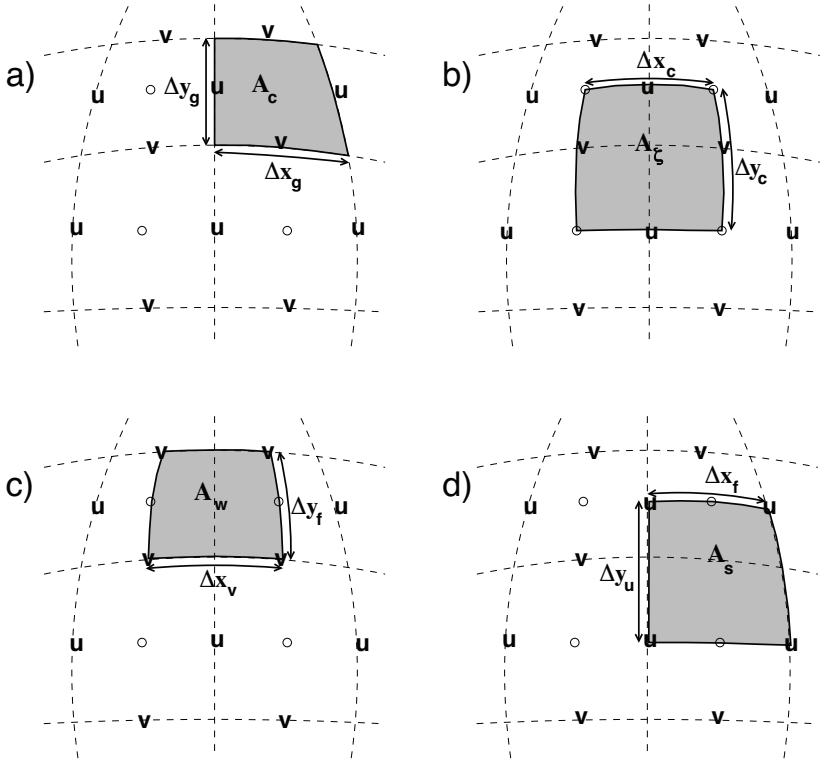

Figure 2.6: Staggering of horizontal grid descriptors (lengths and areas). The grid lines indicate the tracer cell boundaries and are the reference grid for all panels. a) The area of a tracer cell, A_c, is bordered by the lengths Δx_g and Δy_g. b) The area of a vorticity cell, A_ζ, is bordered by the lengths Δx_c and Δy_c. c) The area of a u cell, A_w, is bordered by the lengths Δx_v and Δy_f. d) The area of a v cell, A_s, is bordered by the lengths Δx_f and Δy_u.

Figure 2.6 (a) shows the tracer cell (synonymous with the continuity cell). The length of the southern edge, Δx_g, western edge, Δy_g and surface area, A_c, presented in the vertical are stored in arrays dxG, dyG and rA. The "g" suffix indicates that the lengths are along the defining grid boundaries. The "c" suffix associates the quantity with the cell

centers. The quantities are staggered in space and the indexing is such that **dxG(i,j)** is positioned to the south of **rA(i,j)** and **dyG(i,j)** positioned to the west.

Figure 2.6 (b) shows the vorticity cell. The length of the southern edge, Δx_c, western edge, Δy_c and surface area, A_ζ, presented in the vertical are stored in arrays dxC, dyC and rAz. The "z" suffix indicates that the lengths are measured between the cell centers and the "ζ" suffix associates points with the vorticity points. The quantities are staggered in space and the indexing is such that **dxC(i,j)** is positioned to the north of **rAz(i,j)** and **dyC(i,j)** positioned to the east.

Figure 2.6 (c) shows the "u" or western (w) cell. The length of the southern edge, Δx_v, eastern edge, Δy_f and surface area, A_w, presented in the vertical are stored in arrays dxV, dyF and rAw. The "v" suffix indicates that the length is measured between the v-points, the "f" suffix indicates that the length is measured between the (tracer) cell faces and the "w" suffix associates points with the u-points (w stands for west). The quantities are staggered in space and the indexing is such that **dxV(i,j)** is positioned to the south of **rAw(i,j)** and **dyF(i,j)** positioned to the east.

Figure 2.6 (d) shows the "v" or southern (s) cell. The length of the northern edge, Δx_f, western edge, Δy_u and surface area, A_s, presented in the vertical are stored in arrays dxF, dyU and rAs. The "u" suffix indicates that the length is measured between the u-points, the "f" suffix indicates that the length is measured between the (tracer) cell faces and the "s" suffix associates points with the v-points (s stands for south). The quantities are staggered in space and the indexing is such that **dxF(i,j)** is positioned to the north of **rAs(i,j)** and **dyU(i,j)** positioned to the west.

S/R INI_CARTESIAN_GRID , INI_SPHERICAL_POLAR_GRID , INI_CURVILINEAR_GRID

A_c, A_ζ, A_w, A_s : rA, rAz, rAw, rAs (GRID.h)
$\Delta x_g, \Delta y_g$: dxG, dyG (GRID.h)
$\Delta x_c, \Delta y_c$: dxC, dyC (GRID.h)
$\Delta x_f, \Delta y_f$: dxF, dyF (GRID.h)
$\Delta x_v, \Delta y_u$: dxV, dyU (GRID.h)

2.11.4.1 Reciprocals of horizontal grid descriptors

Lengths and areas appear in the denominator of expressions as much as in the numerator. For efficiency and portability, we pre-calculate the reciprocal of the horizontal grid quantities so that in-line divisions can be avoided.

For each grid descriptor (array) there is a reciprocal named using the prefix `recip_`. This doubles the amount of storage in GRID.h but they are all only 2-D descriptors.

S/R INI_MASKS_ETC

$A_c^{-1}, A_\zeta^{-1}, A_w^{-1}, A_s^{-1}$: recip_rA, recip_rAz, recip_rAw, recip_rAs (GRID.h)
$\Delta x_g^{-1}, \Delta y_g^{-1}$: recip_dxG, recip_dyG (GRID.h)
$\Delta x_c^{-1}, \Delta y_c^{-1}$: recip_dxC, recip_dyC (GRID.h)
$\Delta x_f^{-1}, \Delta y_f^{-1}$: recip_dxF, recip_dyF (GRID.h)
$\Delta x_v^{-1}, \Delta y_u^{-1}$: recip_dxV, recip_dyU (GRID.h)

2.11.4.2 Cartesian coordinates

Cartesian coordinates are selected when the logical flag usingCartesianGrid in namelist `PARM04` is set to true. The grid spacing can be set to uniform via scalars dXspacing and dYspacing in namelist `PARM04` or to variable resolution by the vectors DELX and DELY. Units are normally meters. Non-dimensional coordinates can be used by interpreting the gravitational constant as the Rayleigh number.

2.11.4.3 Spherical-polar coordinates

Spherical coordinates are selected when the logical flag usingSphericalPolarGrid in namelist `PARM04` is set to true. The grid spacing can be set to uniform via scalars dXspacing and dYspacing in namelist `PARM04` or to variable resolution by the vectors DELX and DELY. Units of these namelist variables are alway degrees. The horizontal grid descriptors are calculated from these namelist variables have units of meters.

2.11.4.4 Curvilinear coordinates

Curvilinear coordinates are selected when the logical flag usingCurvilinearGrid in namelist `PARM04` is set to true. The grid spacing can not be set via the namelist. Instead, the grid descriptors are read from data files, one for each descriptor. As for other grids, the horizontal grid descriptors have units of meters.

2.11.5 Vertical grid

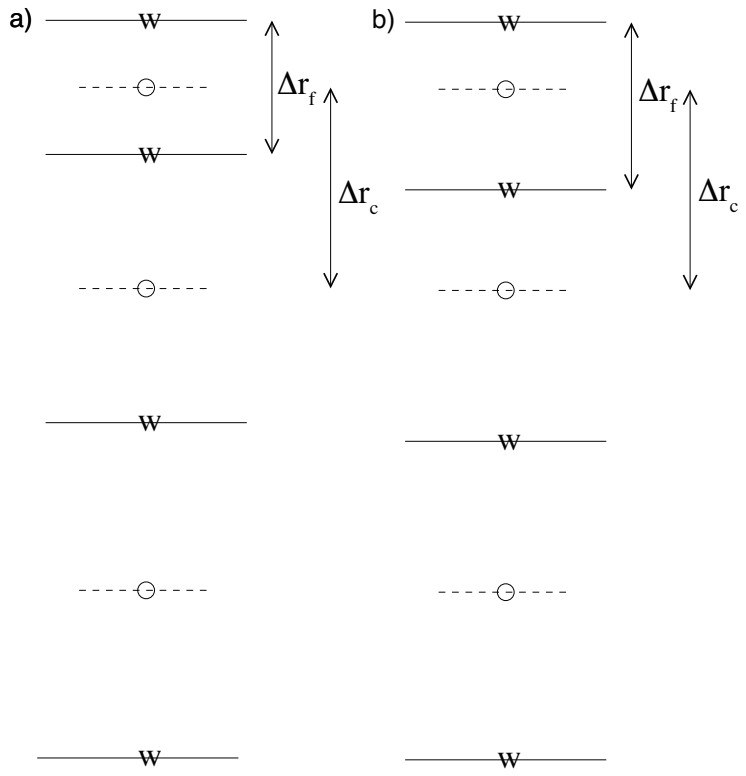

Figure 2.7: Two versions of the vertical grid. a) The cell centered approach where the interface depths are specified and the tracer points centered in between the interfaces. b) The interface centered approach where tracer levels are specified and the w-interfaces are centered in between.

As for the horizontal grid, we use the suffixes "c" and "f" to indicates faces and centers. Figure 2.7 (a) shows the default vertical grid used by the model. Δr_f is the difference in r (vertical coordinate) between the faces (i.e. $\Delta r_f \equiv -\delta_k r$ where the minus sign appears due to the convention that the surface layer has index $k = 1$.).

The vertical grid is calculated in subroutine INI_VERTICAL_GRID and specified via the vector delR in namelist `PARM04`. The units of "r" are either meters or Pascals depending on the isomorphism being used which in turn is

dependent only on the choice of equation of state.

There are alternative namelist vectors delZ and delP which dictate whether z- or p- coordinates are to be used but we intend to phase this out since they are redundant.

The reciprocals Δr_f^{-1} and Δr_c^{-1} are pre-calculated (also in subroutine INI_VERTICAL_GRID). All vertical grid descriptors are stored in common blocks in GRID.h.

The above grid Figure 2.7 (a) is known as the cell centered approach because the tracer points are at cell centers; the cell centers are mid-way between the cell interfaces. This discretization is selected when the thickness of the levels are provided (delR, parameter file `data`, namelist `PARM04`) An alternative, the vertex or interface centered approach, is shown in Figure 2.7 (b). Here, the interior interfaces are positioned mid-way between the tracer nodes (no longer cell centers). This approach is formally more accurate for evaluation of hydrostatic pressure and vertical advection but historically the cell centered approach has been used. An alternative form of subroutine INI_VERTICAL_GRID is used to select the interface centered approach This form requires to specify $Nr+1$ vertical distances delRc (parameter file `data`, namelist `PARM04`, e.g. ideal_2D_oce/input/data) corresponding to surface to center, $Nr-1$ center to center, and center to bottom distances.

S/R INI_VERTICAL_GRID

$\Delta r_f, \Delta r_c$: drF, drC (GRID.h)
$\Delta r_f^{-1}, \Delta r_c^{-1}$: recip_drF, recip_drC (GRID.h)

2.11.6 Topography: partially filled cells

Adcroft et al. (1997) [AHM97] presented two alternatives to the step-wise finite difference representation of topography. The method is known to the engineering community as *intersecting boundary method*. It involves allowing the boundary to intersect a grid of cells thereby modifying the shape of those cells intersected. We suggested allowing the topography to take on a piece-wise linear representation (shaved cells) or a simpler piecewise constant representation (partial step). Both show dramatic improvements in solution compared to the traditional full step representation, the piece-wise linear being the best. However, the storage requirements are excessive so the simpler piece-wise constant or partial-step method is all that is currently supported.

Figure 2.8 shows a schematic of the x-r plane indicating how the thickness of a level is determined at tracer and u points. The physical thickness of a tracer cell is given by $h_c(i,j,k)\Delta r_f(k)$ and the physical thickness of the open side is given by $h_w(i,j,k)\Delta r_f(k)$. Three 3-D descriptors h_c, h_w and h_s are used to describe the geometry: hFacC, hFacW and hFacS respectively. These are calculated in subroutine INI_MASKS_ETC along with there reciprocals recip_hFacC, recip_hFacW and recip_hFacS.

The non-dimensional fractions (or h-facs as we call them) are calculated from the model depth array and then processed to avoid tiny volumes. The rule is that if a fraction is less than hFacMin then it is rounded to the nearer of 0 or hFacMin or if the physical thickness is less than hFacMinDr then it is similarly rounded. The larger of the two methods is used when there is a conflict. By setting hFacMinDr equal to or larger than the thinnest nominal layers, $\min(\Delta z_f)$, but setting hFacMin to some small fraction then the model will only lop thick layers but retain stability based on the thinnest unlopped thickness; $\min(\Delta z_f, hFacMinDr)$.

S/R :filelink:INI_MASKS_ETC

h_c, h_w, h_s : hFacC, hFacW, hFacS (GRID.h)
$h_c^{-1}, h_w^{-1}, h_s^{-1}$: recip_hFacC, recip_hFacW, recip_hFacS (GRID.h)

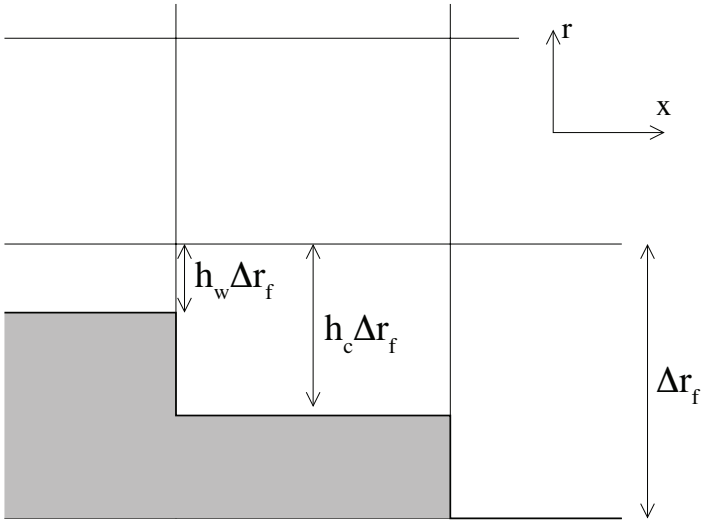

Figure 2.8: A schematic of the x-r plane showing the location of the non-dimensional fractions h_c and h_w . The physical thickness of a tracer cell is given by $h_c(i,j,k)\Delta r_f(k)$ and the physical thickness of the open side is given by $h_w(i,j,k)\Delta r_f(k)$.

2.12 Continuity and horizontal pressure gradient term

The core algorithm is based on the "C grid" discretization of the continuity equation which can be summarized as:

$$\partial_t u + \frac{1}{\Delta x_c}\delta_i \left.\frac{\partial \Phi}{\partial r}\right|_s \eta + \frac{\epsilon_{nh}}{\Delta x_c}\delta_i \Phi'_{nh} = G_u - \frac{1}{\Delta x_c}\delta_i \Phi'_h \tag{2.78}$$

$$\partial_t v + \frac{1}{\Delta y_c}\delta_j \left.\frac{\partial \Phi}{\partial r}\right|_s \eta + \frac{\epsilon_{nh}}{\Delta y_c}\delta_j \Phi'_{nh} = G_v - \frac{1}{\Delta y_c}\delta_j \Phi'_h \tag{2.79}$$

$$\epsilon_{nh}\left(\partial_t w + \frac{1}{\Delta r_c}\delta_k \Phi'_{nh}\right) = \epsilon_{nh} G_w + \overline{b}^k - \frac{1}{\Delta r_c}\delta_k \Phi'_h \tag{2.80}$$

$$\delta_i \Delta y_g \Delta r_f h_w u + \delta_j \Delta x_g \Delta r_f h_s v + \delta_k \mathcal{A}_c w = \mathcal{A}_c \delta_k (P-E)_{r=0} \tag{2.81}$$

where the continuity equation has been most naturally discretized by staggering the three components of velocity as shown in Figure 2.5. The grid lengths Δx_c and Δy_c are the lengths between tracer points (cell centers). The grid lengths Δx_g, Δy_g are the grid lengths between cell corners. Δr_f and Δr_c are the distance (in units of r) between level interfaces (w-level) and level centers (tracer level). The surface area presented in the vertical is denoted $\mathcal{A}_c$. The factors h_w and h_s are non-dimensional fractions (between 0 and 1) that represent the fraction cell depth that is "open" for fluid flow.

The last equation, the discrete continuity equation, can be summed in the vertical to yield the free-surface equation:

$$\mathcal{A}_c \partial_t \eta + \delta_i \sum_k \Delta y_g \Delta r_f h_w u + \delta_j \sum_k \Delta x_g \Delta r_f h_s v = \mathcal{A}_c (P-E)_{r=0} \tag{2.82}$$

The source term $P-E$ on the rhs of continuity accounts for the local addition of volume due to excess precipitation and run-off over evaporation and only enters the top-level of the ocean model.

2.13 Hydrostatic balance

The vertical momentum equation has the hydrostatic or quasi-hydrostatic balance on the right hand side. This discretization guarantees that the conversion of potential to kinetic energy as derived from the buoyancy equation exactly matches the form derived from the pressure gradient terms when forming the kinetic energy equation.

In the ocean, using z-coordinates, the hydrostatic balance terms are discretized:

$$\epsilon_{nh}\partial_t w + g\overline{\rho'}^k + \frac{1}{\Delta z}\delta_k \Phi'_h = \ldots \tag{2.83}$$

In the atmosphere, using p-coordinates, hydrostatic balance is discretized:

$$\overline{\theta'}^k + \frac{1}{\Delta \Pi}\delta_k \Phi'_h = 0 \tag{2.84}$$

where $\Delta\Pi$ is the difference in Exner function between the pressure points. The non-hydrostatic equations are not available in the atmosphere.

The difference in approach between ocean and atmosphere occurs because of the direct use of the ideal gas equation in forming the potential energy conversion term $\alpha\omega$. Because of the different representation of hydrostatic balance between ocean and atmosphere there is no elegant way to represent both systems using an arbitrary coordinate.

The integration for hydrostatic pressure is made in the positive r direction (increasing k-index). For the ocean, this is from the free-surface down and for the atmosphere this is from the ground up.

The calculations are made in the subroutine CALC_PHI_HYD. Inside this routine, one of other of the atmospheric/oceanic form is selected based on the string variable buoyancyRelation.

2.14 Flux-form momentum equations

The original finite volume model was based on the Eulerian flux form momentum equations. This is the default though the vector invariant form is optionally available (and recommended in some cases).

The "G's" (our colloquial name for all terms on rhs!) are broken into the various advective, Coriolis, horizontal dissipation, vertical dissipation and metric forces:

$$G_u = G_u^{adv} + G_u^{cor} + G_u^{h-diss} + G_u^{v-diss} + G_u^{metric} + G_u^{nh-metric} \tag{2.85}$$

$$G_v = G_v^{adv} + G_v^{cor} + G_v^{h-diss} + G_v^{v-diss} + G_v^{metric} + G_v^{nh-metric} \tag{2.86}$$

$$G_w = G_w^{adv} + G_w^{cor} + G_w^{h-diss} + G_w^{v-diss} + G_w^{metric} + G_w^{nh-metric} \tag{2.87}$$

In the hydrostatic limit, $G_w = 0$ and $\epsilon_{nh} = 0$, reducing the vertical momentum to hydrostatic balance.

These terms are calculated in routines called from subroutine MOM_FLUXFORM and collected into the global arrays gU, gV, and gW.

S/R MOM_FLUXFORM

G_u : gU (DYNVARS.h)
G_v : gV (DYNVARS.h)
G_w : gW (NH_VARS.h)

2.14.1 Advection of momentum

The advective operator is second order accurate in space:

$$\mathcal{A}_w \Delta r_f h_w G_u^{adv} = \delta_i \overline{U}^i \overline{u}^i + \delta_j \overline{V}^i \overline{u}^j + \delta_k \overline{W}^i \overline{u}^k \tag{2.88}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{adv} = \delta_i \overline{U}^j \overline{v}^i + \delta_j \overline{V}^j \overline{v}^j + \delta_k \overline{W}^j \overline{v}^k \tag{2.89}$$

$$\mathcal{A}_c \Delta r_c G_w^{adv} = \delta_i \overline{U}^k \overline{w}^i + \delta_j \overline{V}^k \overline{w}^j + \delta_k \overline{W}^k \overline{w}^k \tag{2.90}$$

and because of the flux form does not contribute to the global budget of linear momentum. The quantities U, V and W are volume fluxes defined:

$$U = \Delta y_g \Delta r_f h_w u \tag{2.91}$$

$$V = \Delta x_g \Delta r_f h_s v \tag{2.92}$$

$$W = \mathcal{A}_c w \tag{2.93}$$

The advection of momentum takes the same form as the advection of tracers but by a translated advective flow. Consequently, the conservation of second moments, derived for tracers later, applies to u^2 and v^2 and w^2 so that advection of momentum correctly conserves kinetic energy.

S/R MOM_U_ADV_UU, MOM_U_ADV_VU, MOM_U_ADV_WU

uu, vu, wu : fZon, fMer, fVerUkp (local to MOM_FLUXFORM.F)

S/R MOM_V_ADV_UV, MOM_V_ADV_VV, MOM_V_ADV_WV

uv, vv, wv : fZon, fMer, fVerVkp (local to MOM_FLUXFORM.F)

2.14.2 Coriolis terms

The "pure C grid" Coriolis terms (i.e. in absence of C-D scheme) are discretized:

$$\mathcal{A}_w \Delta r_f h_w G_u^{Cor} = \overline{f \mathcal{A}_c \Delta r_f h_c \overline{v}^j}^i - \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \overline{w}^k}^i \tag{2.94}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{Cor} = -\overline{f \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^j \tag{2.95}$$

$$\mathcal{A}_c \Delta r_c G_w^{Cor} = \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^k \tag{2.96}$$

where the Coriolis parameters f and f' are defined:

$$\begin{aligned} f &= 2\Omega \sin\varphi \\ f' &= 2\Omega \cos\varphi \end{aligned}$$

where φ is geographic latitude when using spherical geometry, otherwise the β-plane definition is used:

$$\begin{aligned} f &= f_o + \beta y \\ f' &= 0 \end{aligned}$$

This discretization globally conserves kinetic energy. It should be noted that despite the use of this discretization in former publications, all calculations to date have used the following different discretization:

$$G_u^{Cor} = f_u \overline{v}^{ji} - \epsilon_{nh} f'_u \overline{w}^{ik} \tag{2.97}$$

$$G_v^{Cor} = -f_v \overline{u}^{ij} \tag{2.98}$$

$$G_w^{Cor} = \epsilon_{nh} f'_w \overline{u}^{ik} \tag{2.99}$$

where the subscripts on f and f' indicate evaluation of the Coriolis parameters at the appropriate points in space. The above discretization does *not* conserve anything, especially energy, but for historical reasons is the default for the code. A flag controls this discretization: set run-time integer selectCoriScheme to two (=2) (which otherwise defaults to zero) to select the energy-conserving conserving form (2.94), (2.95), and (2.96) above.

S/R CD_CODE_SCHEME, MOM_U_CORIOLIS, MOM_V_CORIOLIS

G_u^{Cor}, G_v^{Cor} : cF (local to MOM_FLUXFORM.F)

2.14.3 Curvature metric terms

The most commonly used coordinate system on the sphere is the geographic system (λ, φ). The curvilinear nature of these coordinates on the sphere lead to some "metric" terms in the component momentum equations. Under the thin-atmosphere and hydrostatic approximations these terms are discretized:

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = \overline{\frac{\overline{u}^i}{a} \tan\varphi \mathcal{A}_c \Delta r_f h_c \overline{v}^j}^i \tag{2.100}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = -\overline{\frac{\overline{u}^i}{a} \tan\varphi \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^j \tag{2.101}$$

$$G_w^{metric} = 0 \tag{2.102}$$

where a is the radius of the planet (sphericity is assumed) or the radial distance of the particle (i.e. a function of height). It is easy to see that this discretization satisfies all the properties of the discrete Coriolis terms since the metric factor $\frac{u}{a}\tan\varphi$ can be viewed as a modification of the vertical Coriolis parameter: $f \rightarrow f + \frac{u}{a}\tan\varphi$.

However, as for the Coriolis terms, a non-energy conserving form has exclusively been used to date:

$$G_u^{metric} = \frac{u\overline{v}^{ij}}{a} \tan\varphi$$

$$G_v^{metric} = \frac{\overline{u}^{ij}\overline{u}^{ij}}{a} \tan\varphi$$

where $\tan\varphi$ is evaluated at the u and v points respectively.

S/R MOM_U_METRIC_SPHERE, MOM_V_METRIC_SPHERE

$G_u^{metric}, G_v^{metric}$: mT (local to MOM_FLUXFORM.F)

2.14.4 Non-hydrostatic metric terms

For the non-hydrostatic equations, dropping the thin-atmosphere approximation re-introduces metric terms involving w which are required to conserve angular momentum:

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = -\overline{\frac{\overline{u}^i \overline{w}^k}{a} \mathcal{A}_c \Delta r_f h_c}^i \tag{2.103}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = -\overline{\frac{\overline{v}^j \overline{w}^k}{a} \mathcal{A}_c \Delta r_f h_c}^j \tag{2.104}$$

$$\mathcal{A}_c \Delta r_c G_w^{metric} = \overline{\frac{\overline{u^i}^2 + \overline{v^j}^2}{a} \mathcal{A}_c \Delta r_f h_c}^k \tag{2.105}$$

Because we are always consistent, even if consistently wrong, we have, in the past, used a different discretization in the model which is:

$$\begin{aligned} G_u^{metric} = & \quad -\frac{u}{a}\overline{w}^{ik} \\ G_v^{metric} = & \quad -\frac{v}{a}\overline{w}^{jk} \\ G_w^{metric} = & \quad \frac{1}{a}(\overline{u}^{ik^2} + \overline{v}^{jk^2}) \end{aligned}$$

S/R MOM_U_METRIC_NH, MOM_V_METRIC_NH

$G_u^{metric}, G_v^{metric}$: mT (local to MOM_FLUXFORM.F)

2.14.5 Lateral dissipation

Historically, we have represented the SGS Reynolds stresses as simply down gradient momentum fluxes, ignoring constraints on the stress tensor such as symmetry.

$$\mathcal{A}_w \Delta r_f h_w G_u^{h-diss} = \delta_i \Delta y_f \Delta r_f h_c \tau_{11} + \delta_j \Delta x_v \Delta r_f h_\zeta \tau_{12} \tag{2.106}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{h-diss} = \delta_i \Delta y_u \Delta r_f h_\zeta \tau_{21} + \delta_j \Delta x_f \Delta r_f h_c \tau_{22} \tag{2.107}$$

The lateral viscous stresses are discretized:

$$\tau_{11} = A_h c_{11\Delta}(\varphi) \frac{1}{\Delta x_f} \delta_i u - A_4 c_{11\Delta^2}(\varphi) \frac{1}{\Delta x_f} \delta_i \nabla^2 u \tag{2.108}$$

$$\tau_{12} = A_h c_{12\Delta}(\varphi) \frac{1}{\Delta y_u} \delta_j u - A_4 c_{12\Delta^2}(\varphi) \frac{1}{\Delta y_u} \delta_j \nabla^2 u \tag{2.109}$$

$$\tau_{21} = A_h c_{21\Delta}(\varphi) \frac{1}{\Delta x_v} \delta_i v - A_4 c_{21\Delta^2}(\varphi) \frac{1}{\Delta x_v} \delta_i \nabla^2 v \tag{2.110}$$

$$\tau_{22} = A_h c_{22\Delta}(\varphi) \frac{1}{\Delta y_f} \delta_j v - A_4 c_{22\Delta^2}(\varphi) \frac{1}{\Delta y_f} \delta_j \nabla^2 v \tag{2.111}$$

where the non-dimensional factors $c_{lm\Delta^n}(\varphi), \{l, m, n\} \in \{1, 2\}$ define the "cosine" scaling with latitude which can be applied in various ad-hoc ways. For instance, $c_{11\Delta} = c_{21\Delta} = (\cos\varphi)^{3/2}$, $c_{12\Delta} = c_{22\Delta} = 1$ would represent the anisotropic cosine scaling typically used on the "lat-lon" grid for Laplacian viscosity.

It should be noted that despite the ad-hoc nature of the scaling, some scaling must be done since on a lat-lon grid the converging meridians make it very unlikely that a stable viscosity parameter exists across the entire model domain.

The Laplacian viscosity coefficient, A_h (viscAh), has units of $m^2 s^{-1}$. The bi-harmonic viscosity coefficient, A_4 (viscA4), has units of $m^4 s^{-1}$.

S/R MOM_U_XVISCFLUX, MOM_U_YVISCFLUX

τ_{11}, τ_{12} : vF, v4F (local to MOM_FLUXFORM.F)

S/R MOM_V_XVISCFLUX, MOM_V_YVISCFLUX

τ_{21}, τ_{22} : vF, v4F (local to MOM_FLUXFORM.F)

Two types of lateral boundary condition exist for the lateral viscous terms, no-slip and free-slip.

The free-slip condition is most convenient to code since it is equivalent to zero-stress on boundaries. Simple masking of the stress components sets them to zero. The fractional open stress is properly handled using the lopped cells.

The no-slip condition defines the normal gradient of a tangential flow such that the flow is zero on the boundary. Rather than modify the stresses by using complicated functions of the masks and "ghost" points (see Adcroft and Marshall (1998) [AM98]) we add the boundary stresses as an additional source term in cells next to solid boundaries. This has the advantage of being able to cope with "thin walls" and also makes the interior stress calculation (code) independent of the boundary conditions. The "body" force takes the form:

$$G_u^{side-drag} = \frac{4}{\Delta z_f}\overline{(1-h_\zeta)\frac{\Delta x_v}{\Delta y_u}}^{j}\left(A_h c_{12\Delta}(\varphi)u - A_4 c_{12\Delta^2}(\varphi)\nabla^2 u\right) \tag{2.112}$$

$$G_v^{side-drag} = \frac{4}{\Delta z_f}\overline{(1-h_\zeta)\frac{\Delta y_u}{\Delta x_v}}^{i}\left(A_h c_{21\Delta}(\varphi)v - A_4 c_{21\Delta^2}(\varphi)\nabla^2 v\right) \tag{2.113}$$

In fact, the above discretization is not quite complete because it assumes that the bathymetry at velocity points is deeper than at neighboring vorticity points, e.g. $1-h_w < 1-h_\zeta$

S/R MOM_U_SIDEDRAG, MOM_V_SIDEDRAG

$G_u^{side-drag}, G_v^{side-drag}$: vF (local to MOM_FLUXFORM.F)

2.14.6 Vertical dissipation

Vertical viscosity terms are discretized with only partial adherence to the variable grid lengths introduced by the finite volume formulation. This reduces the formal accuracy of these terms to just first order but only next to boundaries; exactly where other terms appear such as linear and quadratic bottom drag.

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w}\delta_k \tau_{13} \tag{2.114}$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s}\delta_k \tau_{23} \tag{2.115}$$

$$G_w^{v-diss} = \epsilon_{nh}\frac{1}{\Delta r_f h_d}\delta_k \tau_{33} \tag{2.116}$$

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

$$\begin{aligned}\tau_{13} &= A_v\frac{1}{\Delta r_c}\delta_k u\\ \tau_{23} &= A_v\frac{1}{\Delta r_c}\delta_k v\\ \tau_{33} &= A_v\frac{1}{\Delta r_f}\delta_k w\end{aligned}$$

It should be noted that in the non-hydrostatic form, the stress tensor is even less consistent than for the hydrostatic (see Wajsowicz (1993) [Waj93]). It is well known how to do this properly (see Griffies and Hallberg (2000) [GH00]) and is on the list of to-do's.

S/R MOM_U_RVISCFLUX, MOM_V_RVISCFLUX

τ_{13} : fVrUp, fVrDw (local to MOM_FLUXFORM.F)
τ_{23} : fVrUp, fVrDw (local to MOM_FLUXFORM.F)

As for the lateral viscous terms, the free-slip condition is equivalent to simply setting the stress to zero on boundaries. The no-slip condition is implemented as an additional term acting on top of the interior and free-slip stresses. Bottom drag represents additional friction, in addition to that imposed by the no-slip condition at the bottom. The drag is cast as a stress expressed as a linear or quadratic function of the mean flow in the layer above the topography:

$$\tau_{13}^{bottom-drag} = \left(2A_v \frac{1}{\Delta r_c} + r_b + C_d \sqrt{\overline{2KE}^i} \right) u \tag{2.117}$$

$$\tau_{23}^{bottom-drag} = \left(2A_v \frac{1}{\Delta r_c} + r_b + C_d \sqrt{\overline{2KE}^j} \right) v \tag{2.118}$$

where these terms are only evaluated immediately above topography. r_b (bottomDragLinear) has units of ms^{-1} and a typical value of the order 0.0002 ms^{-1}. C_d (bottomDragQuadratic) is dimensionless with typical values in the range 0.001–0.003.

S/R MOM_U_BOTTOMDRAG, MOM_V_BOTTOMDRAG

$\tau_{13}^{bottom-drag}/\Delta r_f, \tau_{23}^{bottom-drag}/\Delta r_f$: vF (local to MOM_FLUXFORM.F)

2.14.7 Derivation of discrete energy conservation

These discrete equations conserve kinetic plus potential energy using the following definitions:

$$KE = \frac{1}{2} \left(\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh} \overline{w^2}^k \right) \tag{2.119}$$

2.14.8 Mom Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
VISCAHZ | 15 |SZ      MR      |m^2/s           |Harmonic Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VISCA4Z | 15 |SZ      MR      |m^4/s           |Biharmonic Visc Coefficient (m4/s)␣
↪(Zeta Pt)
VISCAHD | 15 |SM      MR      |m^2/s           |Harmonic Viscosity Coefficient (m2/s)␣
↪(Div Pt)
VISCA4D | 15 |SM      MR      |m^4/s           |Biharmonic Viscosity Coefficient (m4/
↪s) (Div Pt)
VAHZMAX | 15 |SZ      MR      |m^2/s           |CFL-MAX Harm Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VA4ZMAX | 15 |SZ      MR      |m^4/s           |CFL-MAX Biharm Visc Coefficient (m4/
↪s) (Zeta Pt)
VAHDMAX | 15 |SM      MR      |m^2/s           |CFL-MAX Harm Visc Coefficient (m2/s)␣
↪(Div Pt)
VA4DMAX | 15 |SM      MR      |m^4/s           |CFL-MAX Biharm Visc Coefficient (m4/
↪s) (Div Pt)
VAHZMIN | 15 |SZ      MR      |m^2/s           |RE-MIN Harm Visc Coefficient (m2/s)␣
↪(Zeta Pt)
```

(continues on next page)

(continued from previous page)

```
VA4ZMIN | 15 |SZ      MR      |m^4/s          |RE-MIN Biharm Visc Coefficient (m4/s)␣
↪(Zeta Pt)
VAHDMIN | 15 |SM      MR      |m^2/s          |RE-MIN Harm Visc Coefficient (m2/s)␣
↪(Div Pt)
VA4DMIN | 15 |SM      MR      |m^4/s          |RE-MIN Biharm Visc Coefficient (m4/s)␣
↪(Div Pt)
VAHZLTH | 15 |SZ      MR      |m^2/s          |Leith Harm Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VA4ZLTH | 15 |SZ      MR      |m^4/s          |Leith Biharm Visc Coefficient (m4/s)␣
↪(Zeta Pt)
VAHDLTH | 15 |SM      MR      |m^2/s          |Leith Harm Visc Coefficient (m2/s)␣
↪(Div Pt)
VA4DLTH | 15 |SM      MR      |m^4/s          |Leith Biharm Visc Coefficient (m4/s)␣
↪(Div Pt)
VAHZLTHD| 15 |SZ      MR      |m^2/s          |LeithD Harm Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VA4ZLTHD| 15 |SZ      MR      |m^4/s          |LeithD Biharm Visc Coefficient (m4/s)␣
↪(Zeta Pt)
VAHDLTHD| 15 |SM      MR      |m^2/s          |LeithD Harm Visc Coefficient (m2/s)␣
↪(Div Pt)
VA4DLTHD| 15 |SM      MR      |m^4/s          |LeithD Biharm Visc Coefficient (m4/s)␣
↪(Div Pt)
VAHZSMAG| 15 |SZ      MR      |m^2/s          |Smagorinsky Harm Visc Coefficient (m2/
↪s) (Zeta Pt)
VA4ZSMAG| 15 |SZ      MR      |m^4/s          |Smagorinsky Biharm Visc Coeff. (m4/s)␣
↪(Zeta Pt)
VAHDSMAG| 15 |SM      MR      |m^2/s          |Smagorinsky Harm Visc Coefficient (m2/
↪s) (Div Pt)
VA4DSMAG| 15 |SM      MR      |m^4/s          |Smagorinsky Biharm Visc Coeff. (m4/s)␣
↪(Div Pt)
momKE   | 15 |SM      MR      |m^2/s^2        |Kinetic Energy (in momentum Eq.)
momHDiv | 15 |SM      MR      |s^-1           |Horizontal Divergence (in momentum Eq.
↪)
momVort3| 15 |SZ      MR      |s^-1           |3rd component (vertical) of Vorticity
Strain  | 15 |SZ      MR      |s^-1           |Horizontal Strain of Horizontal␣
↪Velocities
Tension | 15 |SM      MR      |s^-1           |Horizontal Tension of Horizontal␣
↪Velocities
UBotDrag| 15 |UU   129MR      |m/s^2          |U momentum tendency from Bottom Drag
VBotDrag| 15 |VV   128MR      |m/s^2          |V momentum tendency from Bottom Drag
USidDrag| 15 |UU   131MR      |m/s^2          |U momentum tendency from Side Drag
VSidDrag| 15 |VV   130MR      |m/s^2          |V momentum tendency from Side Drag
Um_Diss | 15 |UU   133MR      |m/s^2          |U momentum tendency from Dissipation
Vm_Diss | 15 |VV   132MR      |m/s^2          |V momentum tendency from Dissipation
Um_Advec| 15 |UU   135MR      |m/s^2          |U momentum tendency from Advection␣
↪terms
Vm_Advec| 15 |VV   134MR      |m/s^2          |V momentum tendency from Advection␣
↪terms
Um_Cori | 15 |UU   137MR      |m/s^2          |U momentum tendency from Coriolis term
Vm_Cori | 15 |VV   136MR      |m/s^2          |V momentum tendency from Coriolis term
Um_Ext  | 15 |UU   137MR      |m/s^2          |U momentum tendency from external␣
↪forcing
Vm_Ext  | 15 |VV   138MR      |m/s^2          |V momentum tendency from external␣
↪forcing
Um_AdvZ3| 15 |UU   141MR      |m/s^2          |U momentum tendency from Vorticity␣
↪Advection
Vm_AdvZ3| 15 |VV   140MR      |m/s^2          |V momentum tendency from Vorticity␣
↪Advection
```

(continues on next page)

(continued from previous page)

```
Um_AdvRe| 15 |UU   143MR      |m/s^2          |U momentum tendency from vertical␣
↪Advection (Explicit part)
Vm_AdvRe| 15 |VV   142MR      |m/s^2          |V momentum tendency from vertical␣
↪Advection (Explicit part)
ADVx_Um | 15 |UM   145MR      |m^4/s^2        |Zonal      Advective Flux of U␣
↪momentum
ADVy_Um | 15 |VZ   144MR      |m^4/s^2        |Meridional Advective Flux of U␣
↪momentum
ADVrE_Um| 15 |WU      LR      |m^4/s^2        |Vertical   Advective Flux of U␣
↪momentum (Explicit part)
ADVx_Vm | 15 |UZ   148MR      |m^4/s^2        |Zonal      Advective Flux of V␣
↪momentum
ADVy_Vm | 15 |VM   147MR      |m^4/s^2        |Meridional Advective Flux of V␣
↪momentum
ADVrE_Vm| 15 |WV      LR      |m^4/s^2        |Vertical   Advective Flux of V␣
↪momentum (Explicit part)
VISCx_Um| 15 |UM   151MR      |m^4/s^2        |Zonal      Viscous Flux of U momentum
VISCy_Um| 15 |VZ   150MR      |m^4/s^2        |Meridional Viscous Flux of U momentum
VISrE_Um| 15 |WU      LR      |m^4/s^2        |Vertical   Viscous Flux of U momentum␣
↪(Explicit part)
VISrI_Um| 15 |WU      LR      |m^4/s^2        |Vertical   Viscous Flux of U momentum␣
↪(Implicit part)
VISCx_Vm| 15 |UZ   155MR      |m^4/s^2        |Zonal      Viscous Flux of V momentum
VISCy_Vm| 15 |VM   154MR      |m^4/s^2        |Meridional Viscous Flux of V momentum
VISrE_Vm| 15 |WV      LR      |m^4/s^2        |Vertical   Viscous Flux of V momentum␣
↪(Explicit part)
VISrI_Vm| 15 |WV      LR      |m^4/s^2        |Vertical   Viscous Flux of V momentum␣
↪(Implicit part)
```

2.15 Vector invariant momentum equations

The finite volume method lends itself to describing the continuity and tracer equations in curvilinear coordinate systems. However, in curvilinear coordinates many new metric terms appear in the momentum equations (written in Lagrangian or flux-form) making generalization far from elegant. Fortunately, an alternative form of the equations, the vector invariant equations are exactly that; invariant under coordinate transformations so that they can be applied uniformly in any orthogonal curvilinear coordinate system such as spherical coordinates, boundary following or the conformal spherical cube system.

The non-hydrostatic vector invariant equations read:

$$\partial_t \vec{v} + (2\vec{\Omega} + \vec{\zeta}) \wedge \vec{v} - b\hat{r} + \vec{\nabla} B = \vec{\nabla} \cdot \vec{\tau} \tag{2.120}$$

which describe motions in any orthogonal curvilinear coordinate system. Here, B is the Bernoulli function and $\vec{\zeta} = \nabla \wedge \vec{v}$ is the vorticity vector. We can take advantage of the elegance of these equations when discretizing them and use the discrete definitions of the grad, curl and divergence operators to satisfy constraints. We can also consider the analogy to forming derived equations, such as the vorticity equation, and examine how the discretization can be adjusted to give suitable vorticity advection among other things.

The underlying algorithm is the same as for the flux form equations. All that has changed is the contents of the "G's". For the time-being, only the hydrostatic terms have been coded but we will indicate the points where non-hydrostatic contributions will enter:

$$G_u = G_u^{fv} + G_u^{\zeta_3 v} + G_u^{\zeta_2 w} + G_u^{\partial_x B} + G_u^{\partial_z \tau^x} + G_u^{h-dissip} + G_u^{v-dissip} \tag{2.121}$$

$$G_v = G_v^{fu} + G_v^{\zeta_3 u} + G_v^{\zeta_1 w} + G_v^{\partial_y B} + G_v^{\partial_z \tau^y} + G_v^{h-dissip} + G_v^{v-dissip} \tag{2.122}$$

$$G_w = G_w^{fu} + G_w^{\zeta_1 v} + G_w^{\zeta_2 u} + G_w^{\partial_z B} + G_w^{h-dissip} + G_w^{v-dissip} \tag{2.123}$$

S/R MOM_VECINV

G_u : gU (DYNVARS.h)
G_v : gV (DYNVARS.h)
G_w : gW (NH_VARS.h)

2.15.1 Relative vorticity

The vertical component of relative vorticity is explicitly calculated and use in the discretization. The particular form is crucial for numerical stability; alternative definitions break the conservation properties of the discrete equations.

Relative vorticity is defined:

$$\zeta_3 = \frac{\Gamma}{A_\zeta} = \frac{1}{\mathcal{A}_\zeta}(\delta_i \Delta y_c v - \delta_j \Delta x_c u) \tag{2.124}$$

where $\mathcal{A}_\zeta$ is the area of the vorticity cell presented in the vertical and Γ is the circulation about that cell.

S/R MOM_CALC_RELVORT3

ζ_3 : vort3 (local to MOM_VECINV.F)

2.15.2 Kinetic energy

The kinetic energy, denoted KE, is defined:

$$KE = \frac{1}{2}(\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh}\overline{w^2}^k) \tag{2.125}$$

S/R MOM_CALC_KE

KE : KE (local to MOM_VECINV.F)

2.15.3 Coriolis terms

The potential enstrophy conserving form of the linear Coriolis terms are written:

$$G_u^{fv} = \frac{1}{\Delta x_c}\overline{\frac{f}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \tag{2.126}$$

$$G_v^{fu} = -\frac{1}{\Delta y_c}\overline{\frac{f}{h_\zeta}}^i \overline{\overline{\Delta y_g h_w u}^i}^j \tag{2.127}$$

Here, the Coriolis parameter f is defined at vorticity (corner) points.

The potential enstrophy conserving form of the non-linear Coriolis terms are written:

$$G_u^{\zeta_3 v} = \frac{1}{\Delta x_c}\overline{\frac{\zeta_3}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \tag{2.128}$$

$$G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c}\overline{\frac{\zeta_3}{h_\zeta}}^{i}\overline{\overline{\Delta y_g h_w u}^{i}}^{j} \tag{2.129}$$

The Coriolis terms can also be evaluated together and expressed in terms of absolute vorticity $f + \zeta_3$. The potential enstrophy conserving form using the absolute vorticity is written:

$$G_u^{fv} + G_u^{\zeta_3 v} = \frac{1}{\Delta x_c}\overline{\frac{f+\zeta_3}{h_\zeta}}^{j}\overline{\overline{\Delta x_g h_s v}^{j}}^{i} \tag{2.130}$$

$$G_v^{fu} + G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c}\overline{\frac{f+\zeta_3}{h_\zeta}}^{i}\overline{\overline{\Delta y_g h_w u}^{i}}^{j} \tag{2.131}$$

The distinction between using absolute vorticity or relative vorticity is useful when constructing higher order advection schemes; monotone advection of relative vorticity behaves differently to monotone advection of absolute vorticity. Currently the choice of relative/absolute vorticity, centered/upwind/high order advection is available only through commented subroutine calls.

S/R MOM_VI_CORIOLIS, MOM_VI_U_CORIOLIS, MOM_VI_V_CORIOLIS

$G_u^{fv}, G_u^{\zeta_3 v}$: uCf (local to MOM_VECINV.F)
$G_v^{fu}, G_v^{\zeta_3 u}$: vCf (local to MOM_VECINV.F)

2.15.4 Shear terms

The shear terms ($\zeta_2 w$ and $\zeta_1 w$) are are discretized to guarantee that no spurious generation of kinetic energy is possible; the horizontal gradient of Bernoulli function has to be consistent with the vertical advection of shear:

$$G_u^{\zeta_2 w} = \frac{1}{\mathcal{A}_w \Delta r_f h_w}\overline{\overline{\mathcal{A}_c w}^{i}(\delta_k u - \epsilon_{nh}\delta_j w)}^{k} \tag{2.132}$$

$$G_v^{\zeta_1 w} = \frac{1}{\mathcal{A}_s \Delta r_f h_s}\overline{\overline{\mathcal{A}_c w}^{i}(\delta_k u - \epsilon_{nh}\delta_j w)}^{k} \tag{2.133}$$

S/R MOM_VI_U_VERTSHEAR, MOM_VI_V_VERTSHEAR

$G_u^{\zeta_2 w}$: uCf (local to MOM_VECINV.F)
$G_v^{\zeta_1 w}$: vCf (local to MOM_VECINV.F)

2.15.5 Gradient of Bernoulli function

$$G_u^{\partial_x B} = \frac{1}{\Delta x_c}\delta_i(\phi' + KE) \tag{2.134}$$

$$G_v^{\partial_y B} = \frac{1}{\Delta x_y}\delta_j(\phi' + KE) \tag{2.135}$$

S/R MOM_VI_U_GRAD_KE, MOM_VI_V_GRAD_KE

$G_u^{\partial_x KE}$: uCf (local to MOM_VECINV.F)
$G_v^{\partial_y KE}$: vCf (local to MOM_VECINV.F)

2.15.6 Horizontal divergence

The horizontal divergence, a complimentary quantity to relative vorticity, is used in parameterizing the Reynolds stresses and is discretized:

$$D = \frac{1}{\mathcal{A}_c h_c}(\delta_i \Delta y_g h_w u + \delta_j \Delta x_g h_s v) \tag{2.136}$$

S/R MOM_CALC_KE

D : hDiv (local to MOM_VECINV.F)

2.15.7 Horizontal dissipation

The following discretization of horizontal dissipation conserves potential vorticity (thickness weighted relative vorticity) and divergence and dissipates energy, enstrophy and divergence squared:

$$G_u^{h-dissip} = \frac{1}{\Delta x_c}\delta_i(A_D D - A_{D4} D^*) - \frac{1}{\Delta y_u h_w}\delta_j h_\zeta (A_\zeta \zeta - A_{\zeta 4}\zeta^*) \tag{2.137}$$

$$G_v^{h-dissip} = \frac{1}{\Delta x_v h_s}\delta_i h_\zeta (A_\zeta \zeta - A_\zeta \zeta^*) + \frac{1}{\Delta y_c}\delta_j (A_D D - A_{D4} D^*) \tag{2.138}$$

where

$$\begin{aligned} D^* = &\quad \frac{1}{\mathcal{A}_c h_c}(\delta_i \Delta y_g h_w \nabla^2 u + \delta_j \Delta x_g h_s \nabla^2 v) \\ \zeta^* = &\quad \frac{1}{\mathcal{A}_\zeta}(\delta_i \Delta y_c \nabla^2 v - \delta_j \Delta x_c \nabla^2 u) \end{aligned}$$

S/R MOM_VI_HDISSIP

$G_u^{h-dissip}$: uDissip (local to MOM_VI_HDISSIP.F)
$G_v^{h-dissip}$: vDissip (local to MOM_VI_HDISSIP.F)

2.15.8 Vertical dissipation

Currently, this is exactly the same code as the flux form equations.

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w}\delta_k \tau_{13} \tag{2.139}$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s}\delta_k \tau_{23} \tag{2.140}$$

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

$$\begin{aligned} \tau_{13} = &\quad A_v \frac{1}{\Delta r_c}\delta_k u \\ \tau_{23} = &\quad A_v \frac{1}{\Delta r_c}\delta_k v \end{aligned}$$

S/R MOM_U_RVISCFLUX, MOM_V_RVISCFLUX

τ_{13}, τ_{23} : vrf (local to MOM_VECINV.F)

2.16 Tracer equations

The basic discretization used for the tracer equations is the second order piece-wise constant finite volume form of the forced advection-diffusion equations. There are many alternatives to second order method for advection and alternative parameterizations for the sub-grid scale processes. The Gent-McWilliams eddy parameterization, KPP mixing scheme and PV flux parameterization are all dealt with in separate sections. The basic discretization of the advection-diffusion part of the tracer equations and the various advection schemes will be described here.

2.16.1 Time-stepping of tracers: ABII

The default advection scheme is the centered second order method which requires a second order or quasi-second order time-stepping scheme to be stable. Historically this has been the quasi-second order Adams-Bashforth method (ABII) and applied to all terms. For an arbitrary tracer, τ, the forced advection-diffusion equation reads:

$$\partial_t \tau + G^\tau_{adv} = G^\tau_{diff} + G^\tau_{forc} \tag{2.141}$$

where G^τ_{adv}, G^τ_{diff} and G^τ_{forc} are the tendencies due to advection, diffusion and forcing, respectively, namely:

$$G^\tau_{adv} = \partial_x u\tau + \partial_y v\tau + \partial_r w\tau - \tau\nabla\cdot\mathbf{v} \tag{2.142}$$

$$G^\tau_{diff} = \nabla\cdot\mathbf{K}\nabla\tau \tag{2.143}$$

and the forcing can be some arbitrary function of state, time and space.

The term, $\tau\nabla\cdot\mathbf{v}$, is required to retain local conservation in conjunction with the linear implicit free-surface. It only affects the surface layer since the flow is non-divergent everywhere else. This term is therefore referred to as the surface correction term. Global conservation is not possible using the flux-form (as here) and a linearized free-surface (Griffies and Hallberg (2000) [GH00] , Campin et al. (2004) [CAHM04]).

The continuity equation can be recovered by setting $G_{diff} = G_{forc} = 0$ and $\tau = 1$.

The driver routine that calls the routines to calculate tendencies are CALC_GT and CALC_GS for temperature and salt (moisture), respectively. These in turn call a generic advection diffusion routine GAD_CALC_RHS that is called with the flow field and relevant tracer as arguments and returns the collective tendency due to advection and diffusion. Forcing is add subsequently in CALC_GT or CALC_GS to the same tendency array.

S/R GAD_CALC_RHS

τ : tau (argument)
$G^{(n)}$: gTracer (argument)
F_r : fVerT (argument)

The space and time discretization are treated separately (method of lines). Tendencies are calculated at time levels n and $n-1$ and extrapolated to $n+1/2$ using the Adams-Bashforth method:

$$G^{(n+1/2)} = (\frac{3}{2}+\epsilon)G^{(n)} - (\frac{1}{2}+\epsilon)G^{(n-1)} \tag{2.144}$$

where $G^{(n)} = G^\tau_{adv} + G^\tau_{diff} + G^\tau_{src}$ at time step n. The tendency at $n-1$ is not re-calculated but rather the tendency at n is stored in a global array for later re-use.

S/R ADAMS_BASHFORTH2

$G^{(n+1/2)}$: gTracer (argument on exit)

$G^{(n)}$: gTracer (argument on entry)
$G^{(n-1)}$: gTrNm1 (argument)
ϵ : ABeps (PARAMS.h)

The tracers are stepped forward in time using the extrapolated tendency:

$$\tau^{(n+1)} = \tau^{(n)} + \Delta t G^{(n+1/2)} \tag{2.145}$$

S/R TIMESTEP_TRACER

$\tau^{(n+1)}$: gTracer (argument on exit)
$\tau^{(n)}$: tracer (argument on entry)
$G^{(n+1/2)}$: gTracer (argument)
Δt : deltaTtracer (PARAMS.h)

Strictly speaking the ABII scheme should be applied only to the advection terms. However, this scheme is only used in conjunction with the standard second, third and fourth order advection schemes. Selection of any other advection scheme disables Adams-Bashforth for tracers so that explicit diffusion and forcing use the forward method.

2.17 Advection schemes

2.17.1 Linear advection schemes

The advection schemes known as centered second order, centered fourth order, first order upwind and upwind biased third order are known as linear advection schemes because the coefficient for interpolation of the advected tracer are linear and a function only of the flow, not the tracer field it self. We discuss these first since they are most commonly used in the field and most familiar.

2.17.1.1 Centered second order advection-diffusion

The basic discretization, centered second order, is the default. It is designed to be consistent with the continuity equation to facilitate conservation properties analogous to the continuum. However, centered second order advection is notoriously noisy and must be used in conjunction with some finite amount of diffusion to produce a sensible solution.

The advection operator is discretized:

$$\mathcal{A}_c \Delta r_f h_c G^{\tau}_{adv} = \delta_i F_x + \delta_j F_y + \delta_k F_r \tag{2.146}$$

where the area integrated fluxes are given by:

$$\begin{aligned} F_x &= U\overline{\tau}^i \\ F_y &= V\overline{\tau}^j \\ F_r &= W\overline{\tau}^k \end{aligned}$$

The quantities U, V and W are volume fluxes. defined as:

$$\begin{aligned} U &= \Delta y_g \Delta r_f h_w u \\ V &= \Delta x_g \Delta r_f h_s v \\ W &= \mathcal{A}_c w \end{aligned}$$

For non-divergent flow, this discretization can be shown to conserve the tracer both locally and globally and to globally conserve tracer variance, τ^2. The proof is given in Adcroft (1995) [Adc95] and Adcroft et al. (1997) [AHM97] .

S/R GAD_C2_ADV_X

F_x : uT (argument)
U : uTrans (argument)
τ : tracer (argument)

S/R GAD_C2_ADV_Y

F_y : vT (argument)
V : vTrans (argument)
τ : tracer (argument)

S/R GAD_C2_ADV_R

F_r : wT (argument)
W : rTrans (argument)
τ : tracer (argument)

2.17.1.2 Third order upwind bias advection

Upwind biased third order advection offers a relatively good compromise between accuracy and smoothness. It is not a "positive" scheme meaning false extrema are permitted but the amplitude of such are significantly reduced over the centered second order method.

The third order upwind fluxes are discretized:

$$
\begin{aligned}
F_x = &\quad \overline{U\tau - \frac{1}{6}\delta_{ii}\tau}^{i} + \frac{1}{2}|U|\delta_i\frac{1}{6}\delta_{ii}\tau \\
F_y = &\quad \overline{V\tau - \frac{1}{6}\delta_{ii}\tau}^{j} + \frac{1}{2}|V|\delta_j\frac{1}{6}\delta_{jj}\tau \\
F_r = &\quad \overline{W\tau - \frac{1}{6}\delta_{ii}\tau}^{k} + \frac{1}{2}|W|\delta_k\frac{1}{6}\delta_{kk}\tau
\end{aligned}
$$

At boundaries, $\delta_{\hat{n}}\tau$ is set to zero allowing δ_{nn} to be evaluated. We are currently examine the accuracy of this boundary condition and the effect on the solution.

S/R GAD_U3_ADV_X

F_x : uT (argument)
U : uTrans (argument)
τ : tracer (argument)

S/R GAD_U3_ADV_Y

F_y : vT (argument)

V : vTrans (argument)
τ : tracer (argument)

S/R GAD_U3_ADV_R

F_r : wT (argument)
W : rTrans (argument)
τ : tracer (argument)

2.17.1.3 Centered fourth order advection

Centered fourth order advection is formally the most accurate scheme we have implemented and can be used to great effect in high resolution simulations where dynamical scales are well resolved. However, the scheme is noisy, like the centered second order method, and so must be used with some finite amount of diffusion. Bi-harmonic is recommended since it is more scale selective and less likely to diffuse away the well resolved gradient the fourth order scheme worked so hard to create.

The centered fourth order fluxes are discretized:

$$
\begin{aligned}
F_x = &\quad \overline{U\tau - \frac{1}{6}\delta_{ii}\tau}^{i} \\
F_y = &\quad \overline{V\tau - \frac{1}{6}\delta_{ii}\tau}^{j} \\
F_r = &\quad \overline{W\tau - \frac{1}{6}\delta_{ii}\tau}^{k}
\end{aligned}
$$

As for the third order scheme, the best discretization near boundaries is under investigation but currently $\delta_i \tau = 0$ on a boundary.

S/R GAD_C4_ADV_X

F_x : uT (argument)
U : uTrans (argument)
τ : tracer (argument)

S/R GAD_C4_ADV_Y

F_y : vT (argument)
V : vTrans (argument)
τ : tracer (argument)

S/R GAD_C4_ADV_R

F_r : wT (argument)
W : rTrans (argument)
τ : tracer (argument)

2.17.1.4 First order upwind advection

Although the upwind scheme is the underlying scheme for the robust or non-linear methods given in Section 2.17.2, we haven't actually implemented this method for general use. It would be very diffusive and it is unlikely that it could ever produce more useful results than the positive higher order schemes.

Upwind bias is introduced into many schemes using the *abs* function and it allows the first order upwind flux to be written:

$$\begin{aligned} F_x = &\quad U\overline{\tau}^i - \frac{1}{2}|U|\delta_i\tau \\ F_y = &\quad V\overline{\tau}^j - \frac{1}{2}|V|\delta_j\tau \\ F_r = &\quad W\overline{\tau}^k - \frac{1}{2}|W|\delta_k\tau \end{aligned}$$

If for some reason the above method is desired, the second order flux limiter scheme described in Section 2.17.2.1 reduces to the above scheme if the limiter is set to zero.

2.17.2 Non-linear advection schemes

Non-linear advection schemes invoke non-linear interpolation and are widely used in computational fluid dynamics (non-linear does not refer to the non-linearity of the advection operator). The flux limited advection schemes belong to the class of finite volume methods which neatly ties into the spatial discretization of the model.

When employing the flux limited schemes, first order upwind or direct-space-time method, the time-stepping is switched to forward in time.

2.17.2.1 Second order flux limiters

The second order flux limiter method can be cast in several ways but is generally expressed in terms of other flux approximations. For example, in terms of a first order upwind flux and second order Lax-Wendroff flux, the limited flux is given as:

$$F = F_1 + \psi(r)F_{LW} \tag{2.147}$$

where $\psi(r)$ is the limiter function,

$$F_1 = u\overline{\tau}^i - \frac{1}{2}|u|\delta_i\tau$$

is the upwind flux,

$$F_{LW} = F_1 + \frac{|u|}{2}(1-c)\delta_i\tau$$

is the Lax-Wendroff flux and $c = \frac{u\Delta t}{\Delta x}$ is the Courant (CFL) number.

The limiter function, $\psi(r)$, takes the slope ratio

$$\begin{aligned} r = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}}\forall \quad u > 0 \\ r = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}}\forall \quad u < 0 \end{aligned}$$

as its argument. There are many choices of limiter function but we only provide the Superbee limiter (Roe 1995 [Roe85]):

$$\psi(r) = \max[0, \min[1, 2r], \min[2, r]]$$

S/R GAD_FLUXLIMIT_ADV_X

F_x : uT (argument)
U : uTrans (argument)
τ : tracer (argument)

S/R GAD_FLUXLIMIT_ADV_Y

F_y : vT (argument)
V : vTrans (argument)
τ : tracer (argument)

S/R GAD_FLUXLIMIT_ADV_R

F_r : wT (argument)
W : rTrans (argument)
τ : tracer (argument)

2.17.2.2 Third order direct space-time

The direct space-time method deals with space and time discretization together (other methods that treat space and time separately are known collectively as the "Method of Lines"). The Lax-Wendroff scheme falls into this category; it adds sufficient diffusion to a second order flux that the forward-in-time method is stable. The upwind biased third order DST scheme is:

$$\begin{aligned} F &= u\left(\tau_{i-1} + d_0(\tau_i - \tau_{i-1}) + d_1(\tau_{i-1} - \tau_{i-2})\right) \quad \forall \quad u > 0 \\ F &= u\left(\tau_i - d_0(\tau_i - \tau_{i-1}) - d_1(\tau_{i+1} - \tau_i)\right) \quad \forall \quad u < 0 \end{aligned} \tag{2.148}$$

where

$$\begin{aligned} d_0 &= \frac{1}{6}(2 - |c|)(1 - |c|) \\ d_1 &= \frac{1}{6}(1 - |c|)(1 + |c|) \end{aligned}$$

The coefficients d_0 and d_1 approach $1/3$ and $1/6$ respectively as the Courant number, c, vanishes. In this limit, the conventional third order upwind method is recovered. For finite Courant number, the deviations from the linear method are analogous to the diffusion added to centered second order advection in the Lax-Wendroff scheme.

The DST3 method described above must be used in a forward-in-time manner and is stable for $0 \leq |c| \leq 1$. Although the scheme appears to be forward-in-time, it is in fact third order in time and the accuracy increases with the Courant number! For low Courant number, DST3 produces very similar results (indistinguishable in Figure 2.10) to the linear third order method but for large Courant number, where the linear upwind third order method is unstable, the scheme is extremely accurate (Figure 2.11) with only minor overshoots.

S/R GAD_DST3_ADV_X

F_x : uT (argument)
U : uTrans (argument)
τ : tracer (argument)

S/R GAD_DST3_ADV_Y

F_y : vT (argument)
V : vTrans (argument)
τ : tracer (argument)

S/R GAD_DST3_ADV_R

F_r : wT (argument)
W : rTrans (argument)
τ : tracer (argument)

2.17.2.3 Third order direct space-time with flux limiting

The overshoots in the DST3 method can be controlled with a flux limiter. The limited flux is written:

$$F = \frac{1}{2}(u + |u|)\left(\tau_{i-1} + \psi(r^+)(\tau_i - \tau_{i-1})\right) + \frac{1}{2}(u - |u|)\left(\tau_{i-1} + \psi(r^-)(\tau_i - \tau_{i-1})\right) \tag{2.149}$$

where

$$r^+ = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}}$$
$$r^- = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}}$$

and the limiter is the Sweby limiter:

$$\psi(r) = \max[0, \min[\min(1, d_0 + d_1 r], \frac{1-c}{c} r]]$$

S/R GAD_DST3FL_ADV_X

F_x : uT (argument)
U : uTrans (argument)
τ : tracer (argument)

S/R GAD_DST3FL_ADV_Y

F_y : vT (argument)
V : vTrans (argument)
τ : tracer (argument)

S/R GAD_DST3FL_ADV_R

F_r : wT (argument)
W : rTrans (argument)
τ : tracer (argument)

2.17.2.4 Multi-dimensional advection

In many of the aforementioned advection schemes the behavior in multiple dimensions is not necessarily as good as the one dimensional behavior. For instance, a shape preserving monotonic scheme in one dimension can have severe shape distortion in two dimensions if the two components of horizontal fluxes are treated independently. There is a large body of literature on the subject dealing with this problem and among the fixes are operator and flux splitting methods, corner flux methods, and more. We have adopted a variant on the standard splitting methods that allows the flux calculations to be implemented as if in one dimension:

$$\begin{aligned}
\tau^{n+1/3} &= \tau^n - \Delta t \left(\frac{1}{\Delta x} \delta_i F^x(\tau^n) - \tau^n \frac{1}{\Delta x} \delta_i u \right) \\
\tau^{n+2/3} &= \tau^{n+1/3} - \Delta t \left(\frac{1}{\Delta y} \delta_j F^y(\tau^{n+1/3}) - \tau^n \frac{1}{\Delta y} \delta_i v \right) \\
\tau^{n+3/3} &= \tau^{n+2/3} - \Delta t \left(\frac{1}{\Delta r} \delta_k F^x(\tau^{n+2/3}) - \tau^n \frac{1}{\Delta r} \delta_i w \right)
\end{aligned} \tag{2.150}$$

In order to incorporate this method into the general model algorithm, we compute the effective tendency rather than update the tracer so that other terms such as diffusion are using the n time-level and not the updated $n+3/3$ quantities:

$$G_{adv}^{n+1/2} = \frac{1}{\Delta t}(\tau^{n+3/3} - \tau^n)$$

So that the over all time-stepping looks likes:

$$\tau^{n+1} = \tau^n + \Delta t \left(G_{adv}^{n+1/2} + G_{diff}(\tau^n) + G_{forcing}^n \right)$$

S/R GAD_ADVECTION

τ : tracer (argument)
$G_{adv}^{n+1/2}$: gTracer (argument)
F_x, F_y, F_r : aF (local)
U : uTrans (local)
V : vTrans (local)
W : rTrans (local)

A schematic of multi-dimension time stepping for the cube sphere configuration is show in Figure 2.9 .

2.17.3 Comparison of advection schemes

Table 2.2 shows a summary of the different advection schemes available in MITgcm. "AB" stands for Adams-Bashforth and "DST" for direct space-time. The code corresponds to the number used to select the corresponding advection scheme in the parameter file (e.g., `tempAdvScheme=3` in file `data` selects the 3rd order upwind advection scheme for temperature advection).

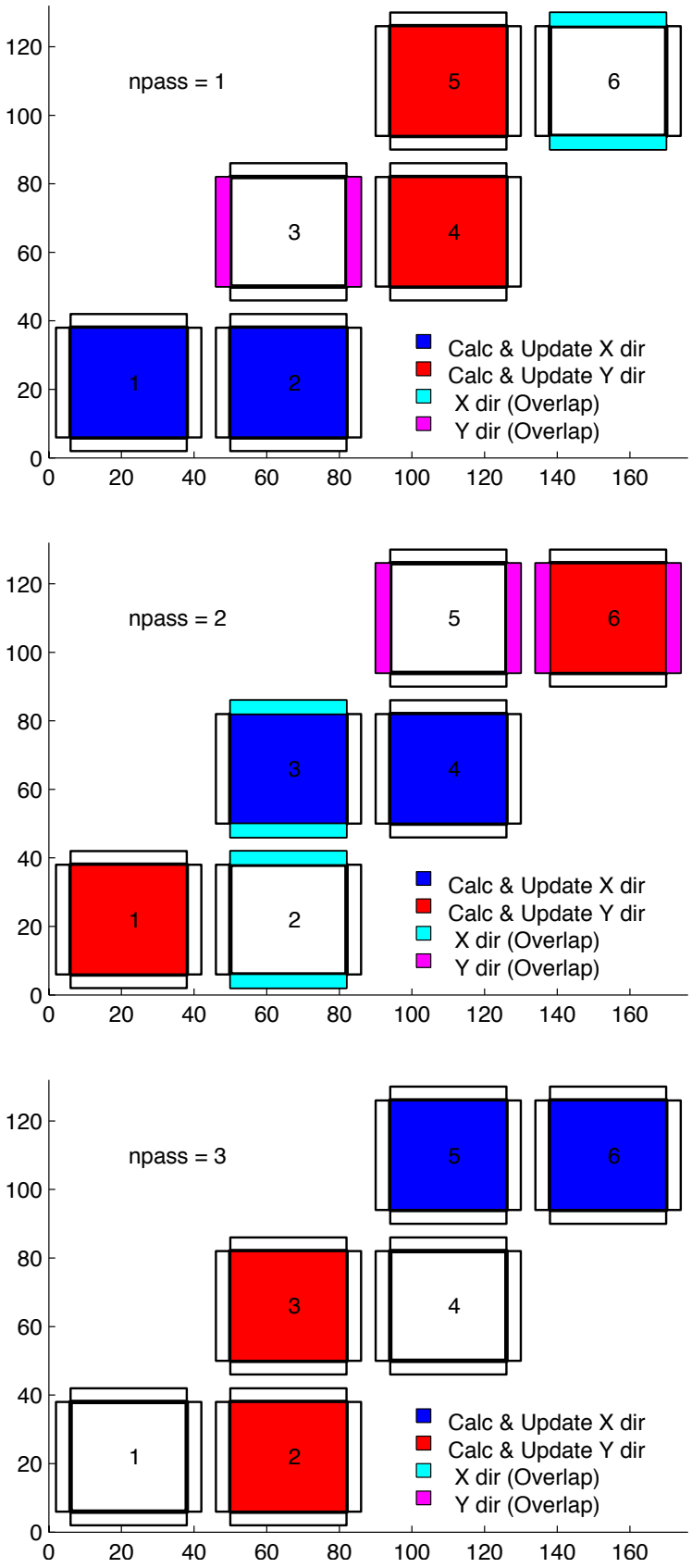

Figure 2.9: Multi-dimensional advection time-stepping with cubed-sphere topology.

Table 2.2: MITgcm Advection Schemes

Advection Scheme	Code	Use AB?	Use multi-dim?	Stencil (1-D)	Comments
1st order upwind	1	no	yes*	3	linear τ, non-linear $\vec{v}$
centered 2nd order	2	yes	no	3	linear
3rd order upwind	3	yes	no	5	linear τ
centered 4th order	4	yes	no	5	linear
2nd order DST (Lax-Wendroff)	20	no	yes*	3	linear τ, non-linear $\vec{v}$
3rd order DST	30	no	yes*	5	linear τ, non-linear $\vec{v}$
2nd order flux limiters	77	no	yes*	5	non-linear
3rd order DST flux limiter	33	no	yes*	5	non-linear
piecewise parabolic w/"null" limiter	40	no	yes	7	non-linear
piecewise parabolic w/"mono" limiter	41	no	yes	7	non-linear
piecewise parabolic w/"weno" limiter	42	no	yes	7	non-linear
piecewise quartic w/"null" limiter	50	no	yes	9	non-linear
piecewise quartic w/"mono" limiter	51	no	yes	9	non-linear
piecewise quartic w/"weno" limiter	52	no	yes	9	non-linear
7th order one-step method w/monotonicity preserving limiter	7	no	yes	9	non-linear
second order-moment Prather	80	no	yes	3	non-linear
second order-moment Prather w/limiter	81	no	yes	3	non-linear

yes* indicates that either the multi-dim advection algorithm or standard approach can be utilized, controlled by a namelist parameter multiDimAdvection (in these cases, given that these schemes was designed to use multi-dim advection, using the standard approach is not recommended). The minimum size of the required tile overlap region (OLx, OLx) is (stencil size -1)/2. The minimum overlap required by the model in general is 2, so for some of the above choices the advection scheme will not cost anything in terms of an additional overlap requirement, but especially given a small tile size, using scheme 7 for example would require costly additional overlap points (note a cube sphere grid with a "wet-corner" requires doubling this overlap!) In the 'Comments' column, τ refers to tracer advection, $\vec{v}$ momentum advection.

Shown in Figure 2.10 and Figure 2.11 is a 1-D comparison of advection schemes. Here we advect both a smooth hill and a hill with a more abrupt shock. Figure 2.10 shown the result for a weak flow (low Courant number) whereas Figure 2.11 shows the result for a stronger flow (high Courant number).

Figure 2.12, Figure 2.13 and Figure 2.14 show solutions to a simple diagonal advection problem using a selection of schemes for low, moderate and high Courant numbers, respectively. The top row shows the linear schemes, integrated with the Adams-Bashforth method. Theses schemes are clearly unstable for the high Courant number and weakly unstable for the moderate Courant number. The presence of false extrema is very apparent for all Courant numbers. The middle row shows solutions obtained with the unlimited but multi-dimensional schemes. These solutions also exhibit false extrema though the pattern now shows symmetry due to the multi-dimensional scheme. Also, the schemes are stable at high Courant number where the linear schemes weren't. The bottom row (left and middle) shows the limited schemes and most obvious is the absence of false extrema. The accuracy and stability of the unlimited non-linear schemes is retained at high Courant number but at low Courant number the tendency is to lose amplitude in sharp peaks due to diffusion. The one dimensional tests shown in Figure 2.10 and Figure 2.11 show this phenomenon.

Finally, the bottom left and right panels use the same advection scheme but the right does not use the multi-dimensional method. At low Courant number this appears to not matter but for moderate Courant number severe distortion of the feature is apparent. Moreover, the stability of the multi-dimensional scheme is determined by the maximum Courant

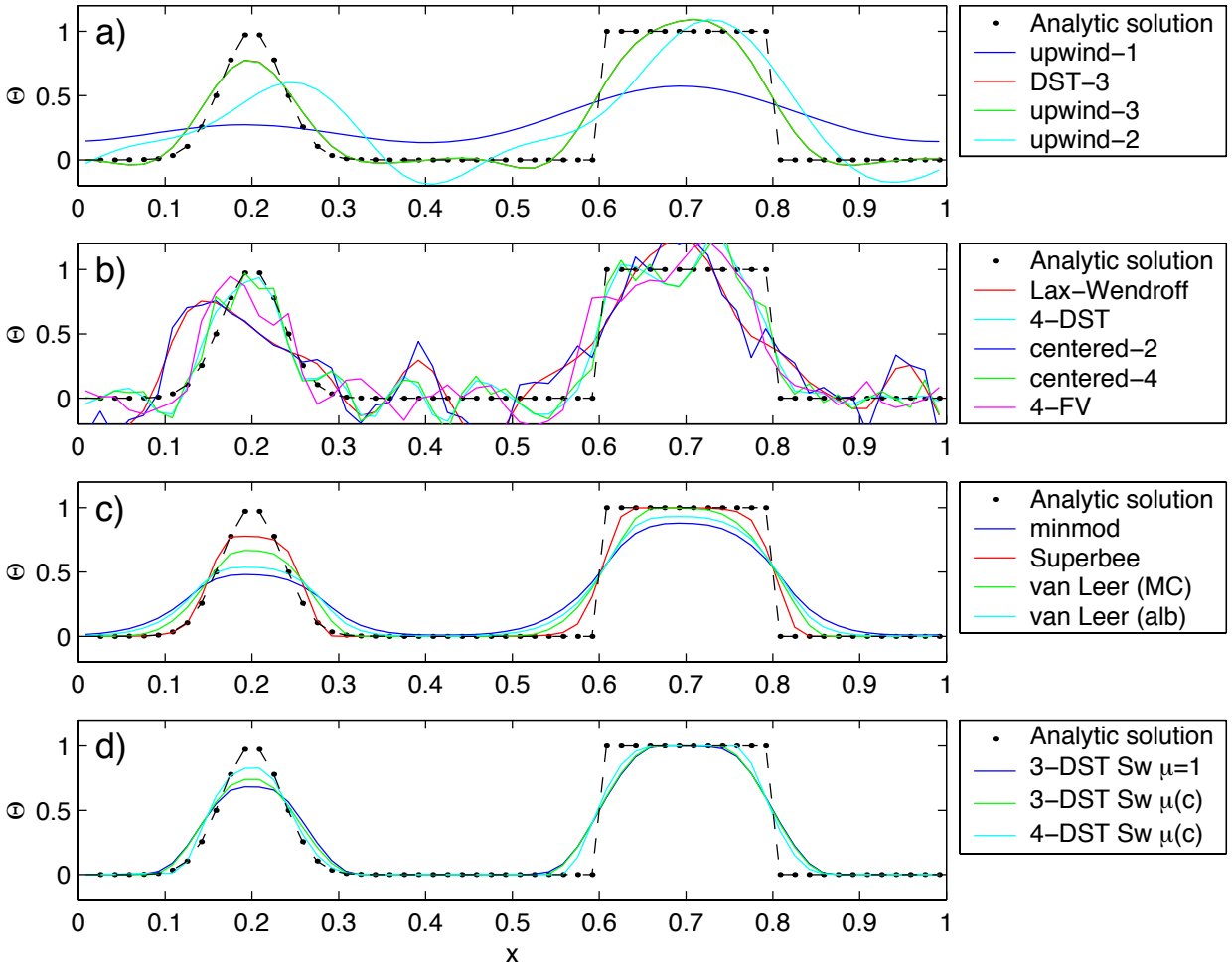

Figure 2.10: Comparison of 1-D advection schemes: Courant number is 0.05 with 60 points and solutions are shown for T=1 (one complete period). a) Shows the upwind biased schemes; first order upwind, DST3, third order upwind and second order upwind. b) Shows the centered schemes; Lax-Wendroff, DST4, centered second order, centered fourth order and finite volume fourth order. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1$, $\mu = c/(1-c)$ and a fourth order DST method with Sweby limiter, $\mu = c/(1-c)$.

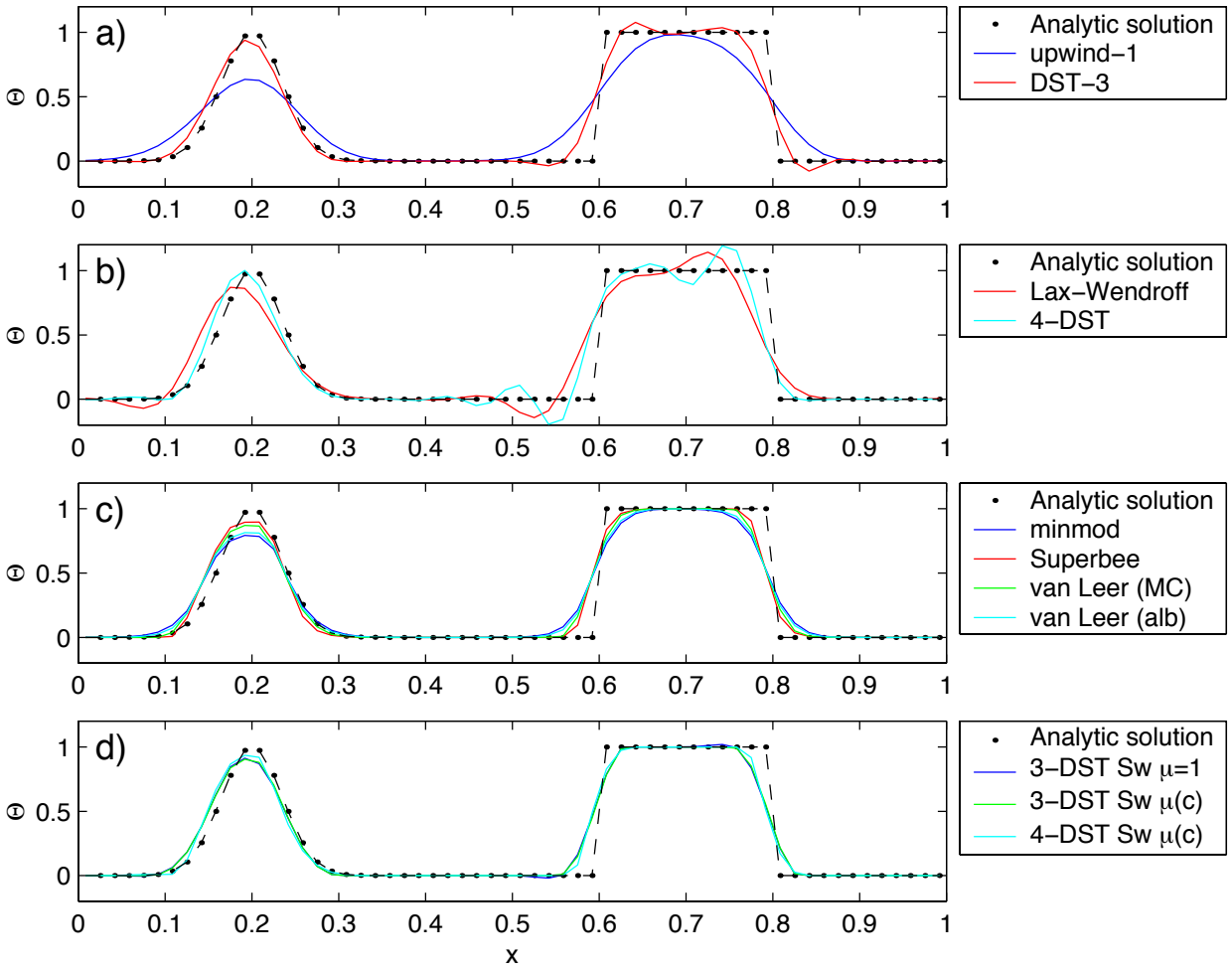

Figure 2.11: Comparison of 1-D advection schemes: Courant number is 0.89 with 60 points and solutions are shown for T=1 (one complete period). a) Shows the upwind biased schemes; first order upwind and DST3. Third order upwind and second order upwind are unstable at this Courant number. b) Shows the centered schemes; Lax-Wendroff, DST4. Centered second order, centered fourth order and finite volume fourth order are unstable at this Courant number. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1$, $\mu = c/(1-c)$ and a fourth order DST method with Sweby limiter, $\mu = c/(1-c)$.

number applied of each dimension while the stability of the method of lines is determined by the sum. Hence, in the high Courant number plot, the scheme is unstable.

With many advection schemes implemented in the code two questions arise: "Which scheme is best?" and "Why don't you just offer the best advection scheme?". Unfortunately, no one advection scheme is "the best" for all particular applications and for new applications it is often a matter of trial to determine which is most suitable. Here are some guidelines but these are not the rule;

- If you have a coarsely resolved model, using a positive or upwind biased scheme will introduce significant diffusion to the solution and using a centered higher order scheme will introduce more noise. In this case, simplest may be best.
- If you have a high resolution model, using a higher order scheme will give a more accurate solution but scale-selective diffusion might need to be employed. The flux limited methods offer similar accuracy in this regime.
- If your solution has shocks or propagating fronts then a flux limited scheme is almost essential.
- If your time-step is limited by advection, the multi-dimensional non-linear schemes have the most stability (up to Courant number 1).
- If you need to know how much diffusion/dissipation has occurred you will have a lot of trouble figuring it out with a non-linear method.
- The presence of false extrema is non-physical and this alone is the strongest argument for using a positive scheme.

2.18 Shapiro Filter

The Shapiro filter (Shapiro 1970) [Sha70] is a high order horizontal filter that efficiently remove small scale grid noise without affecting the physical structures of a field. It is applied at the end of the time step on both velocity and tracer fields.

Three different space operators are considered here (S1,S2 and S4). They differ essentially by the sequence of derivative in both X and Y directions. Consequently they show different damping response function specially in the diagonal directions X+Y and X-Y.

Space derivatives can be computed in the real space, taking into account the grid spacing. Alternatively, a pure computational filter can be defined, using pure numerical differences and ignoring grid spacing. This later form is stable whatever the grid is, and therefore specially useful for highly anisotropic grid such as spherical coordinate grid. A damping time-scale parameter τ_{shap} defines the strength of the filter damping.

The three computational filter operators are :

$$\text{S1c}: \quad [1 - 1/2\frac{\Delta t}{\tau_{shap}}\{(\frac{1}{4}\delta_{ii})^n + (\frac{1}{4}\delta_{jj})^n\}]$$

$$\text{S2c}: \quad [1 - \frac{\Delta t}{\tau_{shap}}\{\frac{1}{8}(\delta_{ii} + \delta_{jj})\}^n]$$

$$\text{S4c}: \quad [1 - \frac{\Delta t}{\tau_{shap}}(\frac{1}{4}\delta_{ii})^n][1 - \frac{\Delta t}{\tau_{shap}}(\frac{1}{4}\delta_{jj})^n]$$

In addition, the S2 operator can easily be extended to a physical space filter:

$$\text{S2g}: \quad [1 - \frac{\Delta t}{\tau_{shap}}\{\frac{L_{shap}^2}{8}\overline{\nabla}^2\}^n]$$

with the Laplacian operator $\overline{\nabla}^2$ and a length scale parameter L_{shap}. The stability of this S2g filter requires $L_{shap} < \text{Min}^{(Global)}(\Delta x, \Delta y)$.

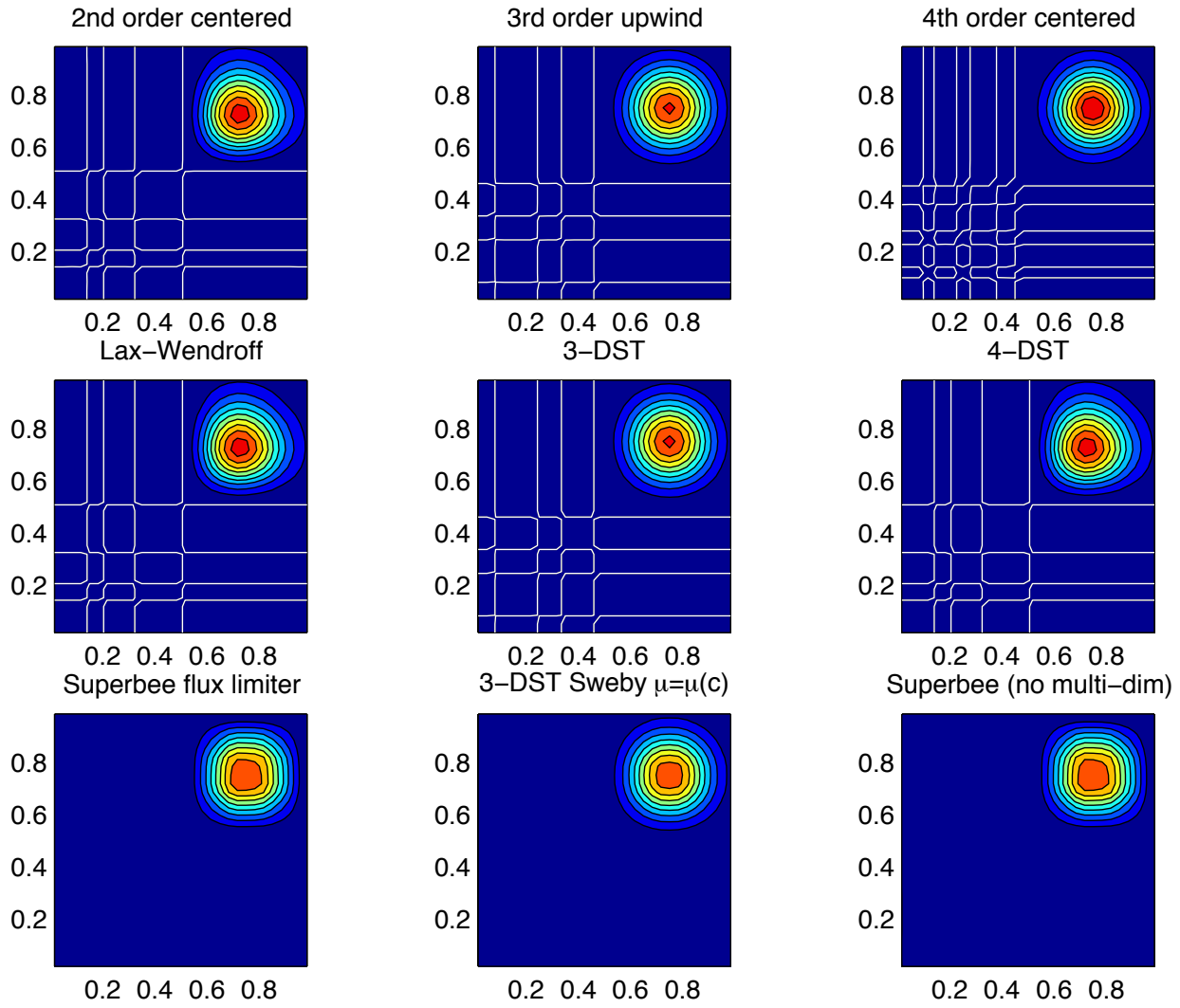

Figure 2.12: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.01 with 30 $\times$ 30 points and solutions are shown for T=1/2. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

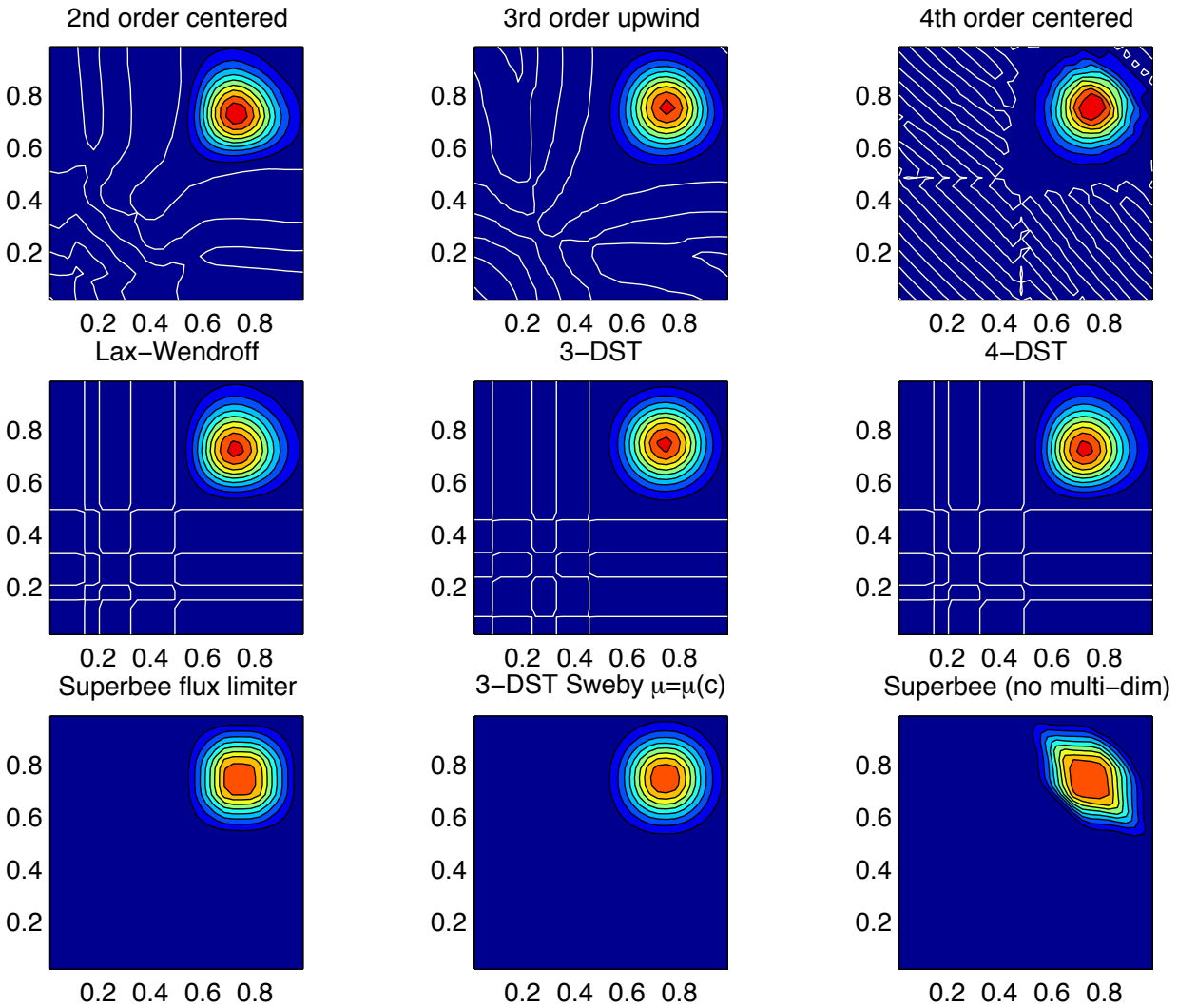

Figure 2.13: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.27 with 30 $\times$ 30 points and solutions are shown for T=1/2. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

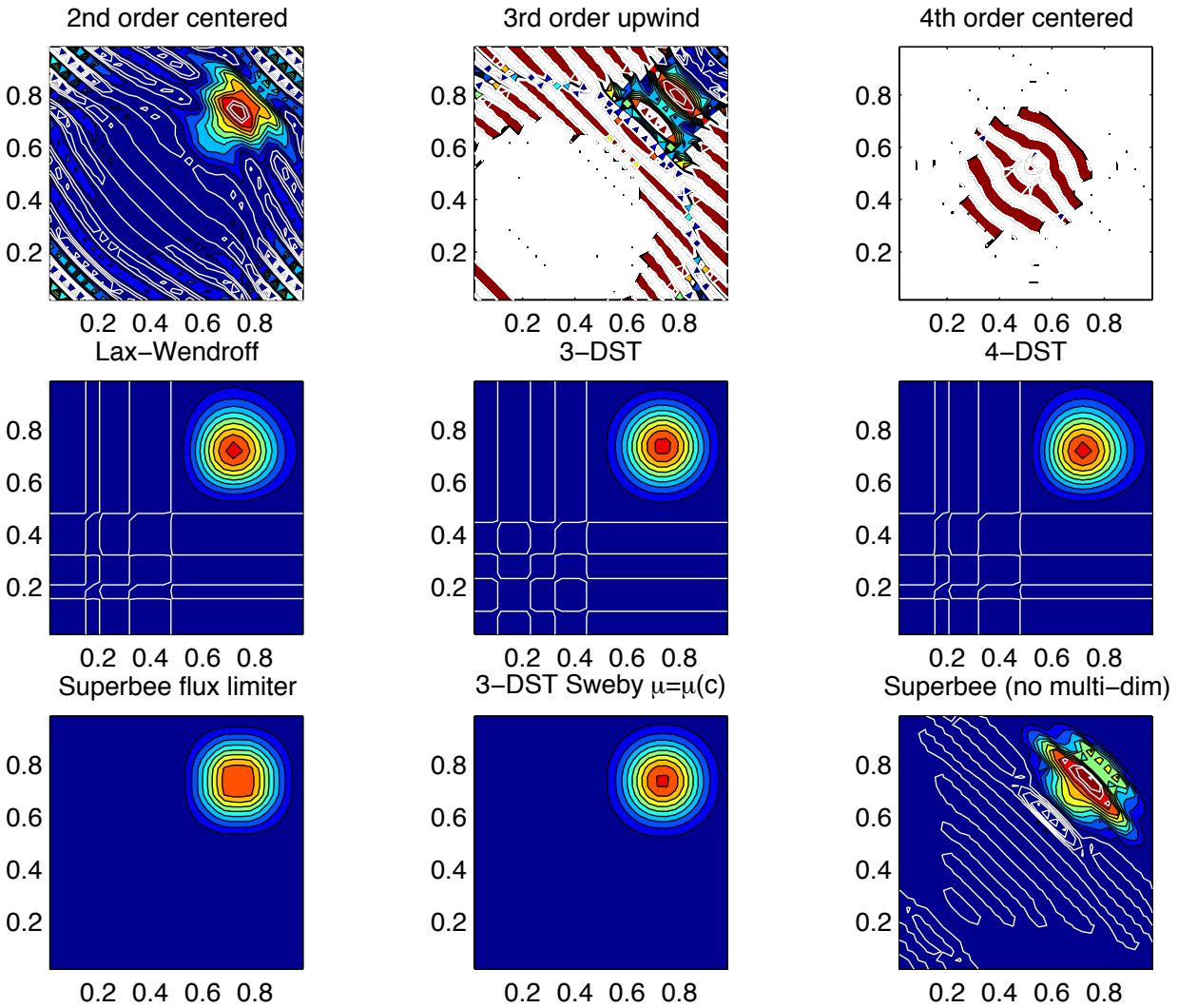

Figure 2.14: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.47 with 30 $\times$ 30 points and solutions are shown for T=1/2. White lines indicate zero crossings and initial maximum values (ie. the presence of false extrema). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

2.18.1 SHAP Diagnostics

```
------------------------------------------------------------
<-Name->|Levs|parsing code|<-Units->|<- Tile (max=80c)
------------------------------------------------------------
SHAP_dT |   5 |SM      MR  |K/s      |Temperature Tendency due to Shapiro Filter
SHAP_dS |   5 |SM      MR  |g/kg/s   |Specific Humidity Tendency due to Shapiro Filter
SHAP_dU |   5 |UU   148MR  |m/s^2    |Zonal Wind Tendency due to Shapiro Filter
SHAP_dV |   5 |VV   147MR  |m/s^2    |Meridional Wind Tendency due to Shapiro Filter
```

2.19 Nonlinear Viscosities for Large Eddy Simulation

In Large Eddy Simulations (LES), a turbulent closure needs to be provided that accounts for the effects of subgridscale motions on the large scale. With sufficiently powerful computers, we could resolve the entire flow down to the molecular viscosity scales ($L_\nu \approx 1\mathrm{cm}$). Current computation allows perhaps four decades to be resolved, so the largest problem computationally feasible would be about 10m. Most oceanographic problems are much larger in scale, so some form of LES is required, where only the largest scales of motion are resolved, and the subgridscale effects on the large-scale are parameterized.

To formalize this process, we can introduce a filter over the subgridscale L: $u_\alpha \rightarrow \overline{u_\alpha}$ and L: $b \rightarrow \bar{b}$. This filter has some intrinsic length and time scales, and we assume that the flow at that scale can be characterized with a single velocity scale (V) and vertical buoyancy gradient (N^2). The filtered equations of motion in a local Mercator projection about the gridpoint in question (see Appendix for notation and details of approximation) are:

$$\frac{\overline{D\tilde{u}}}{\overline{Dt}} - \frac{\tilde{\bar{v}}\sin\theta}{\mathrm{Ro}\sin\theta_0} + \frac{M_{Ro}}{\mathrm{Ro}}\frac{\partial\overline{\pi}}{\partial x} = -\left(\frac{\overline{D\tilde{u}}}{Dt} - \frac{\overline{D\tilde{u}}}{\overline{Dt}}\right) + \frac{\nabla^2\tilde{\bar{u}}}{\mathrm{Re}} \tag{2.151}$$

$$\frac{\overline{D\tilde{v}}}{\overline{Dt}} - \frac{\tilde{\bar{u}}\sin\theta}{\mathrm{Ro}\sin\theta_0} + \frac{M_{Ro}}{\mathrm{Ro}}\frac{\partial\overline{\pi}}{\partial y} = -\left(\frac{\overline{D\tilde{v}}}{Dt} - \frac{\overline{D\tilde{v}}}{\overline{Dt}}\right) + \frac{\nabla^2\tilde{\bar{v}}}{\mathrm{Re}} \tag{2.152}$$

$$\frac{\overline{Dw}}{\overline{Dt}} + \frac{\frac{\partial\overline{\pi}}{\partial z} - \bar{b}}{\mathrm{Fr}^2\lambda^2} = -\left(\frac{\overline{Dw}}{Dt} - \frac{\overline{Dw}}{\overline{Dt}}\right) + \frac{\nabla^2\overline{w}}{\mathrm{Re}}$$

$$\frac{\overline{D\bar{b}}}{\overline{Dt}} + \overline{w} = -\left(\frac{\overline{Db}}{Dt} - \frac{\overline{D\bar{b}}}{\overline{Dt}}\right) + \frac{\nabla^2\bar{b}}{\mathrm{Pr}\,\mathrm{Re}}$$

$$\mu^2\left(\frac{\partial\tilde{\bar{u}}}{\partial x} + \frac{\partial\tilde{\bar{v}}}{\partial y}\right) + \frac{\partial\overline{w}}{\partial z} = 0 \tag{2.153}$$

Tildes denote multiplication by $\cos\theta/\cos\theta_0$ to account for converging meridians.

The ocean is usually turbulent, and an operational definition of turbulence is that the terms in parentheses (the 'eddy' terms) on the right of (2.151) - (2.153)) are of comparable magnitude to the terms on the left-hand side. The terms proportional to the inverse of , instead, are many orders of magnitude smaller than all of the other terms in virtually every oceanic application.

2.19.1 Eddy Viscosity

A turbulent closure provides an approximation to the 'eddy' terms on the right of the preceding equations. The simplest form of LES is just to increase the viscosity and diffusivity until the viscous and diffusive scales are resolved. That is, we approximate (2.151) - (2.153):

$$\left(\frac{\overline{D\tilde{u}}}{Dt} - \frac{\overline{D\tilde{u}}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2\tilde{\bar{u}}}{\mathrm{Re}_h} + \frac{\frac{\partial^2\tilde{\bar{u}}}{\partial z^2}}{\mathrm{Re}_v} \tag{2.154}$$

$$\left(\overline{\frac{D\tilde{v}}{Dt}} - \frac{\overline{D\tilde{v}}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2 \tilde{\overline{v}}}{\mathrm{Re}_h} + \frac{\frac{\partial^2 \tilde{\overline{v}}}{\partial z^2}}{\mathrm{Re}_v} \tag{2.155}$$

$$\left(\overline{\frac{Dw}{Dt}} - \frac{\overline{Dw}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2 \overline{w}}{\mathrm{Re}_h} + \frac{\frac{\partial^2 \overline{w}}{\partial z^2}}{\mathrm{Re}_v} \tag{2.156}$$

$$\left(\overline{\frac{Db}{Dt}} - \frac{\overline{Db}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2 \overline{b}}{\mathrm{Pr}\,\mathrm{Re}_h} + \frac{\frac{\partial^2 \overline{b}}{\partial z^2}}{\mathrm{Pr}\,\mathrm{Re}_v}$$

2.19.1.1 Reynolds-Number Limited Eddy Viscosity

One way of ensuring that the gridscale is sufficiently viscous (i.e., resolved) is to choose the eddy viscosity A_h so that the gridscale horizontal Reynolds number based on this eddy viscosity, Re_h, is O(1). That is, if the gridscale is to be viscous, then the viscosity should be chosen to make the viscous terms as large as the advective ones. Bryan et al. (1975) [BMP75] notes that a computational mode is squelched by using Re_h <2.

MITgcm users can select horizontal eddy viscosities based on Re_h using two methods. 1) The user may estimate the velocity scale expected from the calculation and grid spacing and set viscAh to satisfy $\mathrm{Re}_h < 2$. 2) The user may use viscAhReMax, which ensures that the viscosity is always chosen so that $\mathrm{Re}_h <$ viscAhReMax. This last option should be used with caution, however, since it effectively implies that viscous terms are fixed in magnitude relative to advective terms. While it may be a useful method for specifying a minimum viscosity with little effort, tests Bryan et al. (1975) [BMP75] have shown that setting viscAhReMax =2 often tends to increase the viscosity substantially over other more 'physical' parameterizations below, especially in regions where gradients of velocity are small (and thus turbulence may be weak), so perhaps a more liberal value should be used, e.g. viscAhReMax =10.

While it is certainly necessary that viscosity be active at the gridscale, the wavelength where dissipation of energy or enstrophy occurs is not necessarily $L = A_h/U$. In fact, it is by ensuring that either the dissipation of energy in a 3-d turbulent cascade (Smagorinsky) or dissipation of enstrophy in a 2-d turbulent cascade (Leith) is resolved that these parameterizations derive their physical meaning.

2.19.1.2 Vertical Eddy Viscosities

Vertical eddy viscosities are often chosen in a more subjective way, as model stability is not usually as sensitive to vertical viscosity. Usually the 'observed' value from finescale measurements is used (e.g. viscAr$\approx 1 \times 10^{-4} m^2/s$). However, Smagorinsky (1993) [Sma93] notes that the Smagorinsky parameterization of isotropic turbulence implies a value of the vertical viscosity as well as the horizontal viscosity (see below).

2.19.1.3 Smagorinsky Viscosity

Some suggest (see Smagorinsky 1963 [Sma63]; Smagorinsky 1993 [Sma93]) choosing a viscosity that *depends on the resolved motions*. Thus, the overall viscous operator has a nonlinear dependence on velocity. Smagorinsky chose his form of viscosity by considering Kolmogorov's ideas about the energy spectrum of 3-d isotropic turbulence.

Kolmogorov supposed that energy is injected into the flow at large scales (small k) and is 'cascaded' or transferred conservatively by nonlinear processes to smaller and smaller scales until it is dissipated near the viscous scale. By setting the energy flux through a particular wavenumber k, ϵ, to be a constant in k, there is only one combination of viscosity and energy flux that has the units of length, the Kolmogorov wavelength. It is $L_\epsilon(\nu) \propto \pi\epsilon^{-1/4}\nu^{3/4}$ (the π stems from conversion from wavenumber to wavelength). To ensure that this viscous scale is resolved in a numerical model, the gridscale should be decreased until $L_\epsilon(\nu) > L$ (so-called Direct Numerical Simulation, or DNS). Alternatively, an eddy viscosity can be used and the corresponding Kolmogorov length can be made larger than the gridscale, $L_\epsilon(A_h) \propto \pi\epsilon^{-1/4}A_h^{3/4}$ (for Large Eddy Simulation or LES).

There are two methods of ensuring that the Kolmogorov length is resolved in MITgcm. 1) The user can estimate the flux of energy through spectral space for a given simulation and adjust grid spacing or viscAh to ensure that

$L_\epsilon(A_h) > L$; 2) The user may use the approach of Smagorinsky with viscC2Smag, which estimates the energy flux at every grid point, and adjusts the viscosity accordingly.

Smagorinsky formed the energy equation from the momentum equations by dotting them with velocity. There are some complications when using the hydrostatic approximation as described by Smagorinsky (1993) [Sma93]. The positive definite energy dissipation by horizontal viscosity in a hydrostatic flow is νD^2, where D is the deformation rate at the viscous scale. According to Kolmogorov's theory, this should be a good approximation to the energy flux at any wavenumber $\epsilon \approx \nu D^2$. Kolmogorov and Smagorinsky noted that using an eddy viscosity that exceeds the molecular value ν should ensure that the energy flux through viscous scale set by the eddy viscosity is the same as it would have been had we resolved all the way to the true viscous scale. That is, $\epsilon \approx A_{hSmag}\overline{D}^2$. If we use this approximation to estimate the Kolmogorov viscous length, then

$$L_\epsilon(A_{hSmag}) \propto \pi\epsilon^{-1/4}A_{hSmag}^{3/4} \approx \pi(A_{hSmag}\overline{D}^2)^{-1/4}A_{hSmag}^{3/4} = \pi A_{hSmag}^{1/2}\overline{D}^{-1/2} \tag{2.157}$$

To make $L_\epsilon(A_{hSmag})$ scale with the gridscale, then

$$A_{hSmag} = \left(\frac{\mathsf{viscC2Smag}}{\pi}\right)^2 L^2|\overline{D}| \tag{2.158}$$

Where the deformation rate appropriate for hydrostatic flows with shallow-water scaling is

$$|\overline{D}| = \sqrt{\left(\frac{\partial\overline{\bar{u}}}{\partial x} - \frac{\partial\overline{\bar{v}}}{\partial y}\right)^2 + \left(\frac{\partial\overline{\bar{u}}}{\partial y} + \frac{\partial\overline{\bar{v}}}{\partial x}\right)^2} \tag{2.159}$$

The coefficient viscC2Smag is what an MITgcm user sets, and it replaces the proportionality in the Kolmogorov length with an equality. Others (Griffies and Hallberg, 2000 [GH00]) suggest values of viscC2Smag from 2.2 to 4 for oceanic problems. Smagorinsky (1993) [Sma93] shows that values from 0.2 to 0.9 have been used in atmospheric modeling.

Smagorinsky (1993) [Sma93] shows that a corresponding vertical viscosity should be used:

$$A_{vSmag} = \left(\frac{\mathsf{viscC2Smag}}{\pi}\right)^2 H^2\sqrt{\left(\frac{\partial\overline{\bar{u}}}{\partial z}\right)^2 + \left(\frac{\partial\overline{\bar{v}}}{\partial z}\right)^2} \tag{2.160}$$

This vertical viscosity is currently not implemented in MITgcm.

2.19.1.4 Leith Viscosity

Leith (1968, 1996) [Lei68] [Lei96] notes that 2-d turbulence is quite different from 3-d. In two-dimensional turbulence, energy cascades to larger scales, so there is no concern about resolving the scales of energy dissipation. Instead, another quantity, enstrophy, (which is the vertical component of vorticity squared) is conserved in 2-d turbulence, and it cascades to smaller scales where it is dissipated.

Following a similar argument to that above about energy flux, the enstrophy flux is estimated to be equal to the positive-definite gridscale dissipation rate of enstrophy $\eta \approx A_{hLeith}|\nabla\overline{\omega}_3|^2$. By dimensional analysis, the enstrophy-dissipation scale is $L_\eta(A_{hLeith}) \propto \pi A_{hLeith}^{1/2}\eta^{-1/6}$. Thus, the Leith-estimated length scale of enstrophy-dissipation and the resulting eddy viscosity are

$$L_\eta(A_{hLeith}) \propto \pi A_{hLeith}^{1/2}\eta^{-1/6} = \pi A_{hLeith}^{1/3}|\nabla\overline{\omega}_3|^{-1/3} \tag{2.161}$$

$$A_{hLeith} = \left(\frac{\mathsf{viscC2Leith}}{\pi}\right)^3 L^3|\nabla\overline{\omega}_3| \tag{2.162}$$

$$|\nabla\omega_3| \equiv \sqrt{\left[\frac{\partial}{\partial x}\left(\frac{\partial\overline{\bar{v}}}{\partial x} - \frac{\partial\overline{\bar{u}}}{\partial y}\right)\right]^2 + \left[\frac{\partial}{\partial y}\left(\frac{\partial\overline{\bar{v}}}{\partial x} - \frac{\partial\overline{\bar{u}}}{\partial y}\right)\right]^2} \tag{2.163}$$

The runtime flag useFullLeith controls whether or not to calculate the full gradients for the Leith viscosity (.TRUE.) or to use an approximation (.FALSE.). The only reason to set useFullLeith = .FALSE. is if your simulation fails when computing the gradients. This can occur when using the cubed sphere and other complex grids.

2.19.1.5 Modified Leith Viscosity

The argument above for the Leith viscosity parameterization uses concepts from purely 2-dimensional turbulence, where the horizontal flow field is assumed to be non-divergent. However, oceanic flows are only quasi-two dimensional. While the barotropic flow, or the flow within isopycnal layers may behave nearly as two-dimensional turbulence, there is a possibility that these flows will be divergent. In a high-resolution numerical model, these flows may be substantially divergent near the grid scale, and in fact, numerical instabilities exist which are only horizontally divergent and have little vertical vorticity. This causes a difficulty with the Leith viscosity, which can only respond to buildup of vorticity at the grid scale.

MITgcm offers two options for dealing with this problem. 1) The Smagorinsky viscosity can be used instead of Leith, or in conjunction with Leith – a purely divergent flow does cause an increase in Smagorinsky viscosity; 2) The viscC2LeithD parameter can be set. This is a damping specifically targeting purely divergent instabilities near the gridscale. The combined viscosity has the form:

$$A_{hLeith} = L^3 \sqrt{\left(\frac{\text{viscC2Leith}}{\pi}\right)^6 |\nabla\overline{\omega}_3|^2 + \left(\frac{\text{viscC2LeithD}}{\pi}\right)^6 |\nabla\nabla\cdot\overline{\tilde{u}}_h|^2} \tag{2.164}$$

$$|\nabla\nabla\cdot\overline{\tilde{u}}_h| \equiv \sqrt{\left[\frac{\partial}{\partial x}\left(\frac{\partial\overline{\tilde{u}}}{\partial x}+\frac{\partial\overline{\tilde{v}}}{\partial y}\right)\right]^2 + \left[\frac{\partial}{\partial y}\left(\frac{\partial\overline{\tilde{u}}}{\partial x}+\frac{\partial\overline{\tilde{v}}}{\partial y}\right)\right]^2} \tag{2.165}$$

Whether there is any physical rationale for this correction is unclear, but the numerical consequences are good. The divergence in flows with the grid scale larger or comparable to the Rossby radius is typically much smaller than the vorticity, so this adjustment only rarely adjusts the viscosity if viscC2LeithD = viscC2Leith. However, the rare regions where this viscosity acts are often the locations for the largest vales of vertical velocity in the domain. Since the CFL condition on vertical velocity is often what sets the maximum timestep, this viscosity may substantially increase the allowable timestep without severely compromising the verity of the simulation. Tests have shown that in some calculations, a timestep three times larger was allowed when viscC2LeithD = viscC2Leith.

2.19.1.6 Quasi-Geostrophic Leith Viscosity

A variant of Leith viscosity can be derived for quasi-geostrophic dynamics. This leads to a slightly different equation for the viscosity that includes a contribution from quasigeostrophic vortex stretching (Bachman et al. 2017 [BFKP17]). The viscosity is given by

$$\nu_* = \left(\frac{\Lambda\Delta s}{\pi}\right)^3 |\nabla_h(f\hat{\mathbf{z}}) + \nabla_h(\nabla\times\mathbf{v}_{h*}) + \partial_z\frac{f}{N^2}\nabla_h b| \tag{2.166}$$

where Λ is a tunable parameter of $\mathcal{O}(1)$, $\Delta s = \sqrt{\Delta x\Delta y}$ is the grid scale, $f\hat{\mathbf{z}}$ is the vertical component of the Coriolis parameter, $\mathbf{v}_{h*}$ is the horizontal velocity, N^2 is the Brunt-Väisälä frequency, and b is the buoyancy.

However, the viscosity given by (2.166) does not constrain purely divergent motions. As such, a small $\mathcal{O}(\epsilon)$ correction is added

$$\nu_* = \left(\frac{\Lambda\Delta s}{\pi}\right)^3 \sqrt{|\nabla_h(f\hat{\mathbf{z}}) + \nabla_h(\nabla\times\mathbf{v}_{h*}) + \partial_z\frac{f}{N^2}\nabla_h b|^2 + |\nabla[\nabla\cdot\mathbf{v}_h]|^2} \tag{2.167}$$

This form is, however, numerically awkward; as the Brunt-Väisälä Frequency becomes very small in regions of weak or vanishing stratification, the vortex stretching term becomes very large. The resulting large viscosities can lead to numerical instabilities. Bachman et al. (2017) [BFKP17] present two limiting forms for the viscosity based on flow parameters such as Fr_*, the Froude number, and Ro_*, the Rossby number. The second of which,

$$\nu_* = \left(\frac{\Lambda\Delta s}{\pi}\right)^3 \sqrt{min\left(|\nabla_h q_{2*} + \partial_z\frac{f^2}{N^2}\nabla_h b|, \left(1+\frac{Fr_*^2}{Ro_*^2}+Fr_*^4\right)|\nabla_h q_{2*}|\right)^2 + |\nabla[\nabla\cdot\mathbf{v}_h]|^2}, \tag{2.168}$$

has been implemented and is active when `#define` ALLOW_LEITH_QG is included in a copy of MOM_COMMON_OPTIONS.h in a code mods directory (specified through *-mods* command line option in genmake2).

LeithQG viscosity is designed to work best in simulations that resolve some mesoscale features. In simulations that are too coarse to permit eddies or fine enough to resolve submesoscale features, it should fail gracefully. The non-dimensional parameter viscC2LeithQG corresponds to Λ in the above equations and scales the viscosity; the recommended value is 1.

There is no reason to use the quasi-geostrophic form of Leith at the same time as either standard Leith or modified Leith. Therefore, the model will not run if non-zero values have been set for these coefficients; the model will stop during the configuration check. LeithQG can be used regardless of the setting for useFullLeith. Just as for the other forms of Leith viscosity, this flag determines whether or not the full gradients are used. The simplified gradients were originally intended for use on complex grids, but have been shown to produce better kinetic energy spectra even on very straightforward grids.

To add the LeithQG viscosity to the GMRedi coefficient, as was done in some of the simulations in Bachman et al. (2017) [BFKP17], `#define` ALLOW_LEITH_QG must be specified, as described above. In addition to this, the compile-time flag ALLOW_GM_LEITH_QG must also be defined in a (`-mods`) copy of GMREDI_OPTIONS.h when the model is compiled, and the runtime parameter GM_useLeithQG set to .TRUE. in `data.gmredi`. This will use the value of viscC2LeithQG specified in the `data` input file to compute the coefficient.

2.19.1.7 Courant–Freidrichs–Lewy Constraint on Viscosity

Whatever viscosities are used in the model, the choice is constrained by gridscale and timestep by the Courant–Freidrichs–Lewy (CFL) constraint on stability:

$$A_h < \frac{L^2}{4\Delta t}$$
$$A_4 \leq \frac{L^4}{32\Delta t}$$

The viscosities may be automatically limited to be no greater than these values in MITgcm by specifying viscAhGridMax < 1 and viscA4GridMax < 1. Similarly-scaled minimum values of viscosities are provided by viscAhGridMin and viscA4GridMin, which if used, should be set to values $\ll 1$. L is roughly the gridscale (see below).

Following Griffies and Hallberg (2000) [GH00], we note that there is a factor of $\Delta x^2/8$ difference between the harmonic and biharmonic viscosities. Thus, whenever a non-dimensional harmonic coefficient is used in the MITgcm (e.g. viscAhGridMax < 1), the biharmonic equivalent is scaled so that the same non-dimensional value can be used (e.g. viscA4GridMax < 1).

2.19.1.8 Biharmonic Viscosity

Holland (1978) [Hol78] suggested that eddy viscosities ought to be focused on the dynamics at the grid scale, as larger motions would be 'resolved'. To enhance the scale selectivity of the viscous operator, he suggested a biharmonic eddy viscosity instead of a harmonic (or Laplacian) viscosity:

$$\left(\overline{\frac{D\tilde{u}}{Dt}} - \frac{\overline{D\tilde{u}}}{\overline{Dt}}\right) \approx \frac{-\nabla_h^4 \tilde{\overline{u}}}{\mathrm{Re}_4} + \frac{\frac{\partial^2 \tilde{\overline{u}}}{\partial z^2}}{\mathrm{Re}_v} \tag{2.169}$$
$$\left(\overline{\frac{D\tilde{v}}{Dt}} - \frac{\overline{D\tilde{v}}}{\overline{Dt}}\right) \approx \frac{-\nabla_h^4 \tilde{\overline{v}}}{\mathrm{Re}_4} + \frac{\frac{\partial^2 \tilde{\overline{v}}}{\partial z^2}}{\mathrm{Re}_v}$$
$$\left(\overline{\frac{Dw}{Dt}} - \frac{\overline{Dw}}{\overline{Dt}}\right) \approx \frac{-\nabla_h^4 \overline{w}}{\mathrm{Re}_4} + \frac{\frac{\partial^2 \overline{w}}{\partial z^2}}{\mathrm{Re}_v}$$

$$\left(\overline{\frac{Db}{Dt}} - \frac{\overline{Db}}{\overline{Dt}}\right) \approx \frac{-\nabla_h^4 \overline{b}}{\mathrm{Pr}\,\mathrm{Re}_4} + \frac{\frac{\partial^2 \overline{b}}{\partial z^2}}{\mathrm{Pr}\,\mathrm{Re}_v}$$

Griffies and Hallberg (2000) [GH00] propose that if one scales the biharmonic viscosity by stability considerations, then the biharmonic viscous terms will be similarly active to harmonic viscous terms at the gridscale of the model, but much less active on larger scale motions. Similarly, a biharmonic diffusivity can be used for less diffusive flows.

In practice, biharmonic viscosity and diffusivity allow a less viscous, yet numerically stable, simulation than harmonic viscosity and diffusivity. However, there is no physical rationale for such operators being of leading order, and more boundary conditions must be specified than for the harmonic operators. If one considers the approximations of (2.154) - (2.157) and (2.169) - (2.170) to be terms in the Taylor series expansions of the eddy terms as functions of the large-scale gradient, then one can argue that both harmonic and biharmonic terms would occur in the series, and the only question is the choice of coefficients. Using biharmonic viscosity alone implies that one zeros the first non-vanishing term in the Taylor series, which is unsupported by any fluid theory or observation.

Nonetheless, MITgcm supports a plethora of biharmonic viscosities and diffusivities, which are controlled with parameters named similarly to the harmonic viscosities and diffusivities with the substitution h $\rightarrow$ 4 in the MITgcm parameter name. MITgcm also supports biharmonic Leith and Smagorinsky viscosities:

$$A_{4Smag} = \left(\frac{\mathsf{viscC4Smag}}{\pi}\right)^2 \frac{L^4}{8}|D| \tag{2.170}$$

$$A_{4Leith} = \frac{L^5}{8}\sqrt{\left(\frac{\mathsf{viscC4Leith}}{\pi}\right)^6 |\nabla\overline{\omega}_3|^2 + \left(\frac{\mathsf{viscC4LeithD}}{\pi}\right)^6 |\nabla\nabla\cdot\overline{\tilde{\mathbf{u}}}_h|^2} \tag{2.171}$$

However, it should be noted that unlike the harmonic forms, the biharmonic scaling does not easily relate to whether energy-dissipation or enstrophy-dissipation scales are resolved. If similar arguments are used to estimate these scales and scale them to the gridscale, the resulting biharmonic viscosities should be:

$$A_{4Smag} = \left(\frac{\mathsf{viscC4Smag}}{\pi}\right)^5 L^5 |\nabla^2\overline{\tilde{\mathbf{u}}}_h| \tag{2.172}$$

$$A_{4Leith} = L^6\sqrt{\left(\frac{\mathsf{viscC4Leith}}{\pi}\right)^{12} |\nabla^2\overline{\omega}_3|^2 + \left(\frac{\mathsf{viscC4LeithD}}{\pi}\right)^{12} |\nabla^2\nabla\cdot\overline{\tilde{\mathbf{u}}}_h|^2} \tag{2.173}$$

Thus, the biharmonic scaling suggested by Griffies and Hallberg (2000) [GH00] implies:

$$|D| \propto L|\nabla^2\overline{\tilde{\mathbf{u}}}_h|$$
$$|\nabla\overline{\omega}_3| \propto L|\nabla^2\overline{\omega}_3|$$

It is not at all clear that these assumptions ought to hold. Only the Griffies and Hallberg (2000) [GH00] forms are currently implemented in MITgcm.

2.19.1.9 Selection of Length Scale

Above, the length scale of the grid has been denoted L. However, in strongly anisotropic grids, L_x and L_y will be quite different in some locations. In that case, the CFL condition suggests that the minimum of L_x and L_y be used. On the other hand, other viscosities which involve whether a particular wavelength is 'resolved' might be better suited to use the maximum of L_x and L_y. Currently, MITgcm uses useAreaViscLength to select between two options. If false, the square root of the harmonic mean of L_x^2 and L_y^2 is used for all viscosities, which is closer to the minimum and occurs naturally in the CFL constraint. If useAreaViscLength is true, then the square root of the area of the grid cell is used.

2.19.2 Mercator, Nondimensional Equations

The rotating, incompressible, Boussinesq equations of motion (Gill, 1982) [Gil82] on a sphere can be written in Mercator projection about a latitude θ_0 and geopotential height $z = r - r_0$. The nondimensional form of these equations is:

$$\mathrm{Ro}\frac{D\tilde{u}}{Dt} - \frac{\tilde{v}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial x} + \frac{\lambda \mathrm{Fr}^2 M_{Ro}\cos\theta}{\mu\sin\theta_0}w = -\frac{\mathrm{Fr}^2 M_{Ro}\tilde{u}w}{r/H} + \frac{\mathrm{Ro}\hat{\mathbf{x}}\cdot\nabla^2\mathbf{u}}{\mathrm{Re}} \tag{2.174}$$

$$\mathrm{Ro}\frac{D\tilde{v}}{Dt} + \frac{\tilde{u}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial y} = -\frac{\mu\mathrm{Ro}\tan\theta(\tilde{u}^2+\tilde{v}^2)}{r/L} - \frac{\mathrm{Fr}^2 M_{Ro}\tilde{v}w}{r/H} + \frac{\mathrm{Ro}\hat{\mathbf{y}}\cdot\nabla^2\mathbf{u}}{\mathrm{Re}} \tag{2.175}$$

$$\mathrm{Fr}^2\lambda^2\frac{Dw}{Dt} - b + \frac{\partial\pi}{\partial z} - \frac{\lambda\cot\theta_0\tilde{u}}{M_{Ro}} = \frac{\lambda\mu^2(\tilde{u}^2+\tilde{v}^2)}{M_{Ro}(r/L)} + \frac{\mathrm{Fr}^2\lambda^2\hat{\mathbf{z}}\cdot\nabla^2\mathbf{u}}{\mathrm{Re}} \tag{2.176}$$

$$\frac{Db}{Dt} + w = \frac{\nabla^2 b}{\mathrm{Pr}\,\mathrm{Re}}$$

$$\mu^2\left(\frac{\partial\tilde{u}}{\partial x} + \frac{\partial\tilde{v}}{\partial y}\right) + \frac{\partial w}{\partial z} = 0 \tag{2.177}$$

Where

$$\mu \equiv \frac{\cos\theta_0}{\cos\theta}, \quad \tilde{u} = \frac{u^*}{V\mu}, \quad \tilde{v} = \frac{v^*}{V\mu}$$

$$f_0 \equiv 2\Omega\sin\theta_0, \quad \frac{D}{Dt} \equiv \mu^2\left(\tilde{u}\frac{\partial}{\partial x} + \tilde{v}\frac{\partial}{\partial y}\right) + \frac{\mathrm{Fr}^2\mathrm{M_{Ro}}}{\mathrm{Ro}}w\frac{\partial}{\partial z}$$

$$x \equiv \frac{r}{L}\phi\cos\theta_0, \quad y \equiv \frac{r}{L}\int_{\theta_0}^{\theta}\frac{\cos\theta_0\,\mathrm{d}\theta'}{\cos\theta'}, \quad z \equiv \lambda\frac{r-r_0}{L}$$

$$t^* = t\frac{L}{V}, \quad b^* = b\frac{Vf_0 M_{Ro}}{\lambda}$$

$$\pi^* = \pi V f_0 L M_{Ro}, \quad w^* = wV\frac{\mathrm{Fr}^2\lambda M_{Ro}}{\mathrm{Ro}}$$

$$\mathrm{Ro} \equiv \frac{V}{f_0 L}, \quad M_{Ro} \equiv \max[1, \mathrm{Ro}]$$

$$\mathrm{Fr} \equiv \frac{V}{N\lambda L}, \quad \mathrm{Re} \equiv \frac{VL}{\nu}, \quad \mathrm{Pr} \equiv \frac{\nu}{\kappa}$$

Dimensional variables are denoted by an asterisk where necessary. If we filter over a grid scale typical for ocean models:

1m $< L <$ 100km
0.0001 $< \lambda <$ 1
0.001m/s $< V <$ 1 m/s
$f_0 <$ 0.0001 s $^{-1}$
0.01 s $^{-1}$ $< N <$ 0.0001 s $^{-1}$

these equations are very well approximated by

$$\mathrm{Ro}\frac{D\tilde{u}}{Dt} - \frac{\tilde{v}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial x} = -\frac{\lambda\mathrm{Fr}^2 M_{Ro}\cos\theta}{\mu\sin\theta_0}w + \frac{\mathrm{Ro}\nabla^2\tilde{u}}{\mathrm{Re}} \tag{2.178}$$

$$\mathrm{Ro}\frac{D\tilde{v}}{Dt} + \frac{\tilde{u}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial y} = \frac{\mathrm{Ro}\nabla^2\tilde{v}}{\mathrm{Re}} \tag{2.179}$$

$$\mathrm{Fr}^2\lambda^2\frac{Dw}{Dt} - b + \frac{\partial\pi}{\partial z} = \frac{\lambda\cot\theta_0\tilde{u}}{M_{Ro}} + \frac{\mathrm{Fr}^2\lambda^2\nabla^2 w}{\mathrm{Re}} \tag{2.180}$$

$$\frac{Db}{Dt} + w = \frac{\nabla^2 b}{\mathrm{Pr}\,\mathrm{Re}} \tag{2.181}$$

$$\mu^2\left(\frac{\partial\tilde{u}}{\partial x} + \frac{\partial\tilde{v}}{\partial y}\right) + \frac{\partial w}{\partial z} = 0 \tag{2.182}$$

$$\nabla^2 \approx \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\lambda^2\partial z^2}\right)$$

Neglecting the non-frictional terms on the right-hand side is usually called the 'traditional' approximation. It is appropriate, with either large aspect ratio or far from the tropics. This approximation is used here, as it does not affect the form of the eddy stresses which is the main topic. The frictional terms are preserved in this approximate form for later comparison with eddy stresses.

CHAPTER 3

Getting Started with MITgcm

This chapter is divided into two main parts. The first part, which is covered in sections Section 3.1 through Section 3.6, contains information about how to download, build and run MITgcm. We believe the best way to familiarize yourself with the model is to run one of the tutorial examples provided in the MITgcm repository (see Section 4), so would suggest newer MITgcm users jump there following a read-through of the first part of this chapter. Information is also provided in this chapter on how to customize the code when you are ready to try implementing the configuration you have in mind, in the second part (Section 3.8). The code and algorithm are described more fully in Section 2 and Section 6 and chapters thereafter.

In this chapter and others (e.g., chapter *Contributing to the MITgcm*), for arguments where the user is expected to replace the text with a user-chosen name, userid, etc., our convention is to show these as upper-case text surrounded by « », such as «`USER_MUST_REPLACE_TEXT_HERE`». The « and » characters are **NOT** typed when the text is replaced.

3.1 Where to find information

There is a web-archived support mailing list for the model that you can email at MITgcm-support@mitgcm.org once you have subscribed.

To sign up (subscribe) for the mailing list (highly recommended), click here

To browse through the support archive, click here

3.2 Obtaining the code

The MITgcm code and documentation are under continuous development and we generally recommend that one downloads the latest version of the code. You will need to decide if you want to work in a "git-aware" environment (*Method 1*) or with a one-time "stagnant" download (*Method 2*). We generally recommend method 1, as it is more flexible and allows your version of the code to be regularly updated as MITgcm developers check in bug fixes and new features. However, this typically requires at minimum a rudimentary understanding of git in order to make it worth one's while.

Periodically we release an official checkpoint (or "tag"). We recommend one download the latest code, unless there are reasons for obtaining a specific checkpoint (e.g. duplicating older results, collaborating with someone using an older release, etc.)

3.2.1 Method 1

This section describes how to download git-aware copies of the repository. In a terminal window, cd to the directory where you want your code to reside. Type:

```
% git clone https://github.com/MITgcm/MITgcm.git
```

This will download the latest available code. If you now want to revert this code to a specific checkpoint release, first `cd` into the MITgcm directory you just downloaded, then type `git checkout checkpoint«XXX»` where «XXX» is the checkpoint version.

Alternatively, if you prefer to use ssh keys (say for example, you have a firewall which won't allow a https download), type:

```
% git clone git@github.com:MITgcm/MITgcm.git
```

You will need a GitHub account for this, and will have to generate a ssh key though your GitHub account user settings.

The fully git-aware download is over several hundred MB, which is considerable if one has limited internet download speed. In comparison, the one-time download zip file (*Method 2*, below) is order 100MB. However, one can obtain a truncated, yet still git-aware copy of the current code by adding the option `--depth=1` to the git clone command above; all files will be present, but it will not include the full git history. However, the repository can be updated going forward.

3.2.2 Method 2

This section describes how to do a one-time download of MITgcm, NOT git-aware. In a terminal window, `cd` to the directory where you want your code to reside. To obtain the current code, type:

```
% wget https://github.com/MITgcm/MITgcm/archive/master.zip
```

For specific checkpoint release `XXX`, instead type:

```
% wget https://github.com/MITgcm/MITgcm/archive/checkpoint«XXX».zip
```

3.3 Updating the code

There are several different approaches one can use to obtain updates to MITgcm; which is best for you depends a bit on how you intend to use MITgcm and your knowledge of git (and/or willingness to learn). Below we outline three suggested update pathways:

1. **Fresh Download of MITgcm**

This approach is the most simple, and virtually foolproof. Whether you downloaded the code from a static zip file (*Method 2*) or used the git clone command (*Method 1*), create a new directory and repeat this procedure to download a current copy of MITgcm. Say for example you are starting a new research project, this would be a great time to grab the most recent code repository and keep this new work entirely separate from any past simulations. This approach requires no understanding of git, and you are free to make changes to any files in the MIT repo tree (although we

generally recommend that you avoid doing so, instead working in new subdirectories or on separate scratch disks as described *here*, for example).

2. **Using `git pull` to update the (unmodified) MITgcm repo tree**

If you have downloaded the code through a git clone command (*Method 1* above), you can incorporate any changes to the source code (including any changes to any files in the MITgcm repository, new packages or analysis routines, etc.) that may have occurred since your original download. There is a simple command to bring all code in the repository to a 'current release' state. From the MITgcm top directory or any of its subdirectories, type:

```
% git pull
```

and all files will be updated to match the current state of the code repository, as it exists at GitHub. (*Note:* if you plan to contribute to MITgcm and followed the steps to download the code as described in Section 5, you will need to type `git pull upstream` instead.)

This update pathway is ideal if you are in the midst of a project and you want to incorporate new MITgcm features into your executable(s), or take advantage of recently added analysis utilties, etc. After the git pull, any changes in model source code and include files will be updated, so you can repeat the build procedure (Section 3.5) and you will include all these new features in your new executable.

Be forewarned, this will only work if you have not modified ANY of the files in the MITgcm repository (adding new files is ok; also, all verification run subdirectories `build` and `run` are also ignored by git). If you have modified files and the `git pull` fails with errors, there is no easy fix other than to learn something about git (continue reading. . .)

3. **Fully embracing the power of git!**

Git offers many tools to help organize and track changes in your work. For example, one might keep separate projects on different branches, and update the code separately (using `git pull`) on these separate branches. You can even make changes to code in the MIT repo tree; when git then tries to update code from upstream (see Figure 5.1), it will notify you about possible conflicts and even merge the code changes together if it can. You can also use `git commit` to help you track what you are modifying in your simulations over time. If you're planning to submit a pull request to include your changes, you should read the contributing guide in Section 5, and we suggest you do this model development in a separate, fresh copy of the code. See Section 5.2 for more information and how to use git effectively to manage your workflow.

3.4 Model and directory structure

The "numerical" model is contained within a execution environment support wrapper. This wrapper is designed to provide a general framework for grid-point models; MITgcm is a specific numerical model that makes use of this framework (see Section 6.2 for additional detail). Under this structure, the model is split into execution environment support code and conventional numerical model code. The execution environment support code is held under the eesupp directory. The grid point model code is held under the model directory. Code execution actually starts in the eesupp routines and not in the model routines. For this reason the top-level main.F is in the eesupp/src directory. In general, end-users should not need to worry about the wrapper support code. The top-level routine for the numerical part of the code is in model/src/the_model_main.F. Here is a brief description of the directory structure of the model under the root tree.

- model: this directory contains the main source code. Also subdivided into two subdirectories: model/inc (includes files) and model/src (source code).
- eesupp: contains the execution environment source code. Also subdivided into two subdirectories: eesupp/inc and eesupp/src.
- pkg: contains the source code for the packages. Each package corresponds to a subdirectory. For example, pkg/gmredi contains the code related to the Gent-McWilliams/Redi scheme, pkg/seaice the code for a dynamic

seaice model which can be coupled to the ocean model. The packages are described in detail in Section 8 and Section 9].

- doc: contains MITgcm documentation in reStructured Text (rst) format.
- tools: this directory contains various useful tools. For example, genmake2 is a script written in bash that should be used to generate your makefile. The subdirectory tools/build_options contains 'optfiles' with the compiler options for many different compilers and machines that can run MITgcm (see Section 3.5.2.2). This directory also contains subdirectories tools/adjoint_options and tools/OAD_support that are used to generate the tangent linear and adjoint model (see details in Section 7).
- utils: this directory contains various utilities. The utils/matlab subdirectory contains matlab scripts for reading model output directly into matlab. The subdirectory utils/python contains similar routines for python. utils/scripts contains C-shell post-processing scripts for joining processor-based and tiled-based model output.
- verification: this directory contains the model examples. See Section 4.
- jobs: contains sample job scripts for running MITgcm.
- lsopt: Line search code used for optimization.
- optim: Interface between MITgcm and line search code.

3.5 Building the model

3.5.1 Quickstart Guide

To compile the code, we use the `make` program. This uses a file (`Makefile`) that allows us to pre-process source files, specify compiler and optimization options and also figures out any file dependencies. We supply a script (genmake2), described in section Section 3.5.2, that automatically generates the `Makefile` for you. You then need to build the dependencies and compile the code (Section 3.5.3).

As an example, assume that you want to build and run experiment verification/exp2. Let's build the code in verification/exp2/build:

```
% cd verification/exp2/build
```

First, generate the `Makefile`:

```
% ../../../tools/genmake2 -mods ../code -optfile «/PATH/TO/OPTFILE»
```

The `-mods` command line option tells genmake2 to override model source code with any files in the subdirectory `../code` (here, you need to configure the size of the model domain by overriding MITgcm's default SIZE.h with an edited copy ../code/SIZE.h containing the specific domain size for exp2).

The `-optfile` command line option tells genmake2 to run the specified *optfile*, a bash shell script, during genmake2's execution. An *optfile* typically contains definitions of environment variables, paths, compiler options, and anything else that needs to be set in order to compile on your local computer system or cluster with your specific Fortan compiler. As an example, we might replace «`/PATH/TO/OPTFILE`» with ../../../tools/build_options/linux_amd64_ifort11 for use with the Intel Fortran compiler (version 11 and above) on a linux x86_64 platform. This and many other *optfiles* for common systems and Fortran compilers are located in tools/build_options.

`-mods`, `-optfile`, and many additional genmake2 command line options are described more fully in Section 3.5.2.1. Detailed instructions on building with MPI are given in Section 3.5.4.

Once a `Makefile` has been generated, we create the dependencies with the command:

```
% make depend
```

It is important to note that the `make depend` stage will occasionally produce warnings or errors if the dependency parsing tool is unable to find all of the necessary header files (e.g., `netcdf.inc`, or worse, say it cannot find a Fortran compiler in your path). In some cases you may need to obtain help from your system administrator to locate these files.

Next, one can compile the code using:

```
% make
```

Assuming no errors occurred, the `make` command creates an executable called `mitgcmuv`.

Now you are ready to run the model. General instructions for doing so are given in section Section 3.6.

3.5.2 Generating a `Makefile` using genmake2

A shell script called `genmake2` for generating a `Makefile` is included as part of MITgcm. Typically `genmake2` is used in a sequence of steps as shown below:

```
% ../../../tools/genmake2 -mods ../code -optfile «/PATH/TO/OPTFILE»
% make depend
% make
```

The first step above creates a unix-style `Makefile`. The `Makefile` is used by `make` to specify how to compile the MITgcm source files (for more detailed descriptions of what the `make` tools are, and how they are used, see here).

This section describes details and capabilities of genmake2, located in the tools directory. genmake2 is a shell script written to work in bash (and with all "sh"–compatible shells including Bourne shells). Like many unix tools, there is a help option that is invoked thru `genmake2 -h`. genmake2 parses information from the following sources, in this order:

1. Command-line options (see Section 3.5.2.1)
2. A `genmake_local` file if one is found in the current directory. This is a bash shell script that is executed prior to the *optfile* (see step #3), used in some special model configurations and/or to set some options that can affect which lines of the *optfile* are executed. For example, this genmake_local file is required for a special setup, building a 'MITgcm coupler' executable; in a more typical setup, one will not require a `genmake_local` file.
3. An "options file" a.k.a. *optfile* (a bash shell script) specified by the command-line option `-optfile «/PATH/TO/OPTFILE»`, as mentioned briefly in Section 3.5.1 and described in detail in Section 3.5.2.2.
4. A `packages.conf` file (if one is found) with the specific list of packages to compile (see Section 8.1.1). The search path for file `packages.conf` is first the current directory, and then each of the `-mods` directories in the given order (as described *here*).

When you run the genmake2 script, typical output might be as follows:

```
% ../../../tools/genmake2 -mods ../code -optfile ../../../tools/build_options/linux_
↪amd64_gfortran

GENMAKE :

A program for GENerating MAKEfiles for the MITgcm project.
   For a quick list of options, use "genmake2 -h"
or for more detail see the documentation, section "Building the model"
   (under "Getting Started") at:  https://mitgcm.readthedocs.io/
```

(continues on next page)

(continued from previous page)

```
===  Processing options files and arguments  ===
  getting local config information:  none found
Warning: ROOTDIR was not specified ; try using a local copy of MITgcm found at "../../
↪.."
  getting OPTFILE information:
    using OPTFILE="../../../tools/build_options/linux_amd64_gfortran"
  getting AD_OPTFILE information:
    using AD_OPTFILE="../../../tools/adjoint_options/adjoint_default"
  check Fortran Compiler...  pass  (set FC_CHECK=5/5)
  check makedepend (local: 0, system: 1, 1)

===  Checking system libraries  ===
  Do we have the system() command using gfortran...  yes
  Do we have the fdate() command using gfortran...  yes
  Do we have the etime() command using gfortran... c,r: yes (SbR)
  Can we call simple C routines (here, "cloc()") using gfortran...  yes
  Can we unlimit the stack size using gfortran...  yes
  Can we register a signal handler using gfortran...  yes
  Can we use stat() through C calls...  yes
  Can we create NetCDF-enabled binaries...  yes
    skip check for LAPACK Libs
  Can we call FLUSH intrinsic subroutine...  yes

===  Setting defaults  ===
  Adding MODS directories: ../code
  Making source files in eesupp from templates
  Making source files in pkg/exch2 from templates
  Making source files in pkg/regrid from templates

===  Determining package settings  ===
  getting package dependency info from  ../../../pkg/pkg_depend
  getting package groups info from      ../../../pkg/pkg_groups
  checking list of packages to compile:
    using PKG_LIST="../code/packages.conf"
    before group expansion packages are: oceanic -kpp -gmredi cd_code
    replacing "oceanic" with:  gfd gmredi kpp
    replacing "gfd" with:  mom_common mom_fluxform mom_vecinv generic_advdiff debug␣
↪mdsio rw monitor
    after group expansion packages are:  mom_common mom_fluxform mom_vecinv generic_
↪advdiff debug mdsio rw monitor gmredi kpp -kpp -gmredi cd_code
  applying DISABLE settings
  applying ENABLE settings
    packages are:  cd_code debug generic_advdiff mdsio mom_common mom_fluxform mom_
↪vecinv monitor rw
  applying package dependency rules
    packages are:  cd_code debug generic_advdiff mdsio mom_common mom_fluxform mom_
↪vecinv monitor rw
  Adding STANDARDDIRS='eesupp model'
  Searching for *OPTIONS.h files in order to warn about the presence
    of "#define "-type statements that are no longer allowed:
    found CPP_EEOPTIONS="../../../eesupp/inc/CPP_EEOPTIONS.h"
    found CPP_OPTIONS="../../../model/inc/CPP_OPTIONS.h"
  Creating the list of files for the adjoint compiler.

===  Creating the Makefile  ===
  setting INCLUDES
```

(continues on next page)

(continued from previous page)

```
  Determining the list of source and include files
  Writing makefile: Makefile
  Add the source list for AD code generation
  Making list of "exceptions" that need ".p" files
  Making list of NOOPTFILES
  Add rules for links
  Adding makedepend marker

===  Done  ===
  original 'Makefile' generated successfully
=> next steps:
  > make depend
  > make       (<-- to generate executable)
```

In the above, notice:

- we did not specify `ROOTDIR`, i.e., a path to your MITgcm repository, but here we are building code from within the repository (specifically, in one of the verification subdirectory experiments). As such, genmake2 was smart enough to locate all necessary files on its own. To specify a remote `ROOTDIR`, see *here*.
- we specified the *optfile* linux_amd64_gfortran based on the computer system and Fortran compiler we used (here, a linux 64-bit machine with gfortran installed).
- genmake2 did some simple checking on availability of certain system libraries; all were found (except LAPACK, which was not checked since it is not needed here). NetCDF only requires a 'yes' if you want to write netCDF output; more specifically, a 'no' response to "Can we create NetCDF-enabled binaries" will disable including pkg/mnc and switch to output plain binary files. While the makefile can still be built with other 'no' responses, sometimes this will foretell errors during the `make depend` or `make` commands.
- any `.F` or `.h` files in the `-mods` directory `../code` will also be compiled, overriding any MITgcm repository versions of files, if they exist.
- a handful of packages are being used in this build; see Section 8.1.1 for more detail about how to enable and disable packages.
- genmake2 terminated without error (note output at end after `=== Done ===`), generating `Makefile` and a log file `genmake.log`. As mentioned, this does not guarantee that your setup will compile properly, but if there are errors during `make depend` or `make`, these error messages and/or the standard output from genmake2 or `genmake.log` may provide clues as to the problem. If instead genmake2 finishes with a warning message `Warning: FORTRAN compiler test failed`, this means that genmake2 is unable to locate the Fortran compiler or pass a trivial "hello world" Fortran compilation test. In this case, you should see `genmake.log` for errors and/or seek assistance from your system administrator; these tests need to pass in order to proceed to the `make` steps.

3.5.2.1 Command-line options:

genmake2 supports a number of helpful command-line options. A complete list of these options can be obtained by:

```
% genmake2 -h
```

The most important command-line options are:

`-optfile «/PATH/TO/OPTFILE»` (or shorter: `-of`) specifies the *optfile* that should be used for a particular build.

If no *optfile* is specified through the command line, genmake2 will try to make a reasonable guess from the list provided in tools/build_options. The method used for making this guess is to first determine the combination

of operating system and hardware and then find a working Fortran compiler within the user's path. When these three items have been identified, genmake2 will try to find an *optfile* that has a matching name. See Section 3.5.2.2.

-mods '«DIR1 DIR2 DIR3 ...»' specifies a list of directories containing "modifications". These directories contain files with names that may (or may not) exist in the main MITgcm source tree but will be overridden by any identically-named sources within the `-mods` directories. Note the quotes around the list of directories, necessary given multiple arguments.

The order of precedence for versions of files with identical names:

- "mods" directories in the order given (e.g., will use copy of file located in DIR1 instead of DIR2)
- Packages either explicitly specified or included by default
- Packages included due to package dependencies
- The "standard dirs" (which may have been specified by the `-standarddirs` option below)

-rootdir «/PATH/TO/MITGCMDIR» specify the location of the MITgcm repository top directory (`ROOTDIR`). By default, genmake2 will try to find this location by looking in parent directories from where genmake2 is executed (up to 5 directory levels above the current directory).

In the quickstart example above (Section 3.5.1) we built the executable in the `build` directory of the experiment. Below, we show how to configure and compile the code on a scratch disk, without having to copy the entire source tree. The only requirement is that you have genmake2 in your $PATH, or you know the absolute path to genmake2. In general, one can compile the code in any given directory by following this procedure. Assuming the model source is in `~/MITgcm`, then the following commands will build the model in `/scratch/exp2-run1`:

```
% cd /scratch/exp2-run1
% ~/MITgcm/tools/genmake2 -rootdir ~/MITgcm -mods ~/MITgcm/verification/exp2/code
% make depend
% make
```

As an alternative to specifying the MITgcm repository location through the `-rootdir` command-line option, genmake2 recognizes the environment variable `$MITGCM_ROOTDIR`.

-standarddirs «/PATH/TO/STANDARDDIR» specify a path to the standard MITgcm directories for source and includes files. By default, model and eesupp directories (`src` and `inc`) are the "standard dirs". This command can be used to reset these default standard directories, or instead NOT include either model or eesupp as done in some specialized configurations.

-oad generates a makefile for an OpenAD build (see Section 7.5)

-adoptfile «/PATH/TO/FILE» (or shorter: `-adof`) specifies the "adjoint" or automatic differentiation options file to be used. The file is analogous to the optfile defined above but it specifies information for the AD build process. See Section 7.2.3.4.

The default file is located in tools/adjoint_options/adjoint_default and it defines the "TAF" and "TAMC" compiler options.

-mpi enables certain MPI features (using CPP `#define`) within the code and is necessary for MPI builds (see Section 3.5.4).

-omp enables OpenMP code and compiler flag `OMPFLAG` (see Section 3.5.5).

-ieee use IEEE numerics (requires support in optfile). This option is typically a good choice if one wants to compare output from different machines running the same code. Note using IEEE disables all compiler optimizations.

-devel use IEEE numerics (requires support in optfile) and add additional compiler options to check array bounds and add other additional warning and debugging flags.

-make «/PATH/TO/GMAKE» due to the poor handling of soft-links and other bugs common with the `make` versions provided by commercial unix vendors, GNU `make` (sometimes called `gmake`) may be preferred. This option provides a means for specifying the make executable to be used.

While it is possible to use genmake2 command-line options to set the Fortran or C compiler name (`-fc` and `-cc` respectively), we generally recommend setting these through an *optfile*, as discussed in Section 3.5.2.2. Other genmake2 options are available to enable performance/timing analyses, etc.; see `genmake2 -h` for more info.

3.5.2.2 Optfiles in tools/build_options directory:

The purpose of the optfiles is to provide all the compilation options for particular "platforms" (where "platform" roughly means the combination of the hardware and the compiler) and code configurations. Given the combinations of possible compilers and library dependencies (e.g., MPI or netCDF) there may be numerous optfiles available for a single machine. The naming scheme for the majority of the optfiles shipped with the code is **OS_HARDWARE_COMPILER** where

OS is the name of the operating system (generally the lower-case output of a linux terminal `uname` command)

HARDWARE is a string that describes the CPU type and corresponds to output from a `uname -m` command. Some common CPU types:

`amd64` use this code for x86_64 systems (most common, including AMD and Intel 64-bit CPUs)

`ia64` is for Intel IA64 systems (eg. Itanium, Itanium2)

`ppc` is for (old) Mac PowerPC systems

COMPILER is the compiler name (generally, the name of the Fortran compiler executable). MITgcm is primarily written in FORTRAN 77. Compiling the code requires a FORTRAN 77 compiler. Any more recent compiler which is backwards compatible with FORTRAN 77 can also be used; for example, the model will build successfully with a Fortran 90 or Fortran 95 compiler. A C99 compatible compiler is also need, together with a C preprocessor . Some optional packages make use of Fortran 90 constructs (either free-form formatting, or dynamic memory allocation); as such, setups which use these packages require a Fortran 90 or later compiler build.

There are existing optfiles that work with many common hardware/compiler configurations; we first suggest you peruse the list in tools/build_options and try to find your platform/compiler configuration. These are the most common:

- linux_amd64_gfortran
- linux_amd64_ifort11
- linux_amd64_ifort+impi
- linux_amd64_pgf77

The above optfiles are all for linux x86_64 (64-bit) systems, utilized in many large high-performance computing centers. All of the above will work with single-threaded, MPI, or shared memory (OpenMP) code configurations. gfortran is GNU Fortran, ifort is Intel Fortran, pgf77 is PGI Fortran (formerly known as "The Portland Group"). Note in the above list there are two `ifort` optfiles: linux_amd64_ifort+impi is for a specific case of using `ifort` with the Intel MPI library (a.k.a. `impi`), which requires special define statements in the optfile (in contrast with Open MPI or MVAPICH2 libraries; see Section 3.5.4). Note that both ifort optfiles require ifort version 11 or higher. Many clusters nowadays use environment modules, which allows one to easily choose which compiler to use through `module load «MODULENAME»`, automatically configuring your environment for a specific compiler choice (type `echo $PATH` to see where genmake2 will look for compilers and system software).

In most cases, your platform configuration will be included in the available optfiles list and will result in a usable `Makefile` being generated. If you are unsure which optfile is correct for your configuration, you can try not specifying an optfile; on some systems the genmake2 program will be able to automatically recognize the hardware, find a compiler and other tools within the user's path, and then make a best guess as to an appropriate optfile from the list in

the tools/build_options directory. However, for some platforms and code configurations, new optfiles must be written. To create a new optfile, it is generally best to start with one of the defaults and modify it to suit your needs. Like genmake2, the optfiles are all written in bash (or using a simple sh–compatible syntax). While nearly all environment variables used within genmake2 may be specified in the optfiles, the critical ones that should be defined are:

FC the Fortran compiler (executable) to use on `.F` files, e.g., `ifort` or `gfortran`, or if using MPI, the mpi-wrapper equivalent, e.g., `mpif77`

F90C the Fortran compiler to use on `.F90` files (only necessary if your setup includes a package which contains `.F90` source code)

CC similarly, the C compiler to use, e.g., `icc` or `gcc`, or if using MPI, the mpi-wrapper equivalent, e.g., `mpicc`

DEFINES command-line options passed to the compiler

CPP the C preprocessor to use, and any necessary command-line options, e.g. `cpp -traditional -P`

CFLAGS, FFLAGS command-line compiler flags required for your C and Fortran compilers, respectively, to compile and execute properly. See your C and Fortran compiler documentation for specific options and syntax.

FOPTIM command-line optimization Fortran compiler settings. See your Fortran compiler documentation for specific options and syntax.

NOOPTFLAGS command-line settings for special files that should not be optimized using the `FOPTIM` flags

NOOPTFILES list of source code files that should be compiled using `NOOPTFLAGS` settings

INCLUDES path for additional files (e.g., `netcdf.inc`, `mpif.h`) to include in the compilation using the command-line `-I` option

INCLUDEDIRS path for additional files to be included in the compilation

LIBS path for additional library files that need to be linked to generate the final executable, e.g., `libnetcdf.a`

For example, an excerpt from an optfile which specifies several of these variables (here, for the linux-amd64 architecture using the PGI Fortran compiler) is as follows:

```
if test "x$MPI" = xtrue ; then
  CC=mpicc
  FC=mpif77
  F90C=mpif90
else
  CC=pgcc
  FC=pgf77
  F90C=pgf90
fi

DEFINES="-DWORDLENGTH=4"
if test "x$ALWAYS_USE_F90" = x1 ; then
    FC=$F90C
else
    DEFINES="$DEFINES -DNML_EXTENDED_F77"
fi
CPP='cpp -traditional -P'
F90FIXEDFORMAT='-Mfixed'
EXTENDED_SRC_FLAG='-Mextend'
GET_FC_VERSION="-V"
OMPFLAG='-mp'

NOOPTFLAGS='-O0'
NOOPTFILES=''
```

(continues on next page)

(continued from previous page)

```
FFLAGS="$FFLAGS -byteswapio -Ktrap=fp"
#- might want to use '-r8' for fizhi pkg:
#FFLAGS="$FFLAGS -r8"

if test "x$IEEE" = x ; then        #- with optimisation:
    FOPTIM='-tp k8-64 -pc=64 -O2 -Mvect=sse'
   #FOPTIM="$FOPTIM -fastsse -O3 -Msmart -Mvect=cachesize:1048576,transform"
else                                #- no optimisation + IEEE :
   #FFLAGS="$FFLAGS -Mdclchk"  #- pkg/zonal_filt does not pass with declaration-check
    FOPTIM='-pc=64 -O0 -Kieee'
fi

F90FLAGS=$FFLAGS
F90OPTIM=$FOPTIM
```

The *above* list of environment variables typically specified in an optfile is by no means complete; additional variables may be required for your specific setup and/or your specific Fortran (or C) compiler.

If you write an optfile for an unrepresented machine or compiler, you are strongly encouraged to submit the optfile to the MITgcm project for inclusion. MITgcm developers are willing to provide help writing or modifing optfiles. Please submit the file through the GitHub issue tracker or email the MITgcm-support@mitgcm.org list.

Instructions on how to use optfiles to build MPI-enabled executables is presented in Section 3.5.4.

3.5.3 `make` commands

Following a successful build of `Makefile`, type `make depend`. This command modifies the `Makefile` by attaching a (usually, long) list of files upon which other files depend. The purpose of this is to reduce re-compilation if and when you start to modify the code. The `make depend` command also creates local links for all source files from the source directories (see "-mods" description in Section 3.5.2.1), so that all source files to be used are visible from the local build directory, either as hardcopy or as symbolic link.

IMPORTANT NOTE: Editing the source code files in the build directory will not edit a local copy (since these are just links) but will edit the original files in model/src (or model/inc) or in the specified `-mods` directory. While the latter might be what you intend, editing the master copy in model/src is usually **NOT** what is intended and may cause grief somewhere down the road. Rather, if you need to add to the list of modified source code files, place a copy of the file(s) to edit in the `-mods` directory, make the edits to these `-mods` directory files, go back to the build directory and type `make Clean`, and then re-build the makefile (these latter steps critical or the makefile will not link to this newly edited file).

The final `make` invokes the C preprocessor to produce the "little f" files (`*.f` and `*.f90`) and then compiles them to object code using the specified Fortran compiler and options. The C preprocessor step converts a number of CPP macros and `#ifdef` statements to actual Fortran and expands C-style `#include` statements to incorporate header files into the "little f" files. CPP style macros and `#ifdef` statements are used to support generating different compile code for different model configurations. The result of the build process is an executable with the name `mitgcmuv`.

Additional make "targets" are defined within the makefile to aid in the production of adjoint (Section 7.2.2) and other versions of MITgcm.

On computers with multiple processor cores, the build process can often be sped up appreciably using the command:

```
% make -j 2
```

where the "2" can be replaced with a number that corresponds to the number of cores (or discrete CPUs) available.

In addition, there are several housekeeping `make clean` options that might be useful:

- `make clean` removes files that `make` generates (e.g., *.o and *.f files)
- `make Clean` removes files and links generated by `make` and `make depend`; strongly recommended for "un-clean" directories which may contain the (perhaps partial) results of previous builds
- `make CLEAN` removes pretty much everything, including any executables and output from genmake2

3.5.4 Building with MPI

Building MITgcm to use MPI libraries can be complicated due to the variety of different MPI implementations available, their dependencies or interactions with different compilers, and their often ad-hoc locations within file systems. For these reasons, its generally a good idea to start by finding and reading the documentation for your machine(s) and, if necessary, seeking help from your local systems administrator.

The steps for building MITgcm with MPI support are:

1. Make sure you have MPI libraries installed on your computer system or cluster. Different Fortran compilers (and different versions of a specific compiler) will generally require a custom version (of a MPI library) built specifically for it. On environment module-enabled clusters, one typically must first load a Fortran compiler, then specific MPI libraries for that compiler will become available to load. If libraries are not installed, MPI implementations and related tools are available including:

   - Open MPI
   - MVAPICH2
   - MPICH
   - Intel MPI

   Ask you systems administrator for assistance in installing these libraries.

2. Determine the location of your MPI library "wrapper" Fortran compiler, e.g., `mpif77` or `mpifort` etc. which will be used instead of the name of the fortran compiler (`gfortran`, `ifort`, `pgi77` etc.) to compile your code. Often the directory in which these wrappers are located will be automatically added to your $PATH environment variable when you perform a `module load «SOME_MPI_MODULE»`; thus, you will not need to do anything beyond the module load itself. If you are on a cluster that does not support environment modules, you may have to manually add this directory to your path, e.g., type `PATH=$PATH:«ADD_ADDITIONAL_PATH_TO_MPI_WRAPPER_HERE»` in a bash shell.

3. Determine the location of the includes file `mpif.h` and any other MPI-related includes files. Often these files will be located in a subdirectory off the main MPI library `include/`. In all optfiles in tools/build_options, it is assumed environment variable `$MPI_INC_DIR` specifies this location; `$MPI_INC_DIR` should be set in your terminal session prior to generating a `Makefile`.

4. Determine how many processors (i.e., CPU cores) you will be using in your run, and modify your configuration's SIZE.h (located in a "modified code" directory, as specified in your genmake2 *command-line*). In SIZE.h, you will need to set variables nPx*nPy to match the number of processors you will specify in your run script's MITgcm execution statement (i.e., typically `mpirun` or some similar command, see Section 3.6.1). Note that MITgcm does not use dynamic memory allocation (a feature of Fortran 90, not FORTRAN 77), so all array sizes, and hence the number of processors to be used in your MPI run, must be specified at compile-time in addition to run-time. More information about the MITgcm WRAPPER, domain decomposition, and how to configure SIZE.h can be found in Section 6.3.

5. Build the code with the genmake2 `-mpi` option using commands such as:

```
%  ../../../tools/genmake2 -mods=../code -mpi -of=«/PATH/TO/OPTFILE»
%  make depend
%  make
```

3.5.5 Building with OpenMP

Unlike MPI, which requires installation of additional software support libraries, using shared memory (OpenMP) for multi-threaded executable builds can be accomplished simply through the genmake2 command-line option `-omp`:

```
%  ../../../tools/genmake2 -mods=../code -omp -of=«/PATH/TO/OPTFILE»
%  make depend
%  make
```

While the most common optfiles specified in Section 3.5.2.2 include support for the `-omp` option, some optfiles in tools/build_options do not include support for multi-threaded executable builds. Before using one of the less common optfiles, check whether `OMPFLAG` is defined.

Note that one does not need to specify the number of threads until runtime (see Section 3.6.2). However, the default maximum number of threads in MITgcm is set to a (low) value of 4, so if you plan on more you will need to change this value in eesupp/inc/EEPARAMS.h in your modified code directory.

3.6 Running the model

If compilation finished successfully (Section 3.5) then an executable called `mitgcmuv` will now exist in the local (`build`) directory.

To run the model as a single process (i.e., not in parallel) simply type (assuming you are still in the `build` directory):

```
% cd ../run
% ln -s ../input/* .
% cp ../build/mitgcmuv .
% ./mitgcmuv
```

Here, we are making a link to all the support data files (in `../input/`) needed by the MITgcm for this experiment, and then copying the executable from the the build directory. The `./` in the last step is a safe-guard to make sure you use the local executable in case you have others that might exist in your `$PATH`. The above command will spew out many lines of text output to your screen. This output contains details such as parameter values as well as diagnostics such as mean kinetic energy, largest CFL number, etc. It is worth keeping this text output with the binary output so we normally re-direct the `stdout` stream as follows:

```
% ./mitgcmuv > output.txt
```

In the event that the model encounters an error and stops, it is very helpful to include the last few line of this `output.txt` file along with the (`stderr`) error message within any bug reports.

For the example experiment in verification/exp2, an example of the output is kept in verification/exp2/results/output.txt for comparison. You can compare your `output.txt` with the corresponding one for that experiment to check that your set-up indeed works. Congratulations!

3.6.1 Running with MPI

Run the code with the appropriate MPI "run" or "exec" program provided with your particular implementation of MPI. Typical MPI packages such as Open MPI will use something like:

```
%  mpirun -np 4 ./mitgcmuv
```

Sightly more complicated scripts may be needed for many machines since execution of the code may be controlled by both the MPI library and a job scheduling and queueing system such as Slurm, PBS/TORQUE, LoadLeveler, or

any of a number of similar tools. See your local cluster documentation or system administrator for the specific syntax required to run on your computing facility.

3.6.2 Running with OpenMP

Assuming the executable `mitgcmuv` was built with OpenMP (see Section 3.5.5), the syntax to run a multi-threaded simulation is the same as running single-threaded (see Section 3.6), except that the following additional steps are required beforehand:

1. Environment variables for the number of threads and the stacksize need to be set prior to executing the model. The exact names of these environment variables differ by Fortran compiler, but are typically some variant of `OMP_NUM_THREADS` and `OMP_STACKSIZE`, respectively. For the latter, in your run script we recommend adding the line `export OMP_STACKSIZE=400M` (or for a C shell-variant, `setenv OMP_STACKSIZE 400M`). If this stacksize setting is insufficient, MITgcm will crash, in which case a larger number can be used. Similarly, `OMP_NUM_THREADS` should be set to the exact number of threads you require.
2. In file `eedata` you will need to change namelist parameters nTx and nTy to reflect the number of threads in x and y, respectively (for a single-threaded run, nTx =nTy=1). The value of nTx *nTy must equal the value of environment variable `OMP_NUM_THREADS` (or its name-equivalent for your Fortan compiler) or MITgcm will terminate during its initialization with an error message.

MITgcm will take the number of tiles used in the model (as specified in SIZE.h) and the number of threads (nTx and nTy from file `eedata`), and in running will spread the tiles out evenly across the threads. This is done independently for x and y. As such, the number of tiles in x (variable nSx as defined in SIZE.h) must divide evenly by the number of threads in x (namelist parameter nTx), and similarly for nSy and nTy, else MITgcm will terminate on initialization. More information about the MITgcm WRAPPER, domain decomposition, and how to configure SIZE.h can be found in Section 6.3.

3.6.3 Output files

The model produces various output files and, when using pkg/mnc (i.e., netCDF), sometimes even directories. Depending upon the I/O package(s) selected at compile time (either pkg/mdsio, pkg/mnc, or both as determined by `packages.conf`) and the run-time flags set (in `data.pkg`), the following output may appear. More complete information describing output files and model diagnostics is described in Section 9.

3.6.3.1 Raw binary output files

The "traditional" output files are generated by the pkg/mdsio (see Section 9.2).The pkg/mdsio model data are written according to a "meta/data" file format. Each variable is associated with two files with suffix names `.data` and `.meta`. The `.data` file contains the data written in binary form (big endian by default). The `.meta` file is a "header" file that contains information about the size and the structure of the `.data` file. This way of organizing the output is particularly useful when running multi-processors calculations.

At a minimum, the instantaneous "state" of the model is written out, which is made of the following files:

- `U.00000nIter` - zonal component of velocity field (m/s and positive eastward).
- `V.00000nIter` - meridional component of velocity field (m/s and positive northward).
- `W.00000nIter` - vertical component of velocity field (ocean: m/s and positive upward, atmosphere: Pa/s and positive towards increasing pressure i.e., downward).
- `T.00000nIter` - potential temperature (ocean: °C, atmosphere: °K).
- `S.00000nIter` - ocean: salinity (psu), atmosphere: water vapor (g/kg).
- `Eta.00000nIter` - ocean: surface elevation (m), atmosphere: surface pressure anomaly (Pa).

The chain `00000nIter` consists of ten figures that specify the iteration number at which the output is written out. For example, `U.0000000300` is the zonal velocity at iteration 300.

In addition, a "pickup" or "checkpoint" file called:

- `pickup.00000nIter`

is written out. This file represents the state of the model in a condensed form and is used for restarting the integration (at the specific iteration number). Some additional parameterizations and packages also produce separate pickup files, e.g.,

- `pickup_cd.00000nIter` if the C-D scheme is used (see *C_D Scheme*)
- `pickup_seaice.00000nIter` if the seaice package is turned on (see *SEAICE Package*)
- `pickup_ptracers.00000nIter` if passive tracers are included in the simulation (see *PTRACERS Package*)

Rolling checkpoint files are the same as the pickup files but are named differently. Their name contain the chain `ckptA` or `ckptB` instead of `00000nIter`. They can be used to restart the model but are overwritten every other time they are output to save disk space during long integrations.

3.6.3.2 NetCDF output files

pkg/mnc is a set of routines written to read, write, and append netCDF files. Unlike the pkg/mdsio output, the pkg/mnc–generated output is usually placed within a subdirectory with a name such as `mnc_output_` (by default, netCDF tries to append, rather than overwrite, existing files, so a unique output directory is helpful for each separate run).

The pkg/mnc output files are all in the "self-describing" netCDF format and can thus be browsed and/or plotted using tools such as:

- ncdump is a utility which is typically included with every netCDF install, and converts the netCDF binaries into formatted ASCII text files.
- ncview is a very convenient and quick way to plot netCDF data and it runs on most platforms. Panoply is a similar alternative.
- MATLAB, GrADS, IDL and other common post-processing environments provide built-in netCDF interfaces.

3.6.4 Looking at the output

3.6.4.1 MATLAB

Raw binary output

The repository includes a few MATLAB utilities to read binary output files written in the /pkg/mdsio format. The MATLAB scripts are located in the directory utils/matlab under the root tree. The script utils/matlab/rdmds.m reads the data. Look at the comments inside the script to see how to use it.

Some examples of reading and visualizing some output in Matlab:

```
% matlab
>> H=rdmds('Depth');
>> contourf(H');colorbar;
>> title('Depth of fluid as used by model');

>> eta=rdmds('Eta',10);
```

(continues on next page)

(continued from previous page)

```
>> imagesc(eta');axis ij;colorbar;
>> title('Surface height at iter=10');

>> eta=rdmds('Eta',[0:10:100]);
>> for n=1:11; imagesc(eta(:,:,n)');axis ij;colorbar;pause(.5);end
```

NetCDF output

Similar scripts for netCDF output (e.g., utils/matlab/rdmnc.m) are available and they are described in Section 9.3.

3.6.4.2 Python

Raw binary output

The repository includes Python scripts for reading binary /pkg/mdsio format under utils/python. The following example shows how to load in some data:

```
# python
import mds

Eta = mds.rdmds('Eta', itrs=10)
```

The docstring for `mds.rdmds` (see file utils/python/MITgcmutils/MITgcmutils/mds.py) contains much more detail about using this function and the options that it takes.

NetCDF output

The netCDF output is currently produced with one file per processor. This means the individual tiles need to be stitched together to create a single netCDF file that spans the model domain. The script utils/python/MITgcmutils/scripts/gluemncbig can do this efficiently from the command line.

The following example shows how to use the xarray python package to read the resulting netCDF file into Python:

```
# python
import xarray as xr

Eta = xr.open_dataset('Eta.nc')
```

3.7 Customizing the Model Configuration - Code Parameters and Compilation Options

3.7.1 Model Array Dimensions

MITgcm's array dimensions need to be configured for each unique model domain. The size of each tile (in dimensions x, y, and vertical coordinate r) the "overlap" region of each tile (in x and y), the number of tiles in the x and y dimensions, and the number of processes (using MPI) in the x and y dimensions all need to be specified in SIZE.h. From these parameters, global domain-size variables Nx, Ny are computed by the model. See a more detailed discussion of SIZE.h parameters in the *barotropic gyre tutorial* and a more technical discussion in Section 6.3.1.

Parameter	Default SIZE.h	Description
sNx	30	number of points in x dimension in a single tile
sNy	15	number of points in y dimension in a single tile
Nr	4	number of points in r dimension
OLx	2	number of "overlap" points in x dimension for a tile
OLy	2	number of "overlap" points in y dimension for a tile
nSx	2	number of tile per process in x dimension
nSy	4	number of tile per process in y dimension
nPx	1	number of processes in x dimension
nPy	1	number of processes in y dimension

Note the repository version of SIZE.h includes several lines of text at the top that will halt compilation with errors. Thus, to use MITgcm you will need to copy SIZE.h to a code modification directory and make edits, including deleting or commenting out the offending lines of text.

3.7.2 C Preprocessor Options

The CPP flags relative to the "numerical model" part of the code are defined and set in the file CPP_OPTIONS.h in the directory model/inc/. In the parameter tables in Section 3.8 we have noted CPP options **that need to be changed from the default** to enable specific runtime parameter to be used properly. Also note many of the options below are for less-common situations or are somewhat obscure, so newer users of the MITgcm are encouraged to jump to Section 3.8 where more basic runtime parameters are discussed.

CPP Flag Name	Default	Description
SHORTWAVE_HEATING	#undef	provide separate shortwave heating file, allowing shortwave to penetrate below surface layer
ALLOW_GEOTHERMAL_FLUX	#undef	include code for applying geothermal heat flux at the bottom of the ocean
ALLOW_FRICTION_HEATING	#undef	include code to allow heating due to friction (and momentum dissipation)
ALLOW_ADDFLUID	#undef	allow mass source or sink of fluid in the interior (3D generalization of oceanic real-fresh water flux)
ATMOSPHERIC_LOADING	#define	include code for atmospheric pressure-loading (and seaice-loading) on ocean surface
ALLOW_BALANCE_FLUXES	#undef	include balancing surface forcing fluxes code
ALLOW_BALANCE_RELAX	#undef	include balancing surface forcing relaxation code
CHECK_SALINITY_FOR_NEGATIVE_VALUES	#undef	include code checking for negative salinity
EXCLUDE_FFIELDS_LOAD	#undef	exclude external forcing-fields load; code allows reading and simple linear time interpolation of oceanic forcing fields, if no specific pkg (e.g., pkg/exf) is used to compute them
INCLUDE_PHIHYD_CALCULATION_CODE	#define	include code to calculate ϕ_{hyd}
INCLUDE_CONVECT_CALL	#define	include code for convective adjustment mixing algorithm
INCLUDE_CALC_DIFFUSIVITY_CALL	#define	include codes that calculates (tracer) diffusivities and viscosities
ALLOW_3D_DIFFKR	#undef	allow full 3D specification of vertical diffusivity
ALLOW_BL79_LAT_VARY	#undef	allow latitudinally varying Bryan and Lewis 1979 [BL79] vertical diffusivity

Continued on next page

Table 3.1 – continued from previous page

CPP Flag Name	Default	Description
EXCLUDE_PCELL_MIX_CODE	#undef	exclude code for partial-cell effect (physical or enhanced) in vertical mixing; this allows accounting for partial-cell in vertical viscosity and diffusion, either from grid-spacing reduction effect or as artificially enhanced mixing near surface & bottom for too thin grid-cell
ALLOW_SOLVE4_PS_AND_DRAG	#undef	include code for combined surface pressure and drag implicit solver
INCLUDE_IMPLVERTADV_CODE	#define	include code for implicit vertical advection
ALLOW_ADAMSBASHFORTH_3	#undef	include code for Adams-Bashforth 3rd-order
EXACT_CONSERV	#define	include code for "exact conservation" of fluid in free-surface formulation (recompute divergence after pressure solver)
NONLIN_FRSURF	#undef	allow the use of non-linear free-surface formulation; implies that grid-cell thickness (hFactors) varies with time
ALLOW_NONHYDROSTATIC	#undef	include non-hydrostatic and 3D pressure solver codes
ALLOW_EDDYPSI	#undef	include GM-like eddy stress in momentum code (untested, not recommended)
ALLOW_CG2D_NSA	#undef	use non-self-adjoint (NSA) conjugate-gradient solver
ALLOW_SRCG	#define	include code for single reduction conjugate gradient solver
SOLVE_DIAGONAL_LOWMEMORY	#undef	low memory footprint (not suitable for AD) choice for implicit solver routines solve_*diagonal.F
SOLVE_DIAGONAL_KINNER	#undef	choice for implicit solver routines solve_*diagonal.F suitable for AD
COSINEMETH_III	#define	selects implementation form of $\cos\varphi$ scaling of bi-harmonic term for viscosity (note, CPP option for tracer diffusivity set independently in GAD_OPTIONS.h)
ISOTROPIC_COS_SCALING	#undef	selects isotropic scaling of harmonic and bi-harmonic viscous terms when using the $\cos\varphi$ scaling (note, CPP option for tracer diffusivity set independently in GAD_OPTIONS.h)

By default, MITgcm includes several core packages, i.e., these packages are enabled during genmake2 execution if a file `packages.conf` is not found. See Section 8.1.1 for more information about `packages.conf`, and see pkg/pkg_groups for more information about default packages and package groups. These default packages are as follows:

- pkg/mom_common
- pkg/mom_fluxform
- pkg/mom_vecinv
- pkg/generic_advdiff
- pkg/debug
- pkg/mdsio

- pkg/rw
- pkg/monitor

Additional CPP options that affect the model core code are set in files `${PKG}_OPTIONS.h` located in these packages' directories. Similarly, optional (non-default) packages also include package-specific CPP options that must be set in files `${PKG}_OPTIONS.h`.

The file eesupp/inc/CPP_EEOPTIONS.h does not contain any CPP options that typically will need to be modified by users.

3.8 Customizing the Model Configuration - Runtime Parameters

When you are ready to run the model in the configuration you want, the most straightforward approach is to use and adapt the setup of a tutorial or verification experiment (described in Section 4) that is the closest to your configuration. Then, the amount of setup will be minimized. In this section, we document the complete list of MITgcm model namelist runtime parameters set in file `data`, which needs to be located in the directory where you will run the model. Model parameters are defined and declared in the file PARAMS.h and their default values are generally set in the routine set_defaults.F, otherwise when initialized in the routine ini_parms.F. Section 3.8.9 documents the "execution environment" namelist parameters in file `eedata`, which must also reside in the current run directory. Note that runtime parameters used by (non-default) MITgcm packages are not documented here but rather in Section 8 and Section 9, and prescribed in package-specific `data.${pkg}` namelist files which are read in via package-specific `${pkg}_readparms.F` where `${pkg}` is the package name (see Section 8.1.1).

In what follows, model parameters are grouped into categories related to configuration/computational domain, algorithmic parameters, equations solved in the model, parameters related to model forcing, and simulation controls. The tables below specify the namelist parameter name, the namelist parameter group in `data` (and `eedata` in Section 3.8.9), the default value, and a short description of its function. Runtime parameters that require **non-default** CPP options to be set prior to compilation (see Section 3.7) for proper use are noted.

3.8.1 Parameters: Configuration, Computational Domain, Geometry, and Time-Discretization

3.8.1.1 Model Configuration

buoyancyRelation is set to `OCEANIC` by default, which employes a z-coordinate vertical axis. To simulate an ocean using pressure coordinates in the vertical, set it to `OCEANICP`. For atmospheric simulations, buoyancyRelation needs to be set to `ATMOSPHERIC`, which also uses pressure as the vertical coordinate. The default model configuration is hydrostatic; to run a non-hydrostatic simulation, set the logical variable nonHydrostatic to `.TRUE.`.

Parameter	Group	Default	Description
buoyancyRelation	PARM01	OCEANIC	buoyancy relation (`OCEANIC`, `OCEANICP`, or `ATMOSPHERIC`)
quasiHydrostatic	PARM01	FALSE	quasi-hydrostatic formulation on/off flag
rhoRefFile	PARM01	`' '`	filename for reference density profile (kg/m^3); activates anelastic form of model
nonHydrostatic	PARM01	FALSE	non-hydrostatic formulation on/off flag; requires #define ALLOW_NONHYDROSTATIC

3.8.1.2 Grid

Four different grids are available: Cartesian, spherical polar, cylindrical, and curvilinear (which includes the cubed sphere). The grid is set through the logical variables usingCartesianGrid, usingSphericalPolarGrid, usingCylindrical-

Grid, and usingCurvilinearGrid. Note that the cylindrical grid is designed for modeling a rotating tank, so that x is the azimuthual direction, y is the radial direction, and r is vertical coordinate (see tutorial *rotating tank*).

The variable xgOrigin sets the position of the western most gridcell face in the x dimension (Cartesian, meters; spherical and cyclindrical, degrees). For a Cartesian or spherical grid, the southern boundary is defined through the variable ygOrigin which corresponds to the latitude of the southern most gridcell face (Cartesian, meters; spherical, degrees). For a cyclindrical grid, a positive ygOrigin (m) adds an inner cylindrical boundary at the center of the tank. The resolution along the x and y directions is controlled by the 1D arrays delX (meters for a Cartesian grid, degrees otherwise) and delY (meters for Cartesian and cyclindrical grids, degrees spherical). On a spherical polar grid, you might decide to set the variable cosPower which is set to 0 by default and which represents n in $(\cos\varphi)^n$, the power of cosine of latitude to multiply horizontal viscosity and tracer diffusivity. The vertical grid spacing is set through the 1D array delR (z-coordinates: in meters; p-coordinates, in Pa). Using a curvilinear grid requires complete specification of all horizontal MITgcm grid variables, either through a default filename (link to new doc section) or as specified by horizGridFile.

The variable seaLev_Z represents the standard position of sea level, in meters. This is typically set to 0 m for the ocean (default value). If instead pressure is used as the vertical coordinate, the pressure at the top (of the atmosphere or ocean) is set through top_Pres, typically 0 Pa. As such, these variables are analogous to xgOrigin and ygOrigin to define the vertical grid axis. But they also are used for a second purpose: in a z-coordinate setup, top_Pres sets a reference top pressure (required in a non-linear equation of state computation, for example); note that 1 bar (i.e., typical Earth atmospheric sea-level pressure) is added already, so the default is 0 Pa. Similarly, for a p-coordinate setup, seaLev_Z is used to set a reference geopotential (after gravity scaling) at the top of the ocean or bottom of the atmosphere.

Parameter	Group	Default	Description
usingCartesianGrid	PARM04	TRUE	use Cartesian grid/coordinates on/off flag
usingSphericalPolarGrid	PARM04	FALSE	use spherical grid/coordinates on/off flag
usingCylindricalGrid	PARM04	FALSE	use cylindrical grid/coordinates on/off flag
usingCurvilinearGrid	PARM04	FALSE	use curvilinear grid/coordinates on/off flag
xgOrigin	PARM04	0.0	west edge x-axis origin (Cartesian: m; spherical and cyclindrical: degrees longitude)
ygOrigin	PARM04	0.0	South edge y-axis origin (Cartesian and cyclindrical: m; spherical: degrees latitude)
dxSpacing	PARM04	unset	x-axis uniform grid spacing, separation between cell faces (Cartesian: m; spherical and cyclindrical: degrees)
delX	PARM04	dxSpacing	1D array of x-axis grid spacing, separation between cell faces (Cartesian: m; spherical and cyclindrical: degrees)
delXFile	PARM04	' '	filename containing 1D array of x-axis grid spacing
dySpacing	PARM04	unset	y-axis uniform grid spacing, separation between cell faces (Cartesian and cyclindrical: m; spherical: degrees)
delY	PARM04	dySpacing	1D array of x-axis grid spacing, separation between cell faces (Cartesian and cyclindrical: m; spherical: degrees)
delYFile	PARM04	' '	filename containing 1D array of y-axis grid spacing
cosPower	PARM01	0.0	power law n in $(\cos\varphi)^n$ factor for horizontal (harmonic or biharmonic) viscosity and tracer diffusivity (spherical polar)
delR	PARM04	computed using delRc	vertical grid spacing 1D array ([r] unit)
delRc	PARM04	computed using delR	vertical cell center spacing 1D array ([r] unit)
delRFile	PARM04	' '	filename for vertical grid spacing 1D array ([r] unit)
delRcFile	PARM04	' '	filename for vertical cell center spacing 1D array ([r] unit)

Continued on next page

Table 3.2 – continued from previous page

Parameter	Group	Default	Description
rSphere	PARM04	6.37E+06	radius of sphere for spherical polar or curvilinear grid (m)
seaLev_Z	PARM04	0.0	reference height of sea level (m)
top_Pres	PARM04	0.0	top pressure (p-coordinates) or top reference pressure (z-coordinates) (Pa)
selectFindRoSurf	PARM01	0	select method to determine surface reference pressure from orography (atmos.-only)
horizGridFile	PARM04	' '	filename containing full set of horizontal grid variables (curvilinear)
radius_fromHorizGrid	PARM04	rSphere	radius of sphere used in input curvilinear horizontal grid file (m)
phiEuler	PARM04	0.0	Euler angle, rotation about original z-axis (spherical polar) (degrees)
thetaEuler	PARM04	0.0	Euler angle, rotation about new x-axis (spherical polar) (degrees)
psiEuler	PARM04	0.0	Euler angle, rotation about new z-axis (spherical polar) (degrees)

3.8.1.3 Topography - Full and Partial Cells

For the ocean, the topography is read from a file that contains a 2D(x, y) map of bathymetry, in meters for z-coordinates, in pascals for p-coordinates. The bathymetry is specified by entering the vertical position of the ocean floor relative to the surface, so by convention in z-coordinates bathymetry is specified as negative numbers ("depth" is defined as positive-definite) whereas in p-coordinates bathymetry data is positive. The file name is represented by the variable bathyFile. See our introductory tutorial setup Section 4.1 for additional details on the file format. Note no changes are required in the model source code to represent enclosed, periodic, or double periodic domains: periodicity is assumed by default and is suppressed by setting the depths to 0 m for the cells at the limits of the computational domain.

To use the partial cell capability, the variable hFacMin needs to be set to a value between 0.0 and 1.0 (it is set to 1.0 by default) corresponding to the minimum fractional size of a gridcell. For example, if a gridcell is 500 m thick and hFacMin is set to 0.1, the minimum thickness for a "thin-cell" for this specific gridcell is 50 m. Thus, if the specified bathymetry depth were to fall exactly in the middle of this 500m thick gridcell, the initial model variable hFacC(x, y, r) would be set to 0.5. If the specified bathymetry depth fell within the top 50m of this gridcell (i.e., less than hFacMin), the model bathymetry would snap to the nearest legal value (i.e., initial hFacC(x, y, r) would be equal to 0.0 or 0.1 depending if the depth was within 0-25 m or 25-50 m, respectively). Also note while specified bathymetry bottom depths (or pressures) need not coincide with the model's levels as deduced from delR, any depth falling below the model's defined vertical axis is truncated.

Parameter	Group	Default	Description
bathyFile	PARM05	' '	filename for 2D bathymetry (ocean) (z-coor.: m, negative; p-coor.: Pa, positive)
topoFile	PARM05	' '	filename for 2D surface topography (atmosphere) (m)
addWwallFile	PARM05	' '	filename for 2D western cell-edge "thin-wall"
addSwallFile	PARM05	' '	filename for 2D southern cell-edge "thin-wall"
hFacMin	PARM01	1.0E+00	minimum fraction size of a cell
hFacMinDr	PARM01	1.0E+00	minimum dimensional size of a cell ([r] unit)
hFacInf	PARM01	2.0E-01	lower threshold fraction for surface cell; for non-linear free surface only, see parameter *nonlinFreeSurf*
hFacSup	PARM01	2.0E+00	upper threshold fraction for surface cell; for non-linear free surface, only see parameter *nonlinFreeSurf*

Continued on next page

Table 3.3 – continued from previous page

Parameter	Group	Default	Description
useMin4hFacEdges	PARM04	FALSE	set hFacW, hFacS as minimum of adjacent hFacC on/off flag
pCellMix_select	PARM04	0	option/factor to enhance mixing at the surface or bottom (0- 99)
pCellMix_maxFac	PARM04	1.0E+04	maximum enhanced mixing factor for too thin partial-cell (non-dim.)
pCellMix_delR	PARM04	0.0	thickness criteria for too thin partial-cell ([r] unit)

3.8.1.4 Physical Constants

Parameter	Group	Default	Description
rhoConst	PARM01	rhoNil	vertically constant reference density (Boussinesq) (kg/m^3)
gravity	PARM01	9.81E+00	gravitational acceleration (m/s^2)
gravityFile	PARM01	' '	filename for 1D gravity vertical profile (m/s^2)
gBaro	PARM01	gravity	gravity constant in barotropic equation (m/s^2)

3.8.1.5 Rotation

For a Cartesian or cylindical grid, the Coriolis parameter f is set through the variables f0 (in s^{-1}) and beta ($\frac{\partial f}{\partial y}$; in m^{-1}s^{-1}), which corresponds to a Coriolis parameter $f = f_o + \beta y$ (the so-called β-plane).

Parameter	Group	Default	Description
rotationPeriod	PARM01	8.6164E+04	rotation period (s)
omega	PARM01	$2\pi/$rotationPeriod	angular velocity (rad/s)
selectCoriMap	PARM01	depends on grid (Cartesian and cylindrical=1, spherical and curvilinear=2)	Coriolis map options • 0: f-plane • 1: beta-plane • 2: spherical Coriolis ($= 2\Omega \sin\varphi$) • 3: read 2D field from file
f0	PARM01	1.0E-04	reference Coriolis parameter (Cartesian or cylindrical grid) (1/s)
beta	PARM01	1.0E-11	β (Cartesian or cylindrical grid) (m^{-1}s^{-1})
fPrime	PARM01	0.0	$2\Omega\cos\phi$ parameter (Cartesian or cylindical grid) (1/s); i.e., for $\cos\varphi$ Coriolis terms from horizontal component of rotation vector (also sometimes referred to as reciprocal Coriolis parm.)

3.8.1.6 Free Surface

The logical variables rigidLid and implicitFreeSurface specify your choice for ocean upper boundary (or lower boundary if using p-coordinates); set one to `.TRUE.` and the other to `.FALSE.`. These settings affect the calculations of surface pressure (for the ocean) or surface geopotential (for the atmosphere); see Section 3.8.2.

Parameter	Group	Default	Description
implicitFreeSurface	PARM01	TRUE	implicit free surface on/off flag
rigidLid	PARM01	FALSE	rigid lid on/off flag

Continued on next page

Table 3.4 – continued from previous page

Parameter	Group	Default	Description
useRealFreshWaterFlux	PARM01	FALSE	use true E-P-R freshwater flux (changes free surface/sea level) on/off flag
implicSurfPress	PARM01	1.0E+00	implicit fraction of the surface pressure gradient (0-1)
implicDiv2Dflow	PARM01	1.0E+00	implicit fraction of the barotropic flow divergence (0-1)
implicitNHPress	PARM01	implicSurfPress	implicit fraction of the non-hydrostatic pressure gradient (0-1); for non-hydrostatic only, see parameter *nonHydrostatic*
nonlinFreeSurf	PARM01	0	non-linear free surface options (-1,0,1,2,3; see Table 2.1); requires #define NONLIN_FRSURF
select_rStar	PARM01	0	vertical coordinate option • 0: use r • >0: use r^* see Table 2.1; requires #define NONLIN_FRSURF
selectNHfreeSurf	PARM01	0	non-hydrostatic free surface formulation option • 0: don't use • >0: use requires non-hydrostatic formulation, see parameter *nonHydrostatic*
exactConserv	PARM01	FALSE	exact total volume conservation (recompute divergence after pressure solver) on/off flag

3.8.1.7 Time-Discretization

The time steps are set through the real variables deltaTMom and deltaTtracer (in seconds) which represent the time step for the momentum and tracer equations, respectively (or you can prescribe a single time step value for all parameters using deltaT). The model "clock" is defined by the variable deltaTClock (in seconds) which determines the I/O frequencies and is used in tagging output. Time in the model is thus computed as:

model time = baseTime + iteration number * deltaTClock

Parameter	Group	Default	Description
deltaT	PARM03	0.0	default value used for model time step parameters (s)
deltaTClock	PARM03	deltaT	timestep used for model clock (s): used for I/O frequency and tagging output and checkpoints
deltaTmom	PARM03	deltaT	momentum equation timestep (s)
deltaTtracer	PARM03	deltaT	tracer equation timestep (s)
dTtracerLev	PARM03	deltaTtracer	tracer equation timestep specified at each vertical level (s)
deltaTfreesurf	PARM03	deltaTmom	free-surface equation timestep (s)
baseTime	PARM03	0.0	model base time corresponding to iteration 0 (s)

3.8.2 Parameters: Main Algorithmic Parameters

3.8.2.1 Pressure Solver

By default, a hydrostatic simulation is assumed and a 2D elliptic equation is used to invert the pressure field. If using a non-hydrostatic configuration, the pressure field is inverted through a 3D elliptic equation (note this capability is

not yet available for the atmosphere). The parameters controlling the behavior of the elliptic solvers are the variables cg2dMaxIters and cg2dTargetResidual for the 2D case and cg3dMaxIters and cg3dTargetResidual for the 3D case.

Parameter	Group	Default	Description
cg2dMaxIters	PARM02	150	upper limit on 2D conjugate gradient solver iterations
cg2dTargetResidual	PARM02	1.0E-07	2D conjugate gradient target residual (non-dim. due to RHS normalization)
cg2dTargetResWunit	PARM02	-1.0E+00	2D conjugate gradient target residual ($\dot{r}$ units); <0: use RHS normalization, i.e., cg2dTargetResidual instead
cg2dPreCondFreq	PARM02	1	frequency (in number of iterations) for updating cg2d preconditioner; for non-linear free surface only, see parameter *nonlinFreeSurf*
cg2dUseMinResSol	PARM02	0 unless flat-bottom, Cartesian	• 0: use last-iteration/converged cg2d solution • 1: use solver minimum-residual solution
cg3dMaxIters	PARM02	150	upper limit on 3D conjugate gradient solver iterations; requires #define ALLOW_NONHYDROSTATIC
cg3dTargetResidual	PARM02	1.0E-07	3D conjugate gradient target residual (non-dim. due to RHS normalization); requires #define ALLOW_NONHYDROSTATIC
useSRCGSolver	PARM02	FALSE	use conjugate gradient solver with single reduction (single call of mpi_allreduce)
printResidualFreq	PARM02	1 unless debugLevel >4	frequency (in number of iterations) of printing conjugate gradient residual
integr_GeoPot	PARM01	2	select method to integrate geopotential • 1: finite volume • $\neq$1: finite difference
uniformLin_PhiSurf	PARM01	TRUE	use uniform b_s relation for ϕ_s on/off flag
deepAtmosphere	PARM04	FALSE	don't make the thin shell/shallow water approximation
nh_Am2	PARM01	1.0E+00	non-hydrostatic terms scaling factor; requires #define ALLOW_NONHYDROSTATIC

3.8.2.2 Time-Stepping Algorithm

The Adams-Bashforth stabilizing parameter is set through the variable abEps (dimensionless). The stagger baroclinic time stepping algorithm can be activated by setting the logical variable staggerTimeStep to `.TRUE.`.

Parameter	Group	Default	Description
abEps	PARM03	1.0E-02	Adams-Bashforth-2 stabilizing weight (non-dim.)
alph_AB	PARM03	0.5E+00	Adams-Bashforth-3 primary factor (non-dim.); requires #define ALLOW_ADAMSBASHFORTH_3
beta_AB	PARM03	5/12	Adams-Bashforth-3 secondary factor (non-dim.); requires #define ALLOW_ADAMSBASHFORTH_3
staggerTimeStep	PARM01	FALSE	use staggered time stepping (thermodynamic vs. flow variables) on/off flag
multiDimAdvection	PARM01	TRUE	use multi-dim. advection algorithm in schemes where non multi-dim. is possible on/off flag
implicitIntGravWave	PARM01	FALSE	treat internal gravity waves implicitly on/off flag; requires #define ALLOW_NONHYDROSTATIC

3.8.3 Parameters: Equation of State

The form of the equation of state is controlled by the model configuration and eosType.

For the atmosphere, eosType must be set to `IDEALGAS`.

For the ocean, several forms of the equation of state are available:

- For a linear approximation, set eosType to `LINEAR`), and you will need to specify the thermal and haline expansion coefficients, represented by the variables tAlpha (in K^{-1}) and sBeta (in psu^{-1}). Because the model equations are written in terms of perturbations, a reference thermodynamic state needs to be specified. This is done through the 1D arrays tRef and sRef. tRef specifies the reference potential temperature profile (in ^oC for the ocean and K for the atmosphere) starting from the level k=1. Similarly, sRef specifies the reference salinity profile (in psu or g/kg) for the ocean or the reference specific humidity profile (in g/kg) for the atmosphere.
- MITgcm offers several approximations to the full (oceanic) non-linear equation of state that can be selected as eosType:

  `'POLYNOMIAL':` This approximation is based on the Knudsen formula (see Bryan and Cox 1972 [BC72]). For this option you need to generate a file of polynomial coefficients called `POLY3.COEFFS`. To do this, use the program utils/knudsen2/knudsen2.f under the model tree (a `Makefile` is available in the same directory; you will need to edit the number and the values of the vertical levels in knudsen2.f so that they match those of your configuration).

  `'UNESCO':` The UNESCO equation of state formula (IES80) of Fofonoff and Millard (1983) [FRM83]. This equation of state assumes in-situ temperature, which is not a model variable; **its use is therefore discouraged**.

  `'JMD95Z':` A modified UNESCO formula by Jackett and McDougall (1995) [JM95], which uses the model variable potential temperature as input. The 'Z' indicates that this equation of state uses a horizontally and temporally constant pressure $p_0 = -g\rho_0 z$.

  `'JMD95P':` A modified UNESCO formula by Jackett and McDougall (1995) [JM95], which uses the model variable potential temperature as input. The 'P' indicates that this equation of state uses the actual hydrostatic pressure of the last time step. Lagging the pressure in this way requires an additional pickup file for restarts.

  `'MDJWF':` A more accurate and less expensive equation of state than UNESCO by McDougall et al. (2003) [MJWF03], also using the model variable potential temperature as input. It also requires lagging the pressure and therefore an additional pickup file for restarts.

  `'TEOS10':` TEOS-10 is based on a Gibbs function formulation from which all thermodynamic properties of seawater (density, enthalpy, entropy sound speed, etc.) can be derived in a thermodynamically consistent manner; see http://www.teos-10.org. See IOC et al. (2010) [ISI10], McDougall and Parker (2011) [MB11], and Roquet et al. (2015) [RMMB15] for implementation details. It also requires lagging the pressure and therefore an additional pickup file for restarts. Note at this time a full implementation of TEOS10 (i.e. ocean variables of conservative temperature and practical salinity, including consideration of surface forcings) has not been implemented; also note the original 48-term polynomial term is used, not the newer, preferred 75-term polynomial.

  For these non-linear approximations, neither a reference profile of temperature or salinity is required, except for a setup where implicitIntGravWave is set to `.TRUE.` or selectP_inEOS_Zc=1.

Note that salinity can can be expressed in either practical salinity units (psu, i.e., unit-less) or g/kg, depending on the choice of equation of state. See Millero (2010) [Mil10] for a detailed discussion of salinity measurements, and why use of the latter is preferred, in the context of the ocean equation of state.

Parameter	Group	Default	Description
eosType	PARM01	LINEAR	equation of state form
tRef	PARM01	20.0 °C (ocn) or 300.0 K (atm)	1D vertical reference temperature profile (°C or K)
tRefFile	PARM01	' '	filename for reference temperature profile (°C or K)
thetaConst	PARM01	tRef(k=1)	vertically constant reference temp. for atmosphere p^* coordinates (°K); for ocean, specify instead of tRef or tRefFile for vertically constant reference temp. (°C)
sRef	PARM01	30.0 psu (ocn) or 0.0 (atm)	1D vertical reference salinity profile (psu or g/kg)
sRefFile	PARM01	' '	filename for reference salinity profile (psu or g/kg)
selectP_inEOS_Zc	PARM01	depends on eosType	select which pressure to use in EOS for z-coor. • 0: use $-g\rho_c z$ • 1: use $p_{ref} = -\int -g\rho(T_{ref}, S_{ref}, p_{ref})dz$ • 2: hydrostatic dynamical pressure • 3: use full hyd.+non-hyd. pressure for `JMD95P`, `UNESCO`, `MDJWF`, `TEOS10` default=2, otherwise default =0
rhonil	PARM01	9.998E+02	reference density for linear EOS (kg/m^3)
tAlpha	PARM01	2.0E-04	linear EOS thermal expansion coefficient (1/°C)
sBeta	PARM01	7.4E-04	linear EOS haline contraction coefficient (1/psu)

3.8.3.1 Thermodynamic Constants

Parameter	Group	Default	Description
HeatCapacity_Cp	PARM01	3.994E+03	specific heat capacity C_p (ocean) (J/kg/K)
celsius2K	PARM01	2.7315E+02	conversion constant °C to Kelvin
atm_Cp	PARM01	1.004E+03	specific heat capacity C_p dry air at const. press. (J/kg/K)
atm_Rd	PARM01	atm_Cp*(2/7)	gas constant for dry air (J/kg/K)
atm_Rq	PARM01	0.0	water vapor specific volume anomaly relative to dry air (g/kg)
atm_Po	PARM01	1.0E+05	atmosphere standard reference pressure (for potential temp. defn.) (Pa)

3.8.4 Parameters: Momentum Equations

3.8.4.1 Configuration

There are a few logical variables that allow you to turn on/off various terms in the momentum equation. These variables are called momViscosity, momAdvection, useCoriolis, momStepping, metricTerms, and momPressureForcing and by default are set to `.TRUE.`. Vertical diffusive fluxes of momentum can be computed implicitly by setting the logical variable implicitViscosity to `.TRUE.`. The details relevant to both the momentum flux-form and the vector-invariant form of the equations and the various (momentum) advection schemes are covered in Section 2.

Parameter	Group	Default	Description
momStepping	PARM01	TRUE	momentum equation time-stepping on/off flag
momViscosity	PARM01	TRUE	momentum friction terms on/off flag
momAdvection	PARM01	TRUE	advection of momentum on/off flag
momPressureForcing	PARM01	TRUE	pressure term in momentum equation on/off flag

Continued on next page

Table 3.7 – continued from previous page

Parameter	Group	Default	Description
metricTerms	PARM01	TRUE	include metric terms (spherical polar, momentum flux-form) on/off flag
useNHMTerms	PARM01	FALSE	use "non-hydrostatic form" of metric terms on/off flag; (see Section 2.14.4; note these terms are non-zero in many model configurations beside non-hydrostatic)
momImplVertAdv	PARM01	FALSE	momentum implicit vertical advection on/off flag; requires #define INCLUDE_IMPLVERTADV_CODE
implicitViscosity	PARM01	FALSE	implicit vertical viscosity on/off flag
interViscAr_pCell	PARM04	FALSE	account for partial-cell in interior vertical viscosity on/off flag
momDissip_In_AB	PARM03	TRUE	use Adams-Bashforth time stepping for dissipation tendency
useCoriolis	PARM01	TRUE	include Coriolis terms on/off flag
use3dCoriolis	PARM01	TRUE	include $\cos\varphi$ Coriolis terms on/off flag
selectCoriScheme	PARM01	0	Coriolis scheme selector • 0: original scheme • 1: wet-point averaging method • 2: Flux-Form: energy conserving; Vector-Inv: hFac weighted average • 3: Flux-Form: energy conserving using wet-point method; Vector-Inv: energy conserving with hFac weight
vectorInvariantMomentum	PARM01	FALSE	use vector-invariant form of momentum equations flag
useJamartMomAdv	PARM01	FALSE	use Jamart wetpoints method for relative vorticity advection (vector invariant form) on/off flag
selectVortScheme	PARM01	1	vorticity scheme (vector invariant form) options • 0,1: enstrophy conserving forms • 2: energy conserving form • 3: energy and enstrophy conserving form see Sadourny 1975 [Sad75] and Burridge & Haseler 1977 [BH77]
upwindVorticity	PARM01	FALSE	bias interpolation of vorticity in the Coriolis term (vector invariant form) on/off flag
useAbsVorticity	PARM01	FALSE	use $f + \zeta$ in Coriolis terms (vector invariant form) on/off flag
highOrderVorticity	PARM01	FALSE	use 3rd/4th order interpolation of vorticity (vector invariant form) on/off flag
upwindShear	PARM01	FALSE	use 1st order upwind for vertical advection (vector invariant form) on/off flag
selectKEscheme	PARM01	0	kinetic energy computation in Bernoulli function (vector invariant form) options • 0: standard form • 1: area-weighted standard form • 2: as 0 but account for partial cells • 3: as 1 w/partial cells see mom_calc_ke.F

3.8.4.2 Initialization

The initial horizontal velocity components can be specified from binary files uVelInitFile and vVelInitFile. These files should contain 3D data ordered in an (x, y, r) fashion with k=1 as the first vertical level (surface level). If no file names are provided, the velocity is initialized to zero. The initial vertical velocity is always derived from the horizontal velocity using the continuity equation. In the case of a restart (from the end of a previous simulation), the velocity field is read from a pickup file (see Section 3.8.7) and the initial velocity files are ignored.

Parameter	Group	Default	Description
uVelInitFile	PARM05	' '	filename for 3D specification of initial zonal velocity field (m/s)
vVelInitFile	PARM05	' '	filename for 3D specification of initial meridional velocity field (m/s)
pSurfInitFile	PARM05	' '	filename for 2D specification of initial free surface position ([r] unit)

3.8.4.3 General Dissipation Scheme

The lateral eddy viscosity coefficient is specified through the variable viscAh (in m^2s^{-1}). The vertical eddy viscosity coefficient is specified through the variable viscAr (in $[r]^2s^{-1}$, where $[r]$ is the dimension of the vertical coordinate). In addition, biharmonic mixing can be added as well through the variable viscA4 (in m^4s^{-1}).

Parameter	Group	Default	Description
viscAh	PARM01	0.0	lateral eddy viscosity (m^2/s)
viscAhD	PARM01	viscAh	lateral eddy viscosity acts on divergence part (m^2/s)
viscAhZ	PARM01	viscAh	lateral eddy viscosity acts on vorticity part (ζ points) (m^2/s)
viscAhW	PARM01	viscAhD	lateral eddy viscosity for mixing vertical momentum (non-hydrostatic form) (m^2/s); for non-hydrostatic only, see parameter *nonHydrostatic*
viscAhDfile	PARM05	' '	filename for 3D specification of lateral eddy viscosity (divergence part) (m^2/s); requires #define ALLOW_3D_VISCAH in pkg/mom_common/MOM_COMMON_OPTIONS.h
viscAhZfile	PARM05	' '	filename for 3D specification of lateral eddy viscosity (vorticity part, ζ points); requires #define ALLOW_3D_VISCAH in pkg/mom_common/MOM_COMMON_OPTIONS.h
viscAhGrid	PARM01	0.0	grid-dependent lateral eddy viscosity (non-dim.)
viscAhMax	PARM01	1.0E+21	maximum lateral eddy viscosity (m^2/s)
viscAhGridMax	PARM01	1.0E+21	maximum lateral eddy (grid-dependent) viscosity (non-dim.)
viscAhGridMin	PARM01	0.0	minimum lateral eddy (grid-dependent) viscosity (non-dim.)
viscAhReMax	PARM01	0.0	minimum lateral eddy viscosity based on Reynolds number (non-dim.)
viscC2leith	PARM01	0.0	Leith harmonic viscosity factor (vorticity part, ζ points) (non-dim.)
viscC2leithD	PARM01	0.0	Leith harmonic viscosity factor (divergence part) (non-dim.)
viscC2LeithQG	PARM01	0.0	Quasi-geostrophic Leith viscosity factor (non-dim.)
viscC2smag	PARM01	0.0	Smagorinsky harmonic viscosity factor (non-dim.)
viscA4	PARM01	0.0	lateral biharmonic viscosity (m^4/s)
viscA4D	PARM01	viscA4	lateral biharmonic viscosity (divergence part) (m^4/s)
viscA4Z	PARM01	viscA4	lateral biharmonic viscosity (vorticity part, ζ points) (m^4/s)
viscA4W	PARM01	viscA4D	lateral biharmonic viscosity for mixing vertical momentum (non-hydrostatic form) (m^4/s); for non-hydrostatic only, see parameter *nonHydrostatic*

Continued on next page

Table 3.8 – continued from previous page

Parameter	Group	Default	Description
viscA4Dfile	PARM05	' '	filename for 3D specification of lateral biharmonic viscosity (divergence part) (m^4/s); requires #define ALLOW_3D_VISCA4 in pkg/mom_common/MOM_COMMON_OPTIONS.h
viscA4Zfile	PARM05	' '	filename for 3D specification of lateral biharmonic viscosity (vorticity part, ζ points); requires #define ALLOW_3D_VISCA4 in pkg/mom_common/MOM_COMMON_OPTIONS.h
viscA4Grid	PARM01	0.0	grid dependent biharmonic viscosity (non-dim.)
viscA4Max	PARM01	1.0E+21	maximum biharmonic viscosity (m^4/s)
viscA4GridMax	PARM01	1.0E+21	maximum biharmonic (grid-dependent) viscosity (non-dim.)
viscA4GridMin	PARM01	0.0	minimum biharmonic (grid-dependent) viscosity (mon-dim.)
viscA4ReMax	PARM01	0.0	minimum biharmonic viscosity based on Reynolds number (non-dim.)
viscC4leith	PARM01	0.0	Leith biharmonic viscosity factor (vorticity part, ζ points) (non-dim.)
viscC4leithD	PARM01	0.0	Leith biharmonic viscosity factor (divergence part) (non-dim.)
viscC4smag	PARM01	0.0	Smagorinsky biharmonic viscosity factor (non-dim.)
useFullLeith	PARM01	FALSE	use full form of Leith viscosities on/off flag
useSmag3D	PARM01	FALSE	use isotropic 3D Smagorinsky harmonic viscosities flag; requires #define ALLOW_SMAG_3D in pkg/mom_common/MOM_COMMON_OPTIONS.h
smag3D_coeff	PARM01	1.0E-02	isotropic 3D Smagorinsky coefficient (non-dim.); requires #define ALLOW_SMAG_3D in pkg/mom_common/MOM_COMMON_OPTIONS.h
useStrainTensionVisc	PARM01	FALSE	flag to use strain-tension form of viscous operator
useAreaViscLength	PARM01	FALSE	flag to use area for viscous L^2 instead of harmonic mean of ${L_x}^2, {L_y}^2$
viscAr	PARM01	0.0	vertical eddy viscosity ($[r]^2$/s)
viscArNr	PARM01	0.0	vertical profile of vertical eddy viscosity ($[r]^2$/s)
pCellMix_viscAr	PARM04	viscArNr	vertical viscosity for too thin partial-cell ($[r]^2$/s)

3.8.4.4 Sidewall/Bottom Dissipation

Slip or no-slip conditions at lateral and bottom boundaries are specified through the logical variables no_slip_sides and no_slip_bottom. If set to `.FALSE.`, free-slip boundary conditions are applied. If no-slip boundary conditions are applied at the bottom, a bottom drag can be applied as well. Two forms are available: linear (set the variable bottomDragLinear in $[r]$/s,) and quadratic (set the variable bottomDragQuadratic, $[r]$/m).

Parameter	Group	Default	Description
no_slip_sides	PARM01	TRUE	viscous BCs: no-slip sides on/off flag
sideDragFactor	PARM01	2.0E+00	side-drag scaling factor (2.0: full drag) (non-dim.)
no_slip_bottom	PARM01	TRUE	viscous BCs: no-slip bottom on/off flag
bottomDragLinear	PARM01	0.0	linear bottom-drag coefficient ([r]/s)
bottomDragQuadratic	PARM01	0.0	quadratic bottom-drag coefficient ([r]/m)
selectBotDragQuadr	PARM01	-1	select quadratic bottom drag discretization option • -1: not used • 0: average KE from grid center to u, v location • 1: use local velocity norm @ u, v location • 2: as 1 with wet-point averaging of other velocity component if bottomDragQuadratic $\neq 0$. then default is 0
selectImplicitDrag	PARM01	0	top/bottom drag implicit treatment options • 0: fully explicit • 1: implicit on provisional velocity, i.e., before $\nabla\eta$ increment • 2: fully implicit if =2, requires #define ALLOW_SOLVE4_PS_AND_DRAG
bottomVisc_pCell	PARM01	FALSE	account for partial-cell in bottom viscosity (using no_slip_bottom = `.TRUE.`) on/off flag

3.8.5 Parameters: Tracer Equations

This section covers the tracer equations, i.e., the potential temperature equation and the salinity (for the ocean) or specific humidity (for the atmosphere) equation.

3.8.5.1 Configuration

The logical variables tempAdvection, and tempStepping allow you to turn on/off terms in the temperature equation (similarly for salinity or specific humidity with variables saltAdvection etc.). These variables all default to a value of `.TRUE.`. The vertical diffusive fluxes can be computed implicitly by setting the logical variable implicitDiffusion to `.TRUE.`.

Parameter	Group	Default	Description
tempStepping	PARM01	TRUE	temperature equation time-stepping on/off flag
tempAdvection	PARM01	TRUE	advection of temperature on/off flag
tempAdvScheme	PARM01	2	temperature horizontal advection scheme selector (see Table 2.2)
tempVertAdvScheme	PARM01	tempAdvScheme	temperature vertical advection scheme selector (see Table 2.2)
tempImplVertAdv	PARM01	FALSE	temperature implicit vertical advection on/off flag
addFrictionHeating	PARM01	FALSE	include frictional heating in temperature equation on/off flag; requires #define ALLOW_FRICTION_HEATING
temp_stayPositive	PARM01	FALSE	use Smolarkiewicz hack to ensure temperature stays positive on/off flag; requires #define GAD_SMOLARKIEWICZ_HACK in pkg/generic_advdiff/GAD_OPTIONS.h
saltStepping	PARM01	TRUE	salinity equation time-stepping on/off flag

Continued on next page

Table 3.9 – continued from previous page

Parameter	Group	Default	Description
saltAdvection	PARM01	TRUE	advection of salinity on/off flag
saltAdvScheme	PARM01	2	salinity horizontal advection scheme selector (see Table 2.2)
saltVertAdvScheme	PARM01	saltAdvScheme	salinity vertical advection scheme selector (see Table 2.2)
saltImplVertAdv	PARM01	FALSE	salinity implicit vertical advection on/off flag
salt_stayPositive	PARM01	FALSE	use Smolarkiewicz hack to ensure salinity stays positive on/off flag; requires #define GAD_SMOLARKIEWICZ_HACK in pkg/generic_advdiff/GAD_OPTIONS.h
implicitDiffusion	PARM01	FALSE	implicit vertical diffusion on/off flag
interDiffKr_pCell	PARM04	FALSE	account for partial-cell in interior vertical diffusion on/off flag
linFSConserveTr	PARM01	TRUE	correct source/sink of tracer due to use of linear free surface on/off flag
doAB_onGtGs	PARM03	TRUE	apply Adams-Bashforth on tendencies (rather than on T,S) on/off flag

3.8.5.2 Initialization

The initial tracer data can be contained in the binary files hydrogThetaFile and hydrogSaltFile. These files should contain 3D data ordered in an (x, y, r) fashion with k=1 as the first vertical level. If no file names are provided, the tracers are then initialized with the values of tRef and sRef discussed in Section 3.8.3. In this case, the initial tracer data are uniform in x and y for each depth level.

Parameter	Group	Default	Description
hydrogThetaFile	PARM05	' '	filename for 3D specification of initial potential temperature (^{o}C)
hydrogSaltFile	PARM05	' '	filename for 3D specification of initial salinity (psu or g/kg)
maskIniTemp	PARM05	TRUE	apply (center-point) mask to initial hydrographic theta data on/off flag
maskIniSalt	PARM05	TRUE	apply (center-point) mask to initial hydrographic salinity on/off flag
checkIniTemp	PARM05	TRUE	check if initial theta (at wet-point) identically zero on/off flag
checkIniSalt	PARM05	TRUE	check if initial salinity (at wet-point) identically zero on/off flag

3.8.5.3 Tracer Diffusivities

Lateral eddy diffusivities for temperature and salinity/specific humidity are specified through the variables diffKhT and diffKhS (in m^2/s). Vertical eddy diffusivities are specified through the variables diffKrT and diffKrS.In addition, biharmonic diffusivities can be specified as well through the coefficients diffK4T and diffK4S (in m^4/s). The Gent and McWilliams parameterization for advection and mixing of oceanic tracers is described in Section 8.4.1.

Parameter	Group	Default	Description
diffKhT	PARM01	0.0	Laplacian diffusivity of heat laterally (m^2/s)
diffK4T	PARM01	0.0	biharmonic diffusivity of heat laterally (m^4/s)
diffKrT	PARM01	0.0	Laplacian diffusivity of heat vertically (m^2/s)
diffKr4T	PARM01	0.0	biharmonic diffusivity of heat vertically (m^2/s)
diffKrNrT	PARM01	0.0 at k=top	vertical profile of vertical diffusivity of temperature (m^2/s)
pCellMix_diffKr	PARM04	diffKrNr	vertical diffusivity for too thin partial-cell ($[r]^2/s$)

Continued on next page

Table 3.10 – continued from previous page

Parameter	Group	Default	Description
diffKhS	PARM01	0.0	Laplacian diffusivity of salt laterally (m^2/s)
diffK4S	PARM01	0.0	biharmonic diffusivity of salt laterally (m^4/s)
diffKrS	PARM01	0.0	Laplacian diffusivity of salt vertically (m^2/s)
diffKr4S	PARM01	0.0	biharmonic diffusivity of salt vertically (m^2/s)
diffKrNrS	PARM01	0.0 at k=top	vertical profile of vertical diffusivity of salt (m^2/s)
diffKrFile	PARM05	' '	filename for 3D specification of vertical diffusivity (m^2/s); requires #define ALLOW_3D_DIFFKR
diffKrBL79surf	PARM01	0.0	surface diffusivity for Bryan & Lewis 1979 [BL79] (m^2/s)
diffKrBL79deep	PARM01	0.0	deep diffusivity for Bryan & Lewis 1979 [BL79] (m^2/s)
diffKrBL79scl	PARM01	2.0E+02	depth scale for Bryan & Lewis 1979 [BL79] (m)
diffKrBL79Ho	PARM01	-2.0E+03	turning depth for Bryan & Lewis 1979 [BL79] (m)
diffKrBLEQsurf	PARM01	0.0	same as diffKrBL79surf but at equator; requires #define ALLOW_BL79_LAT_VARY
diffKrBLEQdeep	PARM01	0.0	same as diffKrBL79deep but at equator; requires #define ALLOW_BL79_LAT_VARY
diffKrBLEQscl	PARM01	2.0E+02	same as diffKrBL79scl but at equator; requires #define ALLOW_BL79_LAT_VARY
diffKrBLEQHo	PARM01	-2.0E+03	same as diffKrBL79Ho but at equator; requires #define ALLOW_BL79_LAT_VARY
BL79LatVary	PARM01	3.0E+01	transition from diffKrBLEQ to diffKrBL79 parms at this latitude; requires #define ALLOW_BL79_LAT_VARY

3.8.5.4 Ocean Convection

In addition to specific packages that parameterize ocean convection, two main model options are available. To use the first option, a convective adjustment scheme, you need to set the variable cadjFreq, the frequency (in seconds) with which the adjustment algorithm is called, to a non-zero value (note, if cadjFreq set to a negative value by the user, the model will set it to the model clock time step). The second option is to parameterize convection with implicit vertical diffusion. To do this, set the logical variable implicitDiffusion to `.TRUE.` and the real variable ivdc_kappa (in m^2/s) to an appropriate tracer vertical diffusivity value for mixing due to static instabilities (typically, several orders of magnitude above the background vertical diffusivity). Note that cadjFreq and ivdc_kappa cannot both have non-zero value.

Parameter	Group	Default	Description
ivdc_kappa	PARM01	0.0	implicit vertical diffusivity for convection (m^2/s)
cAdjFreq	PARM03	0	frequency of convective adj. scheme; <0: sets value to deltaTclock (s)
hMixCriteria	PARM01	-0.8E+00	• <0: specifies ΔT (°C) to define ML depth where $\Delta\rho = \Delta T * d\rho/dT$ occurs • >1: define ML depth where local strat. exceeds mean strat. by this factor (non-dim.)
hMixSmooth	PARM01	0.0	use this fraction of neighboring points (for smoothing) in ML calculation (0-1; 0: no smoothing)

3.8.6 Parameters: Model Forcing

The forcing options that can be prescribed through runtime parameters in `data` are easy to use but somewhat limited in scope. More complex forcing setups are possible with optional packages such as pkg/exf or pkg/rbcs, in which case most or all of the parameters in this section can simply be left at their default value.

3.8.6.1 Momentum Forcing

This section only applies to the ocean. You need to generate wind-stress data into two files zonalWindFile and meridWindFile corresponding to the zonal and meridional components of the wind stress, respectively (if you want the stress to be along the direction of only one of the model horizontal axes, you only need to generate one file). The format of the files is similar to the bathymetry file. The zonal (meridional) stress data are assumed to be in pascals and located at U-points (V-points). See the matlab program `gendata.m` in the `input` directories of `verification` for several tutorial example (e.g. gendata.m in the *barotropic gyre tutorial*) to see how simple analytical wind forcing data are generated for the case study experiments.

Parameter	Group	Default	Description
momForcing	PARM01	TRUE	included external forcing of momentum on/off flag
zonalWindFile	PARM05	' '	filename for 2D specification of zonal component of wind forcing (N/m^2)
meridWindFile	PARM05	' '	filename for 2D specification of meridional component of wind forcing (N/m^2)
momForcingOutAB	PARM03	0	1: take momentum forcing out of Adams-Bashforth time stepping
momTidalForcing	PARM01	TRUE	tidal forcing of momentum equation on/off flag (requires tidal forcing files)
ploadFile	PARM05	' '	filename for 2D specification of atmospheric pressure loading (ocean z-coor. only) (Pa)

3.8.6.2 Tracer Forcing

A combination of flux data and relaxation terms can be used for driving the tracer equations. For potential temperature, heat flux data (in W/m^2) can be stored in the 2D binary file surfQnetfile. Alternatively or in addition, the forcing can be specified through a relaxation term. The SST data to which the model surface temperatures are restored are stored in the 2D binary file thetaClimFile. The corresponding relaxation time scale coefficient is set through the variable tauThetaClimRelax (in seconds). The same procedure applies for salinity with the variable names EmPmRfile, saltClimFile, and tauSaltClimRelax for freshwater flux (in m/s) and surface salinity (in psu or g/kg) data files and relaxation timescale coefficient (in seconds), respectively.

Parameter	Group	Default	Description
tempForcing	PARM01	TRUE	external forcing of temperature forcing on/off flag
surfQnetFile	PARM05	' '	filename for 2D specification of net total heat flux (W/m^2)
surfQswFile	PARM05	' '	filename for 2D specification of net shortwave flux (W/m^2); requires #define SHORTWAVE_HEATING
tauThetaClimRelax	PARM03	0.0	temperature (surface) relaxation time scale (s)
lambdaThetaFile	PARM05	' '	filename for 2D specification of inverse temperature (surface) relaxation time scale (1/s)
ThetaClimFile	PARM05	' '	filename for specification of (surface) temperature relaxation values (^{o}C)

Continued on next page

Table 3.11 – continued from previous page

Parameter	Group	Default	Description
balanceThetaClimRelax	PARM01	FALSE	subtract global mean heat flux due to temp. relaxation flux every time step on/off flag; requires #define ALLOW_BALANCE_RELAX
balanceQnet	PARM01	FALSE	subtract global mean Qnet every time step on/off flag; requires #define ALLOW_BALANCE_FLUXES
geothermalFile	PARM05	' '	filename for 2D specification of geothermal heating flux through bottom (W/m^2); requires #define ALLOW_GEOTHERMAL_FLUX
temp_EvPrRn	PARM01	UNSET	temperature of rain and evaporated water (unset, use local temp.) (^oC)
allowFreezing	PARM01	FALSE	limit (ocean) temperature at surface to >= -1.9^oC
saltForcing	PARM01	TRUE	external forcing of salinity forcing on/off flag
convertFW2Salt	PARM01	3.5E+01	salinity used to convert freshwater flux to salt flux (-1: use local S) (psu or g/kg) (note default is -1 if useRealFreshWaterFlux= .TRUE.)
rhoConstFresh	PARM01	rhoConst	constant reference density for fresh water (rain) (kg/m^3)
EmPmRFile	PARM05	' '	filename for 2D specification of net freshwater flux (m/s)
saltFluxFile	PARM05	' '	filename for 2D specification of salt flux (from seaice) ($psu.kg/m^2/s$)
tauSaltClimRelax	PARM03	0.0	salinity (surface) relaxation time scale (s)
lambdaSaltFile	PARM05	' '	filename for 2D specification of inverse salinity (surface) relaxation time scale (1/s)
saltClimFile	PARM05	' '	filename for specification of (surface) salinity relaxation values (psu or g/kg)
balanceSaltClimRelax	PARM01	FALSE	subtract global mean flux due to salt relaxation every time step on/off flag
balanceEmPmR	PARM01	FALSE	subtract global mean EmPmR every time step on/off flag; requires #define ALLOW_BALANCE_FLUXES
salt_EvPrRn	PARM01	0.0	salinity of rain and evaporated water (psu or g/kg)
selectAddFluid	PARM01	0	add fluid to ocean interior options (-1, 0: off, or 1); requires #define ALLOW_ADDFLUID
temp_addMass	PARM01	temp_EvPrRn	temp. of added or removed (interior) water (^oC); requires #define ALLOW_ADDFLUID
salt_addMass	PARM01	salt_EvPrRn	salinity of added or removed (interior) water (^oC); requires #define ALLOW_ADDFLUID
addMassFile	PARM05	' '	filename for 3D specification of mass source/sink (+=source, kg/s); requires #define ALLOW_ADDFLUID
balancePrintMean	PARM01	FALSE	print subtracted balancing means to STDOUT on/off flag; requires #define ALLOW_BALANCE_FLUXES and/or #define ALLOW_BALANCE_RELAX
latBandClimRelax	PARM03	whole domain	relaxation to (T,S) climatology equatorward of this latitude band is applied
tracForcingOutAB	PARM03	0	1: take T, S, and pTracer forcing out of Adams-Bashforth time stepping

3.8.6.3 Periodic Forcing

To prescribe time-dependent periodic forcing, concatenate successive time records into a single file ordered in a (x, y,time) fashion and set the following variables: periodicExternalForcing to .TRUE., externForcingPeriod to the period (in seconds between two records in input files) with which the forcing varies (e.g., 1 month), and externForc-

ingCycle to the repeat time (in seconds) of the forcing (e.g., 1 year; note externForcingCycle must be a multiple of externForcingPeriod). With these variables specified, the model will interpolate the forcing linearly at each iteration.

Parameter	Group	Default	Description
periodicExternalForcing	PARM03	FALSE	allow time-dependent periodic forcing on/off flag
externForcingPeriod	PARM03	0.0	period over which forcing varies (e.g. monthly) (s)
externForcingCycle	PARM03	0.0	period over which the forcing cycle repeats (e.g. one year) (s)

3.8.7 Parameters: Simulation Controls

3.8.7.1 Run Start and Duration

The beginning of a simulation is set by specifying a start time (in seconds) through the real variable startTime or by specifying an initial iteration number through the integer variable nIter0. If these variables are set to non-zero values, the model will look for a "pickup" file (by default, `pickup.0000nIter0`) to restart the integration. The end of a simulation is set through the real variable endTime (in seconds). Alternatively, one can instead specify the number of time steps to execute through the integer variable nTimeSteps. Iterations are referenced to deltaTClock, i.e., each iteration is deltaTClock seconds of model time.

Parameter	Group	Default	Description
nIter0	PARM03	0	starting timestep iteration number for this integration
nTimeSteps	PARM03	0	number of (model clock) timesteps to execute
nEndIter	PARM03	0	run ending timestep iteration number (alternate way to prescribe nTimeSteps)
startTime	PARM03	baseTime	run start time for this integration (s) (alternate way to prescribe nIter0)
endTime	PARM03	0.0	run ending time (s) (with startTime, alternate way to prescribe nTimeSteps)

3.8.7.2 Input/Output Files

The precision with which to read binary data is controlled by the integer variable readBinaryPrec, which can take the value 32 (single precision) or 64 (double precision). Similarly, the precision with which to write binary data is controlled by the integer variable writeBinaryPrec. By default, MITgcm writes output (snapshots, diagnostics, and pickups) separately for individual tiles, leaving it to the user to reassemble these into global files, if needed (scripts are available in utils/). There are two options however to have the model do this for you. Setting globalFiles to `.TRUE.` should always work in a single process setup (including multi-threaded processes), but for MPI runs this will depend on the platform – it requires simultaneous write access to a common file (permissible in typical Lustre setups, but not on all file systems). Alternatively, one can set useSingleCpuIO to `.TRUE.` to generate global files, which should always work, but requires additional mpi-passing of data and may result in slower execution.

Parameter	Group	Default	Description
globalFiles	PARM01	FALSE	write output "global" (i.e. not per tile) files on/off flag
useSingleCpuIO	PARM01	FALSE	only master MPI process does I/O (producing global output files)
the_run_name	PARM05	' '	string identifying the name of the model "run" for meta files
readBinaryPrec	PARM01	32	precision used for reading binary files (32 or 64)
writeBinaryPrec	PARM01	32	precision used for writing binary files (32 or 64)
outputTypesInclusive	PARM03	FALSE	allows writing of output files in multiple formats (i.e. pkg/mdsio and pkg/mnc)
rwSuffixType	PARM03	0	controls the format of the pkg/mdsio binary file "suffix" • 0: use iteration number (myIter, I10.10) • 1: 100*myTime • 2: myTime • 3: myTime/360 • 4: myTime/3600 where myTime is model time in seconds
mdsioLocalDir	PARM05	' '	if not blank, read-write output tiled files from/to this directory name (+four-digit processor-rank code)

3.8.7.3 Frequency/Amount of Output

The frequency (in seconds) with which output is written to disk needs to be specified. dumpFreq controls the frequency with which the instantaneous state of the model is written. monitorFreq controls the frequency with which monitor output is dumped to the standard output file(s). The frequency of output is referenced to deltaTClock.

Parameter	Group	Default	Description
dumpFreq	PARM03	0.0	interval to write model state/snapshot data (s)
dumpInitAndLast	PARM03	TRUE	write out initial and last iteration model state on/off flag
diagFreq	PARM03	0.0	interval to write additional intermediate (debugging cg2d/3d) output (s)
monitorFreq	PARM03	lowest of other output *Freq parms	interval to write monitor output (s)
monitorSelect	PARM03	2 (3 if fluid is water)	select group of monitor variables to output • 1: dynamic variables only • 2: add vorticity variables • 3: add surface variables
debugLevel	PARM01	depends on debugMode	level of printing of MITgcm activity messages/statistics (1-5, higher -> more activity messages)
plotLevel	PARM01	debugLevel	controls printing of field maps (1-5, higher -> more fields)

3.8.7.4 Restart/Pickup Files

chkPtFreq and pchkPtFreq control the output frequency of rolling and permanent pickup (a.k.a. checkpoint) files, respectively. These frequencies are referenced to deltaTClock.

Parameter	Group	Default	Description
pChkPtFreq	PARM03	0.0	permanent restart/pickup checkpoint file write interval (s)
chkPtFreq	PARM03	0.0	rolling restart/pickup checkpoint file write interval (s)
pickupSuff	PARM03	' '	force run to use pickups (even if nIter0 =0) and read files with this suffix (10 char. max)
pickupStrictlyMatch	PARM03	TRUE	force pickup (meta) file formats to exactly match (or terminate with error) on/off flag
writePickupAtEnd	PARM03	FALSE	write a (rolling) pickup file at run completion on/off flag
usePickupBeforeC54	PARM01	FALSE	initialize run using old pickup format from code prior to checkpoint54a
startFromPickupAB2	PARM03	FALSE	using Adams-Bashforth-3, start using Adams-Bashforth-2 pickup format; requires #define ALLOW_ADAMSBASHFORTH_3

3.8.8 Parameters Used In Optional Packages

Some optional packages were not written with package-specific namelist parameters in a `data.${pkg}` file; or for historical and/or other reasons, several package-specific namelist parameters remain in `data`.

3.8.8.1 C-D Scheme

(package pkg/cd_code)

If you run at a sufficiently coarse resolution, you might choose to enable the C-D scheme for the computation of the Coriolis terms. The variable tauCD, which represents the C-D scheme coupling timescale (in seconds) needs to be set.

Parameter	Group	Default	Description
useCDscheme	PARM01	FALSE	use C-D scheme for Coriolis terms on/off flag
tauCD	PARM03	deltaTMom	C-D scheme coupling timescale (s)
rCD	PARM03	1 - deltaTMom/tauCD	C-D scheme normalized coupling parameter (non-dim.)
epsAB_CD	PARM03	abEps	Adams-Bashforth-2 stabilizing weight used in C-D scheme

3.8.8.2 Automatic Differentiation

(package pkg/autodiff; see Section 7)

Parameter	Group	Default	Description
nTimeSteps_l2	PARM03	4	number of inner timesteps to execute per timestep
adjdumpFreq	PARM03	0.0	interval to write model state/snapshot data adjoint run (s)
adjMonitorFreq	PARM03	0.0	interval to write monitor output adjoint run (s)
adTapeDir	PARM05	' '	if not blank, read-write checkpointing files from/to this directory name

3.8.9 Execution Environment Parameters

If running multi-threaded (i.e., using shared memory/OpenMP), you will need to set nTx and/or nTy so that nTx*nTy is the total number of threads (per process).

The parameter useCubedSphereExchange needs to be changed to `.TRUE.` if you are using any type of grid composed of interconnected individual faces, including the cubed sphere topology or a lat-lon cap grid. See (needs section to be written).

Note that setting flag debugMode to `.TRUE.` activates a separate set of debugging print statements than parameter debugLevel (see Section 3.8.7.3). The latter controls print statements that monitor model activity (such as opening files, etc.), whereas the former produces a more coding-oriented set of print statements (e.g., entering and exiting subroutines, etc.)

Parameter	Group	Default	Description
useCubedSphereExchange	EEPARMS	FALSE	use cubed-sphere topology domain on/off flag
nTx	EEPARMS	1	number of threads in the x direction
nTy	EEPARMS	1	number of threads in the y direction
useCoupler	EEPARMS	FALSE	communicate with other model components through a coupler on/off flag
useSETRLSTK	EEPARMS	FALSE	call C routine to set environment stacksize to 'unlimited'
useSIGREG	EEPARMS	FALSE	enable signal handler to receive signal to terminate run cleanly on/off flag
debugMode	EEPARMS	FALSE	print additional debugging messages; also "flush" STDOUT file unit after each print
printMapIncludesZeros	EEPARMS	FALSE	text map plots of fields should ignore exact zero values on/off flag
maxLengthPrt1D	EEPARMS	65	maximum number of 1D array elements to print to standard output

CHAPTER 4

MITgcm Tutorial Example Experiments

The full MITgcm distribution comes with a set of pre-configured numerical experiments. Some of these example experiments are tests of individual parts of the model code, but many are fully fledged numerical simulations. Full tutorials exist for a few of the examples, and are documented in sections Section 4.1 - Section 4.2. The other examples follow the same general structure as the tutorial examples. However, they only include brief instructions in text file README. The examples are located in subdirectories under the directory verification. Each example is briefly described below.

4.1 Barotropic Gyre MITgcm Example

(in directory verification/tutorial_barotropic_gyre/)

This example experiment demonstrates using the MITgcm to simulate a barotropic, wind-forced, ocean gyre circulation. The experiment is a numerical rendition of the gyre circulation problem described analytically by Stommel in 1948 [Sto48] and Munk in 1950 [Mun50], and numerically in Bryan (1963) [Bry63]. Note this tutorial assumes a basic familiarity with ocean dynamics and geophysical fluid dynamics; readers new to the field may which to consult one of the standard texts on these subjects, such as Vallis (2017) [Val17] or Cushman-Roisin and Beckers (2011) [CRB11].

In this experiment the model is configured to represent a rectangular enclosed box of fluid, 1200×1200 km in lateral extent. The fluid depth $D = 5$ km. The fluid is forced by a zonal wind stress, τ_x, that varies sinusoidally in the north-south direction and is constant in time. Topologically the grid is Cartesian and the Coriolis parameter f is defined according to a mid-latitude beta-plane equation

$$f(y) = f_0 + \beta y$$

where y is the distance along the 'north-south' axis of the simulated domain. For this experiment f_0 is set to $10^{-4}\mathrm{s}^{-1}$ and $\beta = 10^{-11}\mathrm{s}^{-1}\mathrm{m}^{-1}$.

The sinusoidal wind-stress variations are defined according to

$$\tau_x(y) = -\tau_0 \cos(\pi \frac{y}{L_y})$$

where L_y is the lateral domain extent and τ_0 is set to $0.1\mathrm{N\ m}^{-2}$.

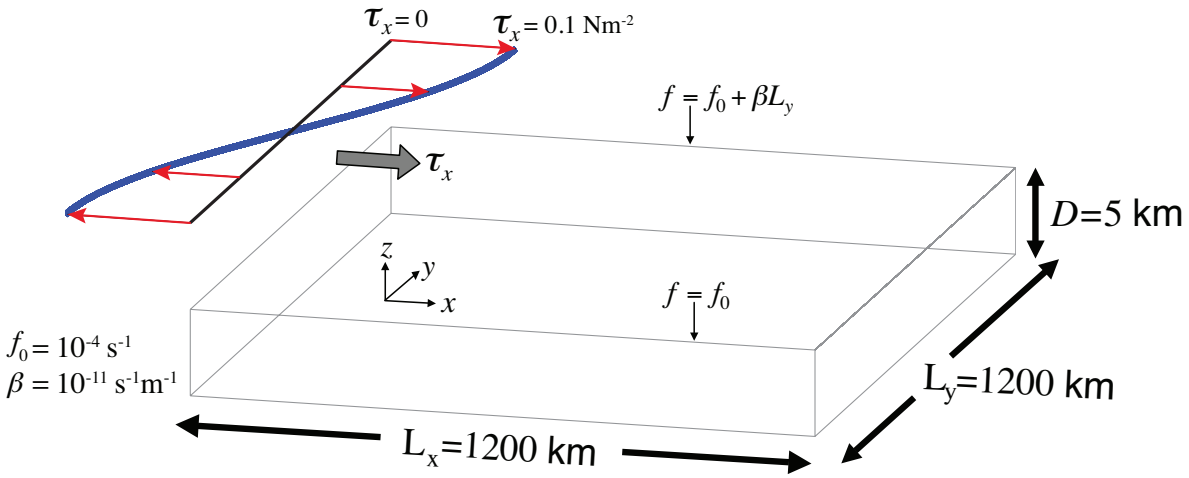

Figure 4.1: Schematic of simulation domain and wind-stress forcing function for barotropic gyre numerical experiment. The domain is enclosed by solid walls at $x = 0$, 1200 km and at $y = 0$, 1200 km.

Figure 4.1 summarizes the configuration simulated.

4.1.1 Equations Solved

The model is configured in hydrostatic form (the MITgcm default). The implicit free surface form of the pressure equation described in Marshall et al. (1997) [MHPA97] is employed. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation. The wind-stress momentum input is added to the momentum equation for the 'zonal flow', u. This effectively yields an active set of equations for this configuration as follows:

$$\frac{Du}{Dt} - fv + g\frac{\partial \eta}{\partial x} - A_h \nabla_h^2 u = \frac{\tau_x}{\rho_c D} \tag{4.1}$$

$$\frac{Dv}{Dt} + fu + g\frac{\partial \eta}{\partial y} - A_h \nabla_h^2 v = 0 \tag{4.2}$$

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{4.3}$$

where u and v are the x and y components of the flow vector $\vec{u}$, η is the free surface height, A_h the horizontal Laplacian viscosity, ρ_c is the fluid density, and g the acceleration due to gravity.

4.1.2 Discrete Numerical Configuration

The domain is discretized with a uniform grid spacing in the horizontal set to $\Delta x = \Delta y = 20$ km, so that there are sixty grid cells in the x and y directions. Vertically the model is configured using a single layer in depth, Δz, of 5000 m.

4.1.2.1 Numerical Stability Criteria

Let's start with our choice for the model's time step. To minimize the amount of required computational resources, typically one opts for as large a time step as possible while keeping the model solution stable. The advective Courant–Friedrichs–Lewy (CFL) condition (see Adcroft 1995 [Adc95]) for an extreme maximum horizontal flow speed is:

$$S_a = 2\left(\frac{|u|\Delta t}{\Delta x}\right) < 0.5 \text{ for stability} \tag{4.4}$$

The 2 factor on the left is because we have a 2D problem (in contrast with the more familiar 1D canonical stability analysis); the right hand side is 0.5 due to our default use of Adams-Bashforth2 (see Section 2.5) rather than the more familiar value of 1 that one would obtain using a forward Euler scheme. In our configuration, let's assume our solution will achieve a maximum $|u| = 1$ ms^{-1} (in reality, current speeds in our solution will be much smaller). To keep Δt safely below the stability threshold, let's choose Δt = 1200 s (= 20 minutes), which results in S_a = 0.12.

The numerical stability for inertial oscillations using Adams-Bashforth2 (Adcroft 1995 [Adc95])

$$S_i = f\Delta t < 0.5 \text{ for stability} \tag{4.5}$$

evaluates to 0.12 for our choice of δt, which is below the stability threshold.

There are two general rules in choosing a horizontal Laplacian eddy viscosity A_h:

- the resulting Munk layer width should be at least as large (preferably, larger) than the lateral grid spacing;
- the viscosity should be sufficiently small that the model is stable for horizontal friction, given the time step.

Let's use this first rule to make our choice for A_h, and check this value using the second rule. The theoretical Munk boundary layer width (as defined by the solution zero-crossing, see Pedlosky 1987 [Ped87]) is given by:

$$M_w = \frac{2\pi}{\sqrt{3}}\left(\frac{A_h}{\beta}\right)^{\frac{1}{3}} \tag{4.6}$$

For our configuration we will choose to resolve a boundary layer of $\approx$ 100 km, or roughly across five grid cells, so we set $A_h = 400$ m^2 s^{-1} (more precisely, this sets the full width at M_w = 124 km). This choice ensures that the frictional boundary layer is well resolved.

Given our choice of Δt, the stability parameter for the horizontal Laplacian friction (Adcroft 1995 [Adc95])

$$S_l = 2\left(4\frac{A_h\Delta t}{\Delta x^2}\right) < 0.6 \text{ for stability} \tag{4.7}$$

evaluates to 0.0096, which is well below the stability threshold. As in (4.4) the above criteria is for a 2D problem using Adams-Bashforth2 time stepping, with the 0.6 value on the right replacing the more familiar 1 that is obtained using a forward Euler scheme.

See Section 2.5 for additional details on Adams-Bashforth time-stepping and numerical stability criteria.

4.1.3 Code Configuration

The model configuration for this experiment resides under the directory verification/tutorial_barotropic_gyre/.

The experiment files

- verification/tutorial_barotropic_gyre/code/SIZE.h
- verification/tutorial_barotropic_gyre/input/data
- verification/tutorial_barotropic_gyre/input/data.pkg

- verification/tutorial_barotropic_gyre/input/eedata
- verification/tutorial_barotropic_gyre/input/bathy.bin
- verification/tutorial_barotropic_gyre/input/windx_cosy.bin

contain the code customizations and parameter settings for this experiment. Below we describe these customizations in detail.

Note: MITgcm's defaults are configured to simulate an ocean rather than an atmosphere, with vertical z-coordinates. To model the ocean using pressure coordinates using MITgcm, additional parameter changes are required; see tutorial ocean_in_p. To switch parameters to model an atmosphere, see tutorial Held_Suarez.

4.1.3.1 File code/SIZE.h

Listing 4.1: verification/tutorial_barotropic_gyre/code/SIZE.h

```
CBOP
C     !ROUTINE: SIZE.h
C     !INTERFACE:
C     include SIZE.h
C     !DESCRIPTION: \bv
C      *==========================================================*
C      | SIZE.h Declare size of underlying computational grid.
C      *==========================================================*
C      | The design here supports a three-dimensional model grid
C      | with indices I,J and K. The three-dimensional domain
C      | is comprised of nPx*nSx blocks (or tiles) of size sNx
C      | along the first (left-most index) axis, nPy*nSy blocks
C      | of size sNy along the second axis and one block of size
C      | Nr along the vertical (third) axis.
C      | Blocks/tiles have overlap regions of size OLx and OLy
C      | along the dimensions that are subdivided.
C      *==========================================================*
C      \ev
C
C      Voodoo numbers controlling data layout:
C      sNx :: Number of X points in tile.
C      sNy :: Number of Y points in tile.
C      OLx :: Tile overlap extent in X.
C      OLy :: Tile overlap extent in Y.
C      nSx :: Number of tiles per process in X.
C      nSy :: Number of tiles per process in Y.
C      nPx :: Number of processes to use in X.
C      nPy :: Number of processes to use in Y.
C      Nx  :: Number of points in X for the full domain.
C      Ny  :: Number of points in Y for the full domain.
C      Nr  :: Number of points in vertical direction.
CEOP
       INTEGER sNx
       INTEGER sNy
       INTEGER OLx
       INTEGER OLy
       INTEGER nSx
       INTEGER nSy
       INTEGER nPx
       INTEGER nPy
       INTEGER Nx
```

(continues on next page)

(continued from previous page)

```
      INTEGER Ny
      INTEGER Nr
      PARAMETER (
     &           sNx =  62,
     &           sNy =  62,
     &           OLx =   2,
     &           OLy =   2,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
     &           Nx  = sNx*nSx*nPx,
     &           Ny  = sNy*nSy*nPy,
     &           Nr  =   1)

C     MAX_OLX :: Set to the maximum overlap region size of any array
C     MAX_OLY    that will be exchanged. Controls the sizing of exch
C                routine buffers.
      INTEGER MAX_OLX
      INTEGER MAX_OLY
      PARAMETER ( MAX_OLX = OLx,
     &            MAX_OLY = OLy )

```

Here we show a modified model/inc source code file, customizing MITgcm's array sizes to our model domain. This file must be uniquely configured for any model setup; using the MITgcm default model/inc/SIZE.h will in fact cause a compilation error. Note that MITgcm's storage arrays are allocated as static variables (hence their size must be declared in the source code), in contrast to some model codes which declare array sizes dynamically, i.e., through runtime (namelist) parameter settings.

For this first tutorial, our setup and run environment is the most simple possible: we run on a single process (i.e., NOT MPI and NOT multi-threaded) using a single model *"tile"*. For a more complete explanation of the parameter choices to use multiple tiles, see the tutorial Baroclinic Gyre.

- These lines set parameters sNx and sNy, the number of grid points in the x and y directions, respectively.

```
     &           sNx =  62,
     &           sNy =  62,
```

- These lines set parameters OLx and OLy in the x and y directions, respectively. These values are the overlap extent of a model tile, the purpose of which will be explained in later tutorials. Here, we simply specify the required minimum value (2) in both x and y.

```
     &           OLx =   2,
     &           OLy =   2,
```

- These lines set parameters nSx, nSy, nPx, and nPy, the number of model tiles and the number of processes in the x and y directions, respectively. As discussed above, in this tutorial we configure a single model tile on a single process, so these parameters are all set to the value one.

```
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
```

- This line sets parameter Nr, the number of points in the vertical dimension. Here we use just a single vertical level.

```
55      &           Nr   =   1)
```

- Note these lines summarize the horizontal size of the model domain (**NOT** to be edited).

```
53      &           Nx   = sNx*nSx*nPx,
54      &           Ny   = sNy*nSy*nPy,
```

Further information and examples about how to configure model/inc/SIZE.h are given in Section 6.3.1.

4.1.3.2 File input/data

Listing 4.2: verification/tutorial_barotropic_gyre/input/data

```
# Model parameters
# Continuous equation parameters
 &PARM01
 viscAh=4.E2,
 f0=1.E-4,
 beta=1.E-11,
 rhoNil=1000.,
 gBaro=9.81,
 rigidLid=.FALSE.,
 implicitFreeSurface=.TRUE.,
# momAdvection=.FALSE.,
 tempStepping=.FALSE.,
 saltStepping=.FALSE.,
 &

# Elliptic solver parameters
 &PARM02
 cg2dTargetResidual=1.E-7,
 cg2dMaxIters=1000,
 &

# Time stepping parameters
 &PARM03
 nIter0=0,
 nTimeSteps=10,
 deltaT=1200.0,
 pChkptFreq=31104000.0,
 chkptFreq=15552000.0,
 dumpFreq=15552000.0,
 monitorFreq=1200.,
 monitorSelect=2,
#-for longer run (3.0 yr):
# nTimeSteps=77760,
# monitorFreq=864000.,
 &

# Gridding parameters
 &PARM04
 usingCartesianGrid=.TRUE.,
 delX=62*20.E3,
 delY=62*20.E3,
 xgOrigin=-20.E3,
 ygOrigin=-20.E3,
```

(continues on next page)

(continued from previous page)

```
 delR=5000.,
 &

# Input datasets
 &PARM05
 bathyFile='bathy.bin'
 zonalWindFile='windx_cosy.bin',
#zonalWindFile='windx_siny.bin',
 meridWindFile=,
 &
```

This file, reproduced completely above, specifies the main parameters for the experiment. The parameters that are significant for this configuration (shown with line numbers to left) are as follows.

PARM01 - Continuous equation parameters

- This line sets parameter viscAh, the horizontal Laplacian viscosity, to 400 m^2 s^{-1}.

```
 viscAh=4.E2,
```

- These lines set f_0 and β (the Coriolis parameter f0 and the gradient of the Coriolis parameter beta) for our beta-plane to 1×10^{-4} s^{-1} and 1×10^{-11} $m^{-1}s^{-1}$, respectively.

```
 f0=1.E-4,
 beta=1.E-11,
```

- This line sets parameter rhoNil, a reference density which will also be used as ρ_c (parameter rhoConst) in (4.1), to 1000 kg/m^3.

```
 rhoNil=1000.,
```

- This line sets parameter gBaro, the acceleration due to gravity g (in the free surface terms in (4.1) and (4.2)), to 9.81 m/s^2. This is the MITgcm default value, i.e., the value used if this line were not included in `data`. One might alter this parameter for a reduced gravity model, or to simulate a different planet, for example.

```
 gBaro=9.81,
```

- These lines set parameters rigidLid and implicitFreeSurface in order to suppress the rigid lid formulation of the surface pressure inverter and activate the implicit free surface formulation.

```
 rigidLid=.FALSE.,
 implicitFreeSurface=.TRUE.,
```

- This line sets parameter momAdvection to suppress the (non-linear) momentum of advection terms in the momentum equations. However, note the # in column 1: this "comments out" the line, so using the above data file verbatim will in fact include the momentum advection terms (i.e., MITgcm default for this parameter is TRUE). We'll explore the linearized solution (i.e., by removing the leading #) in Section 4.1.5. Note the ability to comment out a line in a namelist file is not part of standard Fortran, but this feature is implemented for all MITgcm namelist files.

```
# momAdvection=.FALSE.,
```

- These lines set parameters tempStepping and saltStepping to suppress MITgcm's forward time integration of temperature and salt in the tracer equations, as these prognostic variables are not relevant for the model solution

in this configuration. By default, MITgcm solves equations governing these two (active) tracers; later tutorials will demonstrate how additional passive tracers can be included in the solution. The advantage of NOT solving the temperature and salinity equations is to eliminate many unnecessary computations. In most typical configurations however, one will want the model to compute a solution for T and S, which typically comprises the majority of MITgcm's processing time.

```
12  tempStepping=.FALSE.,
13  saltStepping=.FALSE.,
```

PARM02 - Elliptic solver parameters

- The first line sets the tolerance (parameter cg2dTargetResidual) that the 2D conjugate gradient solver, the iterative method used in the pressure method algorithm, will use to test for convergence. The second line sets parameter cg2dMaxIters, the maximum number of iterations. The solver will iterate until the residual falls below this target value (here, set to 1×10^{-7}) or until this maximum number of solver iterations is reached (here, set to a maximum 1000 iterations). Typically, the solver will converge in far fewer than 1000 iterations, but it does not hurt to allow for a large number. The chosen value for the target residual happens to be the MITgcm default, and will serve well in most model configurations.

```
18  cg2dTargetResidual=1.E-7,
19  cg2dMaxIters=1000,
```

PARM03 - Time stepping parameters

- This line sets the starting (integer) iteration number for the run. Here we set the value to zero, which starts the model from a new, initialized state. If nIter0 is non-zero, the model would require appropriate pickup files (i.e., restart files) in order to continue integration of an existing run.

```
24  nIter0=0,
```

- This line sets parameter nTimeSteps, the (integer) number of timesteps the model will integrate forward. Below, we have set this to integrate for just 10 time steps, for MITgcm automated testing purposes (Section 5.5). To integrate the solution to near steady state, uncomment the line a few lines further down where we set the value to 77760 time steps. When you make this change, be sure to also comment out the line that sets monitorFreq (see below).

```
25  nTimeSteps=10,
```

- This line sets parameter deltaT, the timestep used in stepping forward the model, to 1200 seconds. In combination with the larger value of nTimeSteps mentioned above, we have effectively set the model to integrate forward for 77760×1200 s $= 3.0$ years (based on 360-day years), long enough for the solution to approach equilibrium.

```
26  deltaT=1200.0,
```

- These lines control the frequency at which restart (a.k.a. pickup) files are dumped by MITgcm. Here the value of pChkptFreq is set to 31,104,000 seconds (=1.0 years) of model time; this controls the frequency of "permanent" checkpoint pickup files. With permanent files, the model's iteration number is part of the file name (as the filename "suffix"; see Section 4.1.4.2) in order to save it as a labelled, permanent, pickup state. The value of ChkptFreq is set to 15,552,000 seconds (=0.5 years); the pickup files written at this frequency but will NOT include the iteration number in the filename, instead toggling between `ckptA` and `ckptB` in the filename, and thus these files will be overwritten with new data every $2 \times$ 15,552,000 seconds. Temporary checkpoint files can be written more frequently without requiring additional disk space, for example to peruse (or re-run) the

model state prior to an instability, or restart following a computer crash, etc. Either type of checkpoint file can be used to restart the model.

```
27 pChkptFreq=31104000.0,
28 chkptFreq=15552000.0,
```

- This line sets parameter dumpFreq, frequency of writing model state snapshot diagnostics (of relevance in this setup: variables u, v, and η). Here, we opt for a snapshot of model state every 15,552,000 seconds (=0.5 years), or after every 12960 time steps of integration.

```
29 dumpFreq=15552000.0,
```

- These lines are set to dump monitor output (see Section 9.4) every 1200 seconds (i.e., every time step) to standard output. While this monitor frequency is needed for MITgcm automated testing, this is too much output for our tutorial run. Comment out this line and uncomment the line where monitorFreq is set to 864,000 seconds, i.e., output every 10 days. Parameter monitorSelect is set to 2 here to reduce output of non-applicable quantities for this simple example.

```
30 monitorFreq=1200.,
31 monitorSelect=2,
```

PARM04 - Gridding parameters

- This line sets parameter usingCartesianGrid, which specifies that the simulation will use a Cartesian coordinate system.

```
39 usingCartesianGrid=.TRUE.,
```

- These lines set the horizontal grid spacing of the model grid, as vectors delX and delY (i.e., Δx and Δy respectively). This syntax indicates that we specify 62 values in both the x and y directions, which matches the domain size as specified in SIZE.h. Grid spacing is set to 20×10^3 m (=20 km).

```
40 delX=62*20.E3,
41 delY=62*20.E3,
```

- The cartesian grid default origin is (0,0) so here we set the origin with parameters xgOrigin and ygOrigin to (-20000,-20000), accounting for the bordering solid wall. The centers of the grid boxes will thus be at -10 km, 10 km, 30 km, 50 km, ..., in both x and y directions.

```
42 xgOrigin=-20.E3,
43 ygOrigin=-20.E3,
```

- This line sets parameter delR, the vertical grid spacing in the z-coordinate (i.e., Δz), to 5000 m.

```
44 delR=5000.,
```

PARM05 - Input datasets

- This line sets parameter bathyFile, the name of the bathymetry file. See Section 4.1.3.5 for information about the file format.

```
49 bathyFile='bathy.bin'
```

- These lines specify the names of the files from which the surface wind stress is read. There is a separate file for the x-direction (zonalWindFile) and the y-direction (meridWindFile). Note, here we have left the latter parameter blank, as there is no meridional wind stress forcing in our example.

```
 zonalWindFile='windx_cosy.bin',
#zonalWindFile='windx_siny.bin',
 meridWindFile=,
```

4.1.3.3 File input/data.pkg

Listing 4.3: verification/tutorial_barotropic_gyre/input/data.pkg

```
# Packages
 &PACKAGES
 &
```

This file does not set any namelist parameters, yet is necessary to run – only standard packages (i.e., those compiled in MITgcm by default) are required for this setup, so no other customization is necessary. We will demonstrate how to include additional packages in other tutorial experiments.

4.1.3.4 File input/eedata

Listing 4.4: verification/tutorial_barotropic_gyre/input/data.pkg

```
# Example "eedata" file
# Lines beginning "#" are comments
#  nTx      :: No. threads per process in X
#  nTy      :: No. threads per process in Y
# debugMode :: print debug msg (sequence of S/R calls)
 &EEPARMS
 nTx=1,
 nTy=1,
 &
# Note: Some systems use & as the namelist terminator (as shown here).
#       Other systems use a / character.
```

This file uses standard default values (i.e., MITgcm default is single-threaded) and does not contain customizations for this experiment.

4.1.3.5 File `input/bathy.bin`

This file is a 2D(x, y) map of bottom bathymetry, specified as the z-coordinate of the solid bottom boundary. Here, the value is set to -5000 m everywhere except along the N, S, E, and W edges of the array, where the value is set to 0 (i.e., "land"). The domain in MITgcm is assumed doubly periodic (i.e., periodic in both x- and y-directions), so boundary walls are necessary to set up our enclosed box domain. The points are ordered from low to high coordinates in both axes (varying fastest in x), as a raw binary stream of data that is enumerated in the same way as standard MITgcm 2D horizontal arrays. By default, this file is assumed to contain 32-bit (single precision) binary numbers. The matlab program verification/tutorial_barotropic_gyre/input/gendata.m was used to generate this bathymetry file.

4.1.3.6 File `input/windx_cosy.bin`

Similar to file `input/bathy.bin`, this file is a 2D(x, y) map of τ_x wind stress values, formatted in the same manner. The units are Nm^{-2}. Although τ_x is only a function of y in this experiment, this file must still define a complete 2D map in order to be compatible with the standard code for loading forcing fields in MITgcm. The matlab program verification/tutorial_barotropic_gyre/input/gendata.m was used to generate this wind stress file. To run the barotropic jet variation of this tutorial example (see Figure 4.4), you will in fact need to run this matlab program to generate the file `input/windx_siny.bin`.

4.1.4 Building and running the model

To configure and compile the code (following the procedure described in Section 3.5.1):

```
cd build
../../../tools/genmake2 -mods ../code ««-of my_platform_optionFile»»
make depend
make
cd ..
```

To run the model (following the procedure in Section 3.6):

```
cd run
ln -s ../input/* .
ln -s ../build/mitgcmuv .
./mitgcmuv > output.txt
```

4.1.4.1 Standard output

Your run's standard output file should be similar to verification/tutorial_barotropic_gyre/results/output.txt. The standard output is essentially a log file of the model run. The following information is included (in rough order):

- startup information including MITgcm checkpoint release number and other execution environment information, and a list of activated packages (including all default packages, as well as optional packages).
- the text from all `data.*` and other critical files (in our example here, eedata, SIZE.h, data, and data.pkg).
- information about the grid and bathymetry, including dumps of all grid variables (only if Cartesian or spherical polar coordinates used, as is the case here).
- all runtime parameter choices used by the model, including all model defaults as well as user-specified parameters.
- monitor statistics at regular intervals (as specified by parameter monitorFreq in data. See Section 9.4).
- output from the 2D conjugate gradient solver. More specifically, statistics from the right-hand side of the elliptic equation – for our linear free-surface, see eq. (2.15) – are dumped for every model time step. If the model solution blows up, these statistics will increase to infinity, so one can see exactly when the problem occurred (i.e., to aid in debugging). Additional solver variables, such as number of iterations and residual, are included with the monitor statistics.
- a summary of end-of-run execution information, including user-, wall- and system-time elapsed during execution, and tile communication statistics. These statistics are provided for the overall run, and also broken down by time spent in various subroutines.

Different setups using non-standard packages and/or different parameter choices will include additional or different output as part of the standard output. It is also possible to select more or less output by changing the parameter debugLevel in data; see (missing doc for pkg debug).

STDERR.0000 - if errors (or warnings) occurred during the run, helpful warning and/or error message(s) would appear in this file.

4.1.4.2 Other output files

In addition to raw binary data files with .data extension, each binary file has a corresponding .meta file. These plain-text files include information about the array size, precision (i.e., float32 or float64), and if relevant, time information and/or a list of these fields included in the binary file. The .meta files are used by MITgcm utils when binary data are read.

The following output files are generated:

Grid Data: see Section 2.11 for definitions and description of the Arakawa C-grid staggering of model variables.

- XC, YC - grid cell center point locations
- XG, YG - locations of grid cell vertices
- RC, RF - vertical cell center and cell faces positions
- DXC, DYC - grid cell center point separations (Figure 2.6 b)
- DXG, DYG - separation of grid cell vertices (Figure 2.6 a)
- DRC, DRF - separation of vertical cell centers and faces, respectively
- RAC, RAS, RAW, RAZ - areas of the grid "tracer cells", "southern cells", "western cells" and "vorticity cells", respectively (Figure 2.6)
- hFacC, hFacS, hFacW - fractions of the grid cell in the vertical which are "open" as defined in the center and on the southern and western boundaries, respectively. These variables effectively contain the configuration bathymetric (or topographic) information.
- Depth - bathymetry depths

All these files contain 2D(x, y) data except RC, RF, DRC, DRF, which are 1D(z), and hFacC, hFacS, hFacW, which contain 3D(x, y, z) data. Units for the grid files depends on one's choice of model grid; here, they are all in given in meters (or m^2 for areas).

All the 2D grid data files contain .001.001 in their filename, e.g., DXC.001.001.data – this is the tile number in .XXX.YYY format. Here, we have just a single tile in both x and y, so both tile numbers are 001. Using multiple tiles, the default is that the local tile grid information would be output separately for each tile (as an example, see the baroclinic gyre tutorial, which is set up using multiple tiles), producing multiple files for each 2D grid variable.

State Variable Snapshot Data:

Eta.0000000000.001.001.data, Eta.0000000000.001.001.meta - this is a binary data snapshot of model dynamic variable etaN (the free-surface height) and its meta file, respectively. Note the tile number is included in the filename, as is the iteration number 0000000000, which is simply the time step (the iteration number here is referred to as the "suffix" in MITgcm parlance; there are options to change this suffix to something other than iteration number). In other words, this is a dump of the free-surface height from the initialized state, iteration 0; if you load up this data file, you will see it is all zeroes. More interesting is the free-surface height after some time steps have occurred. Snapshots are written according to our parameter choice dumpFreq, here set to 15,552,000 seconds, which is every 12960 time steps. We will examine the model solutions in Section 4.1.5. The free-surface height is a 2D(x, y) field.

Snapshot files exist for other prognostic model variables, in particular filenames starting with U (uVel), V (uVel), T (theta), and S (salt); given our setup, these latter two fields remain uniform in space and time, thus not very interesting until we explore a baroclinic gyre setup in tutorial_baroclinic_gyre. These are all 3D(x, y, z) fields. The format for the file names is similar to the free-surface height files. Also dumped are snapshots of diagnosed vertical velocity W (wVel) (note that in non-hydrostatic simulations, W is a fully prognostic model variable).

Checkpoint Files:

The following pickup files are generated:

- `pickup.0000025920.001.001.data`, `pickup.0000025920.001.001.meta`, etc. - written at frequency set by pChkptFreq
- `pickup.ckptA.001.001.data`, `pickup.ckptA.001.001.meta`, `pickup.ckptB.001.001.data`, `pickup.ckptB.001.001.meta` - written at frequency set by ChkptFreq

Other Model Output Data: For completeness, here we list the remaining default output files produced by MITgcm (despite being not particularly informative for this simple setup).

`RhoRef.data, RhoRef.meta` - this is a 1D(z) array of reference density. Here we have a single level and have not specified an equation of state relation, thus the file simply contains our prescribed value rhoNil.

`PHrefC.data, PHrefC.meta, PHrefF.data, PHrefF.meta` - these are 1D(z) arrays containing reference hydrostatic "pressure potential" $\phi = p/\rho_c$ (see Section 1.3.6), computed at the (vertical grid) cell centers and cell faces, respectively. In our setup here, `PHrefC` is simply $\frac{\rho_c * g * D/2}{\rho_c}$, i.e., computed at the midpoint of our single vertical cell.

`PH`, `PHL` files - these are a 3D(x, y, z) field of hydrostatic ϕ' (including free-surface contribution) at cell centers and a 2D(x, y) field of ocean bottom ϕ', respectively, as a function of time. To obtain full $\phi(t)$ values, `PHrefC` should be added to `PH`, and `PHrefF`(z =bottom) should be added to `PHL`.

4.1.5 Model Solution

After running the model for 77,760 time steps (3.0 years), the solution is near equilibrium. Given an approximate timescale of one month for barotropic Rossby waves to cross our model domain, one might expect the solution to require several years to achieve an equilibrium state. The model solution of free-surface height η (proportional to streamfunction) at $t = 3.0$ years is shown in Figure 4.2. For further details on this solution, particularly examining the effect of the non-linear terms with increasing Reynolds number, the reader is referred to Pedlosky (1987) [Ped87] section 5.11.

Using matlab for example, visualizing output using the utils/matlab/rdmds.m utility to load the binary data in `Eta.0000077760.001.001.data` is as simple as:

```
addpath ../../../utils/matlab/
XC=rdmds('XC'); YC=rdmds('YC');
Eta=rdmds('Eta',77760);
contourf(XC/1000,YC/1000,Eta,[-.04:.01:.04]); colorbar;
colormap((flipud(hot))); set(gca,'XLim',[0 1200]); set(gca,'YLim',[0 1200])
```

or using python (you will need to copy utils/python/MITgcmutils/MITgcmutils/mds.py to your run directory before proceeding):

```
import mds
import matplotlib.pyplot as plt
XC = mds.rdmds('XC'); YC = mds.rdmds('YC')
Eta = mds.rdmds('Eta', 77760)
plt.contourf(XC, YC, Eta, np.linspace(-0.02, 0.05,8), cmap='hot_r')
plt.colorbar(); plt.show()
```

Let's simplify the example by considering the linear problem where we neglect the advection of momentum terms. In other words, replace $\frac{Du}{Dt}$ and $\frac{Dv}{Dt}$ with $\frac{\partial u}{\partial t}$ and $\frac{\partial v}{\partial t}$, respectively, in in (4.1) and (4.2). To do so, we uncomment (i.e., remove the leading #) in the line `# momAdvection=.FALSE.,` in file `data` and re-run the model. Any existing output files will be overwritten.

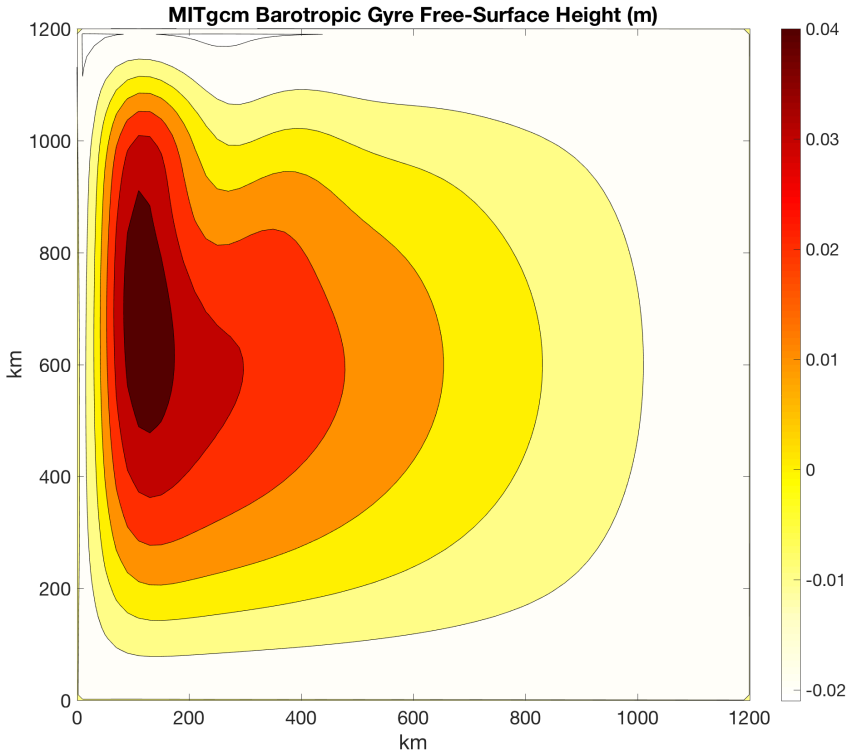

Figure 4.2: MITgcm solution to the barotropic gyre example after $t = 3.0$ years of model integration. Free surface height is shown in meters.

For the linearized equations, the Munk layer (equilibrium) analytical solution is given by:

$$\eta(x,y) = \frac{\tau_o}{\rho_c g D}\frac{f}{\beta}\left(1-\frac{x}{L_x}\right)\pi\sin(\pi\frac{y}{L_y})\left[1-\exp(\frac{-x}{2\delta_m})\left(\cos\frac{\sqrt{3}x}{2\delta_m}+\frac{1}{\sqrt{3}}\sin\frac{\sqrt{3}x}{2\delta_m}\right)\right]$$

where $\delta_m = (\frac{A_h}{\beta})^{\frac{1}{3}}$. Figure 4.3 displays the MITgcm output after switching off momentum advection vs. the analytical solution to the linearized equations. Success!

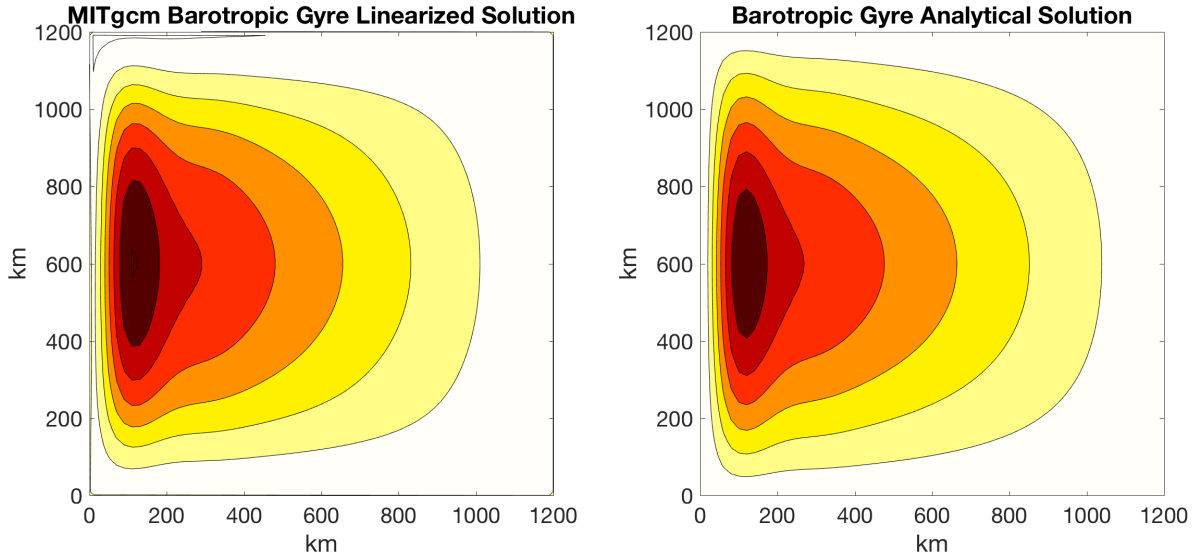

Figure 4.3: Comparison of free surface height for the near-equilibrium MITgcm solution ($t = 3.0$ years) with momentum advection switched off (left) and the analytical equilibrium solution to the linearized equation (right).

Finally, let's examine one additional simulation where we change the cosine profile of wind stress forcing to a sine profile. First, run the matlab script verification/tutorial_barotropic_gyre/input/gendata.m to generate the alternate sine profile wind stress, and place a copy in your run directory. Then, in file data, replace the line `zonalWindFile='windx_cosy.bin',` with `zonalWindFile='windx_siny.bin',`.

The free surface solution given this forcing is shown in Figure 4.4. Two "half gyres" are separated by a strong jet. We'll look more at the solution to this "barotropic jet" setup in later tutorial examples.

4.2 A Rotating Tank in Cylindrical Coordinates

(in directory: verification/rotating_tank/)

This example configuration demonstrates using the MITgcm to simulate a laboratory demonstration using a differentially heated rotating annulus of water. The simulation is configured for a laboratory scale on a $3^\circ \times 1\mathrm{cm}$ cyclindrical grid with twenty-nine vertical levels of 0.5cm each. This is a typical laboratory setup for illustration principles of GFD, as well as for a laboratory data assimilation project.

example illustration from GFD lab here

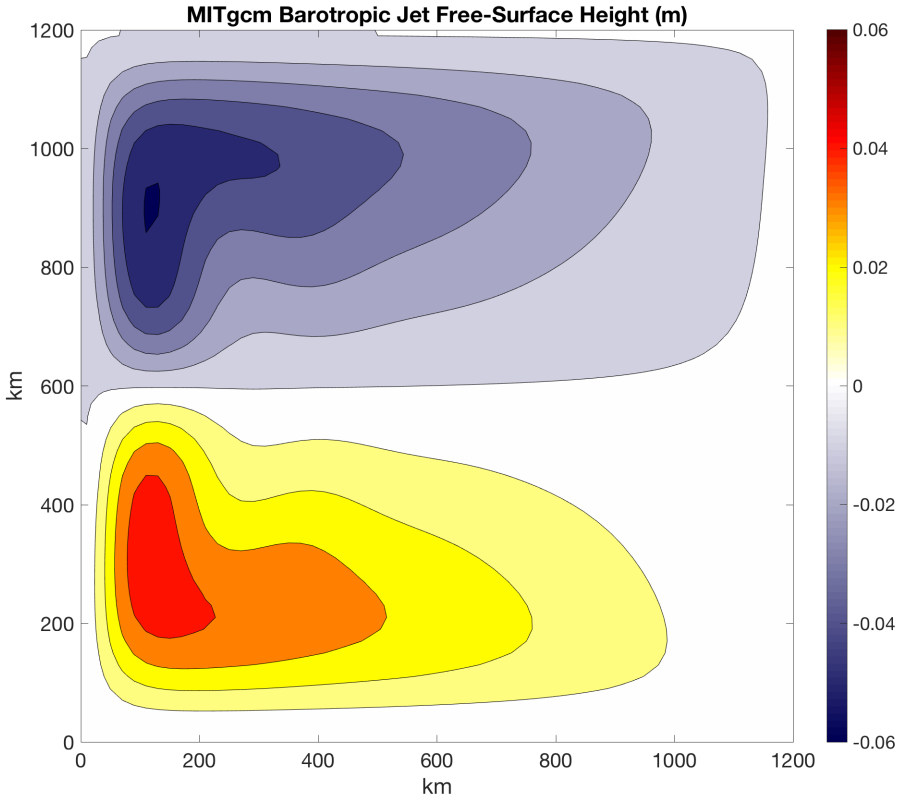

Figure 4.4: MITgcm equilibrium solution to the barotropic setup with alternate sine profile of wind stress forcing, producing a barotropic jet.

4.2.1 Equations Solved

4.2.2 Discrete Numerical Configuration

The domain is discretised with a uniform cylindrical grid spacing in the horizontal set to $\Delta a = 1$Delta phi=3^{circ}‘, so that there are 120 grid cells in the azimuthal direction and thirty-one grid cells in the radial, representing a tank 62cm in diameter. The bathymetry file sets the depth=0 in the nine lowest radial rows to represent the central of the annulus. Vertically the model is configured with twenty-nine layers of uniform 0.5cm thickness.

something about heat flux

4.2.3 Code Configuration

The model configuration for this experiment resides under the directory `verification/rotatingi_tank/`. The experiment files

- `input/data`
- `input/data.pkg`
- `input/eedata`
- `input/bathyPol.bin`
- `input/thetaPol.bin`
- `code/CPP\_EEOPTIONS.h`
- `code/CPP\_OPTIONS.h`
- `code/SIZE.h`

contain the code customizations and parameter settings for this experiments. Below we describe the customizations to these files associated with this experiment.

4.2.3.1 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Lines 9-10,
  - *viscAh=5.0E-6,*
  - *viscAz=5.0E-6,*

These lines set the Laplacian friction coefficient in the horizontal and vertical, respectively. Note that they are several orders of magnitude smaller than the other examples due to the small scale of this example.

- Lines 13-16,
  - diffKhT=2.5E-6,
  - diffKzT=2.5E-6,
  - diffKhS=1.0E-6,
  - diffKzS=1.0E-6,

These lines set horizontal and vertical diffusion coefficients for temperature and salinity. Similarly to the friction coefficients, the values are a couple of orders of magnitude less than most configurations.

- Line 17, *f0=0.5*, this line sets the coriolis term, and represents a tank spinning at about 2.4 rpm.

- Lines 23 and 24
  - *rigidLid=.TRUE.,*
  - *implicitFreeSurface=.FALSE.,*

These lines activate the rigid lid formulation of the surface pressure inverter and suppress the implicit free surface form of the pressure inverter.

- Line 40,
  - *nIter=0,*

This line indicates that the experiment should start from $t=0$ and implicitly suppresses searching for checkpoint files associated with restarting an numerical integration from a previously saved state. Instead, the file thetaPol.bin will be loaded to initialized the temperature fields as indicated below, and other variables will be initialized to their defaults.

- Line 43,
  - *deltaT=0.1,*

This line sets the integration timestep to $0.1s$. This is an unsually small value among the examples due to the small physical scale of the experiment. Using the ensemble Kalman filter to produce input fields can necessitate even shorter timesteps.

- Line 56,
  - *usingCylindricalGrid=.TRUE.,*

This line requests that the simulation be performed in a cylindrical coordinate system.

- Line 57,
  - *dXspacing=3,*

This line sets the azimuthal grid spacing between each x-coordinate line in the discrete grid. The syntax indicates that the discrete grid should be comprised of 120 grid lines each separated by 3^{circ}.

- Line 58,
  - *dYspacing=0.01,*

This line sets the radial cylindrical grid spacing between each a-coordinate line in the discrete grid to $1cm$.

- Line 59,
  - *delZ=29*0.005,*

This line sets the vertical grid spacing between each of 29 z-coordinate lines in the discrete grid to $0.005m$ (5~mm).

- Line 64,
  - *bathyFile='bathyPol.bin',*

This line specifies the name of the file from which the domain 'bathymetry' (tank depth) is read. This file is a two-dimensional (a, ϕ) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the phi coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates an area outside of the tank and a depth f $-0.145m$ indicates the tank itself.

- Line 65,
  - *hydrogThetaFile='thetaPol.bin',*

This line specifies the name of the file from which the initial values of temperature are read. This file is a three-dimensional (x, y, z) map and is enumerated and formatted in the same manner as the bathymetry file.

- Lines 66 and 67
  - *tCylIn = 0*
  - *tCylOut = 20*

These line specify the temperatures in degrees Celsius of the interior and exterior walls of the tank – typically taken to be icewater on the inside and room temperature on the outside.

Other lines in the file *input/data* are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

Listing 4.5: *verification/rotating_tank/input/data*

```
# ====================
# | Model parameters |
# ====================
#
# Continuous equation parameters
 &PARM01
 tRef=29*20.0,
 sRef=29*35.0,
 viscAh=5.0E-6,
 viscAz=5.0E-6,
 no_slip_sides=.FALSE.,
 no_slip_bottom=.FALSE.,
 diffKhT=2.5E-6,
 diffKzT=2.5E-6,
 diffKhS=1.0E-6,
 diffKzS=1.0E-6,
 f0=0.5,
 eosType='LINEAR',
 sBeta =0.,
 gravity=9.81,
 rhoConst=1000.0,
 rhoNil=1000.0,
#heatCapacity_Cp=3900.0,
 rigidLid=.TRUE.,
 implicitFreeSurface=.FALSE.,
 nonHydrostatic=.TRUE.,
 readBinaryPrec=32,
 &

# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=1000,
 cg2dTargetResidual=1.E-7,
 cg3dMaxIters=10,
 cg3dTargetResidual=1.E-9,
 &

# Time stepping parameters
 &PARM03
 nIter0=0,
 nTimeSteps=20,
#nTimeSteps=36000000,
 deltaT=0.1,
```

(continues on next page)

(continued from previous page)

```
 abEps=0.1,
 pChkptFreq=2.0,
#chkptFreq=2.0,
 dumpFreq=2.0,
 monitorSelect=2,
 monitorFreq=0.1,
 &

# Gridding parameters
 &PARM04
 usingCylindricalGrid=.TRUE.,
 dXspacing=3.,
 dYspacing=0.01,
 delZ=29*0.005,
 ygOrigin=0.07,
 &

# Input datasets
 &PARM05
 hydrogThetaFile='thetaPolR.bin',
 bathyFile='bathyPolR.bin',
 tCylIn  = 0.,
 tCylOut = 20.,
 &
```

4.2.3.2 File *input/data.pkg*

This file uses standard default values and does not contain customizations for this experiment.

4.2.3.3 File *input/eedata*

This file uses standard default values and does not contain customizations for this experiment.

4.2.3.4 File *input/thetaPol.bin*

The {it input/thetaPol.bin} file specifies a three-dimensional (x,y,z) map of initial values of $theta$ in degrees Celsius. This particular experiment is set to random values x around 20C to provide initial perturbations.

4.2.3.5 File *input/bathyPol.bin*

The {it input/bathyPol.bin} file specifies a two-dimensional (x,y) map of depth values. For this experiment values are either $0m$ or {bf -delZ}m, corresponding respectively to outside or inside of the tank. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays.

4.2.3.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39, - *sNx=120,*

this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40, - *sNy=31,*

this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

Listing 4.6: *verification/rotating_tank/code/SIZE.h*

```
1  CBOP
2  C    !ROUTINE: SIZE.h
3  C    !INTERFACE:
4  C    include SIZE.h
5  C    !DESCRIPTION: \bv
6  C     *==========================================================*
7  C     | SIZE.h Declare size of underlying computational grid.
8  C     *==========================================================*
9  C     | The design here supports a three-dimensional model grid
10 C     | with indices I,J and K. The three-dimensional domain
11 C     | is comprised of nPx*nSx blocks (or tiles) of size sNx
12 C     | along the first (left-most index) axis, nPy*nSy blocks
13 C     | of size sNy along the second axis and one block of size
14 C     | Nr along the vertical (third) axis.
15 C     | Blocks/tiles have overlap regions of size OLx and OLy
16 C     | along the dimensions that are subdivided.
17 C     *==========================================================*
18 C     \ev
19 C
20 C     Voodoo numbers controlling data layout:
21 C     sNx :: Number of X points in tile.
22 C     sNy :: Number of Y points in tile.
23 C     OLx :: Tile overlap extent in X.
24 C     OLy :: Tile overlap extent in Y.
25 C     nSx :: Number of tiles per process in X.
26 C     nSy :: Number of tiles per process in Y.
27 C     nPx :: Number of processes to use in X.
28 C     nPy :: Number of processes to use in Y.
29 C     Nx  :: Number of points in X for the full domain.
30 C     Ny  :: Number of points in Y for the full domain.
31 C     Nr  :: Number of points in vertical direction.
32 CEOP
33       INTEGER sNx
34       INTEGER sNy
35       INTEGER OLx
36       INTEGER OLy
37       INTEGER nSx
38       INTEGER nSy
39       INTEGER nPx
40       INTEGER nPy
41       INTEGER Nx
42       INTEGER Ny
43       INTEGER Nr
44       PARAMETER (
45      &           sNx =  30,
46      &           sNy =  23,
47      &           OLx =   3,
48      &           OLy =   3,
49      &           nSx =   4,
50      &           nSy =   1,
51      &           nPx =   1,
52      &           nPy =   1,
```

(continues on next page)

(continued from previous page)

```
53      &           Nx  = sNx*nSx*nPx,
54      &           Ny  = sNy*nSy*nPy,
55      &           Nr  =  29)
56
57  C     MAX_OLX :: Set to the maximum overlap region size of any array
58  C     MAX_OLY    that will be exchanged. Controls the sizing of exch
59  C                routine buffers.
60        INTEGER MAX_OLX
61        INTEGER MAX_OLY
62        PARAMETER ( MAX_OLX = OLx,
63       &            MAX_OLY = OLy )
64
```

4.2.3.7 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

4.2.3.8 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

CHAPTER 5

Contributing to the MITgcm

The MITgcm is an open source project that relies on the participation of its users, and we welcome contributions. This chapter sets out how you can contribute to the MITgcm.

5.1 Bugs and feature requests

If you think you've found a bug, the first thing to check that you're using the latest version of the model. If the bug is still in the latest version, then think about how you might fix it and file a ticket in the GitHub issue tracker. Please include as much detail as possible. At a minimum your ticket should include:

- what the bug does;
- the location of the bug: file name and line number(s); and
- any suggestions you have for how it might be fixed.

To request a new feature, or guidance on how to implement it yourself, please open a ticket with the following details:

- a clear explanation of what the feature will do; and
- a summary of the equations to be solved.

5.2 Using Git and Github

To contribute to the source code of the model you will need to fork the repository and place a pull request on GitHub. The two following sections describe this process in different levels of detail. If you are unfamiliar with git, you may wish to skip the quickstart guide and use the detailed instructions. All contributions to the source code are expected to conform with the *Coding style guide*. Contributions to the manual should follow the same procedure and conform with Section 5.6.

5.2.1 Quickstart Guide

1. Fork the project on GitHub (using the fork button).

2. Create a local clone (we strongly suggest keeping a separate repository for development work):

```
% git clone https://github.com/«GITHUB_USERNAME»/MITgcm.git
```

3. Move into your local clone directory (cd MITgcm) and and set up a remote that points to the original:

```
% git remote add upstream https://github.com/MITgcm/MITgcm.git
```

4. Make a new branch from `upstream/master` (name it something appropriate, such as 'bugfix' or 'newfeature' etc.) and make edits on this branch:

```
% git fetch upstream
% git checkout -b «YOUR_NEWBRANCH_NAME» upstream/master
```

5. When edits are done, do all git add's and git commit's. In the commit message, make a succinct (<70 char) summary of your changes. If you need more space to describe your changes, you can leave a blank line and type a longer description, or break your commit into multiple smaller commits. Reference any outstanding issues addressed using the syntax `#«ISSUE_NUMBER»`.

6. Push the edited branch to the origin remote (i.e. your fork) on GitHub:

```
% git push -u origin «YOUR_NEWBRANCH_NAME»
```

7. On GitHub, go to your fork and hit the compare and pull request (PR) button, provide the requested information about your PR (in particular, a non-trivial change to the model requires a suggested addition to doc/tag-index) and wait for the MITgcm head developers to review your proposed changes. In general the MITgcm code reviewers try to respond to a new PR within a week. The reviewers may accept the PR as is, or may request edits and changes. Occasionally the review team will reject changes that are not sufficiently aligned with and do not fit with the code structure. The review team is always happy to discuss their decisions, but wants to avoid people investing extensive effort in code that has a fundamental design flaw. The current review team is Jean-Michel Campin, Ed Doddridge, Chris Hill, Oliver Jahn, and Jeff Scott.

If you want to update your code branch before submitting a PR (or any point in development), follow the recipe below. It will ensure that your GitHub repo stays up to date with the main repository. Note again that your edits should always be to your development branch, not the master branch.

```
% git checkout master
% git pull upstream master
% git push origin master
% git checkout «YOUR_NEWBRANCH_NAME»
% git merge master
```

If you prefer, you can rebase rather than merge in the final step above; just be careful regarding your rebase syntax!

5.2.2 Detailed guide for those less familiar with Git and GitHub

What is Git? Git is a version control software tool used to help coordinate work among the many MITgcm model contributors. Version control is a management system to track changes in code over time, not only facilitating ongoing changes to code, but also as a means to check differences and/or obtain code from any past time in the project history. Without such a tool, keeping track of bug fixes and new features submitted by the global network of MITgcm contributors would be virtually impossible. If you are familiar with the older form of version control used by the MITgcm (CVS), there are many similarities, but we now take advantage of the modern capabilities offered by Git.

Git itself is open source linux software (typically included with any new linux installation, check with your sys-admin if it seems to be missing) that is necessary for tracking changes in files, etc. through your local computer's terminal session. All Git-related terminal commands are of the form `git «arguments»`. Important functions include syncing or updating your code library, adding files to a collection of files with edits, and commands to "finalize" these changes for sending back to the MITgcm maintainers. There are numerous other Git command-line tools to help along the way (see man pages via `man git`).

The most common git commands are:

- `git clone` download (clone) a repository to your local machine
- `git status` obtain information about the local git repository
- `git diff` highlight differences between the current version of a file and the version from the most recent commit
- `git add` stage a file, or changes to a file, so that they are ready for `git commit`
- `git commit` create a commit. A commit is a snapshot of the repository with an associated message that describes the changes.

What is GitHub then? GitHub is a website that has three major purposes: 1) Code Viewer: through your browser, you can view all source code and all changes to such over time; 2) "Pull Requests": facilitates the process whereby code developers submit changes to the primary MITgcm maintainers; 3) the "Cloud": GitHub functions as a cloud server to store different copies of the code. The utility of #1 is fairly obvious. For #2 and #3, without GitHub, one might envision making a big tarball of edited files and emailing the maintainers for inclusion in the main repository. Instead, GitHub effectively does something like this for you in a much more elegant way. Note unlike using (linux terminal command) git, GitHub commands are NOT typed in a terminal, but are typically invoked by hitting a button on the web interface, or clicking on a webpage link etc. To contribute edits to MITgcm, you need to obtain a github account. It's free; do this first if you don't have one already.

Before you start working with git, make sure you identify yourself. From your terminal, type:

```
% git config --global user.email «your_email@example.edu»
% git config --global user.name «'John Doe'»
```

(note the required quotes around your name). You should also personalize your profile associated with your GitHub account.

There are many online tutorials to using Git and GitHub (see for example https://akrabat.com/the-beginners-guide-to-contributing-to-a-github-project); here, we are just communicating the basics necessary to submit code changes to the MITgcm. Spending some time learning the more advanced features of Git will likely pay off in the long run, and not just for MITgcm contributions, as you are likely to encounter it in all sorts of different projects.

To better understand this process, Figure 5.1 shows a conceptual map of the Git setup. Note three copies of the code: the main MITgcm repository sourcecode "upstream" (i.e., owned by the MITgcm maintainers) in the GitHub cloud, a copy of the repository "origin" owned by you, also residing in the GitHub cloud, and a local copy on your personal computer or compute cluster (where you intend to compile and run). The Git and GitHub commands to create this setup are explained more fully below.

One other aspect of Git that requires some explanation to the uninitiated: your local linux copy of the code repository can contain different "branches", each branch being a different copy of the code repository (this can occur in all git-aware directories). When you switch branches, basic unix commands such as `ls` or `cat` will show a different set of files specific to current branch. In other words, Git interacts with your local file system so that edits or newly created files only appear in the current branch, i.e., such changes do not appear in any other branches. So if you swore you made some changes to a particular file, and now it appears those changes have vanished, first check which branch you are on (`git status` is a useful command here), all is probably not lost. NOTE: for a file to be "assigned" to

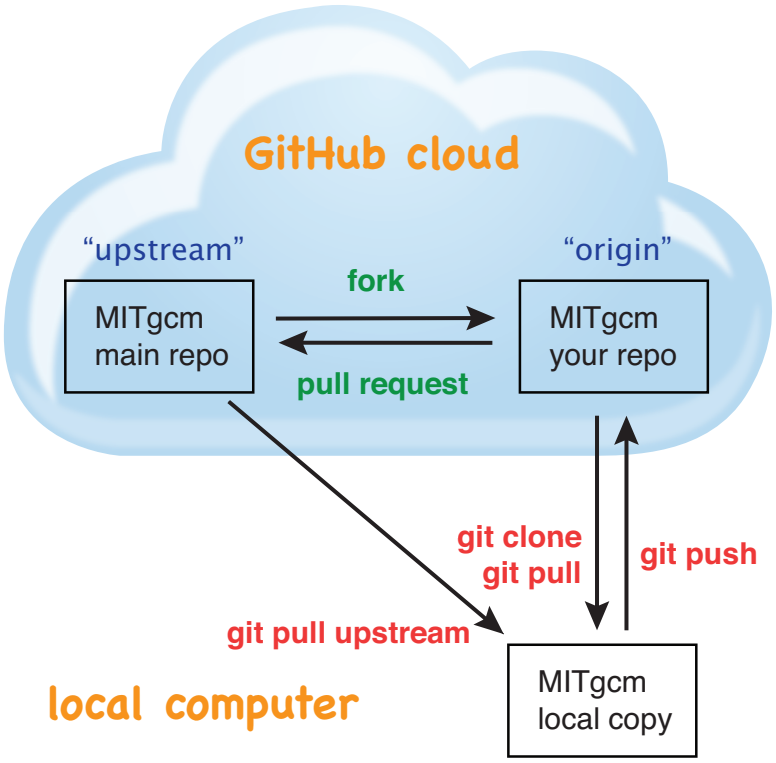

Figure 5.1: A conceptual map of the GitHub setup. Git terminal commands are shown in red, GitHub commands are shown in green.

a specific Git branch, Git must first be "made aware" of the file, which occurs after a `git add` and `git commit` (see *below*). Prior to this, the file will appear in the current folder independently, i.e., regardless of which git branch you are on.

A detailed explanation of steps for contributing MITgcm repository edits:

1. On GitHub, create a local copy of the repository in your GitHub cloud user space: from the main repository (https://github.com/MITgcm/MITgcm) hit the **Fork** button. As mentioned, your GitHub copy "origin" is necessary to streamline the collaborative development process – you need to create a place for your edits in the GitHub cloud, for developers to peruse.

2. Download the code onto your local computer using the git clone command. Even if you previously downloaded the code through a "git-aware" method (i.e., a git clone command, see Section 3.2.1), we **STRONGLY SUGGEST** you download a fresh repository, to a separate disk location, for your development work (keeping your research work separate). Type:

```
% git clone https://github.com/«GITHUB_USERNAME»/MITgcm.git
```

from your terminal (technically, here you are copying the forked "origin" version from the cloud, not the "upstream" version, but these will be identical at this point).

3. Move into the local clone directory on your computer:

```
% cd MITgcm
```

We need to set up a remote that points to the main repository:

```
% git remote add upstream https://github.com/MITgcm/MITgcm.git
```

This means that we now have two "remotes" of the project. A remote is just a pointer to a repository not on your computer, i.e., in the GitHub cloud, one pointing to your GitHub user space ("origin"), and this new remote pointing to the original ("upstream"). You can read and write into your "origin" version (since it belongs to you, in the cloud), but not into the "upstream" version. This command just sets up this remote, which is needed in step #4 – no actual file manipulation is done at this point. If in doubt, the command `git remote -v` will list what remotes have been set up.

4. Next make a new branch.

```
% git fetch upstream
% git checkout -b «YOUR_NEWBRANCH_NAME» upstream/master
```

You will make edits on this new branch, to keep these new edits completely separate from all files on the master branch. The first command `git fetch upstream` makes sure your new branch is the latest code from the main repository; as such, you can redo step 4 at any time to start additional, separate development projects (on a separate, new branch). Note that this second command above not only creates this new branch, from the `upstream/master` branch, it also switches you onto this newly created branch. Naming the branch something descriptive like 'newfeature' or 'bugfix' (preferably, be even more descriptive) is helpful.

5. Doing stuff! This usually comes in one of three flavors:

i) cosmetic changes, formatting, documentation, etc.;
ii) fixing bug(s), or any change to the code which results in different numerical output; or
iii) adding a feature or new package.

To do this you should:

- edit the relevant file(s) and/or create new files. Refer to *Coding style guide* for details on expected documentation standards and code style requirements. Of course, changes should be thoroughly tested to ensure they compile and run successfully!
- type `git add «FILENAME1» «FILENAME2» ...` to stage the file(s) ready for a commit command (note both existing and brand new files need to be added). "Stage" effectively means to notify Git of the the list of files you plan to "commit" for changes into the version tracking system. Note you can change other files and NOT have them sent to model developers; only staged files will be sent. You can repeat this `git add` command as many times as you like and it will continue to augment the list of files. `git diff` and `git status` are useful commands to see what you have done so far.
- use `git commit` to commit the files. This is the first step in bundling a collection of files together to be sent off to the MITgcm maintainers. When you enter this command, an editor window will pop up. On the top line, type a succinct (<70 character) summary of what these changes accomplished. If your commit is non-trivial and additional explanation is required, leave a blank line and then type a longer description of why the action in this commit was appropriate etc. It is good practice to link with known issues using the syntax `#ISSUE_NUMBER` in either the summary line or detailed comment. Note that all the changes do not have to be handled in a single commit (i.e. you can git add some files, do a commit, than continue anew by adding different files, do another commit etc.); the `git commit` command itself does not (yet) submit anything to maintainers.
- if you are fixing a more involved bug or adding a new feature, such that many changes are required, it is preferable to break your contribution into multiple commits (each documented separately) rather than submitting one massive commit; each commit should encompass a single conceptual change to the code base, regardless of how many files it touches. This will allow the MITgcm maintainers to more easily understand your proposed changes and will expedite the review process.

When your changes are tested and documented, continue on to step #6, but read all of step #6 and #7 before proceeding; you might want to do an optional "bring my development branch up to date" sequence of steps before step #6.

6. Now we "push" our modified branch with committed changes onto the origin remote in the GitHub cloud. This effectively updates your GitHub cloud copy of the MITgcm repo to reflect the wonderful changes you are contributing.

```
% git push -u origin «YOUR_NEWBRANCH_NAME»
```

Some time might elapse during step #5, as you make and test your edits, during which continuing development occurs in the main MITgcm repository. In contrast with some models that opt for static, major releases, the MITgcm is in a constant state of improvement and development. It is very possible that some of your edits occur to files that have also been modified by others. Your local clone however will not know anything about any changes that may have occurred to the MITgcm repo in the cloud, which may cause an issue in step #7 below, when one of three things will occur:

- the files you have modified in your development have **NOT** been modified in the main repo during this elapsed time, thus git will have no conflicts in trying to update (i.e. merge) your changes into the main repo.
- during the elapsed time, the files you have modified have also been edited/updated in the main repo, but you edited different places in these files than those edits to the main repo, such that git is smart enough to be able to merge these edits without conflict.
- during the elapsed time, the files you have modified have also been edited/updated in the main repo, but git is not smart enough to know how to deal with this conflict (it will notify you of this problem during step #7).

One option is to NOT attempt to bring your development code branch up to date, instead simply proceed with steps #6 and #7 and let the maintainers assess and resolve any conflict(s), should such occur (there is a checkbox 'Allow edits by maintainers' that is checked by default when you do step #7). If very little time elapsed during step #5, such conflict is less likely. However, if step #5 takes on the order of months, we do suggest you follow this recipe below to update the code and merge yourself. And/or during the development process, you might have reasons to bring the latest changes in the main repo into your development branch, and thus might opt to follow these same steps.

Development branch code update recipe:

```
% git checkout master
% git pull upstream master
% git push origin master
% git checkout «YOUR_NEWBRANCH_NAME»
% git merge master
```

This first command switches you from your development branch to the master branch. The second command above will synchronize your local master branch with the main MITgcm repository master branch (i.e. "pull" any new changes that might have occurred in the upstream repository into your local clone). Note you should not have made any changes to your clone's master branch; in other words, prior to the pull, master should be a stagnant copy of the code from the day you performed step #1 above. The `git push` command does the opposite of pull, so in the third step you are synchronizing your GitHub cloud copy ("origin") master branch to your local clone's master branch (which you just updated). Then, switch back to your development branch via the second `git checkout` command. Finally, the last command will merge any changes into your development branch. If conflicts occur that git cannot resolve, git will provide you a list of the problematic file names, and in these files, areas of conflict will be demarcated. You will need to edit these files at these problem spots (while removing git's demarcation text), then do a `git add «FILENAME»` for each of these files, followed by a final `git commit` to finish off the merger.

Some additional `git diff` commands to help sort out file changes, in case you want to assess the scope of development changes, are as follows. `git diff master upstream/master` will show you all differences between your local master branch and the main MITgcm repo, i.e., so you can peruse what parallel MITgcm changes have occurred while you were doing your development (this assumes you have not yet updated your clone's master branch). You can check for differences on individual files via `git diff master upstream/master «FILENAME»`. If you want to see all differences in files you have modified during your development, the command is `git diff master`. Similarly, to see a combined list of both your changes and those occurring to the main repo, `git diff upstream/master`.

Aside comment: if you are familiar with git, you might realize there is an alternate way to merge, using the "rebase" syntax. If you know what you are doing, feel free to use this command instead of our suggested merge command above.

7. Finally create a "pull request" (a.k.a. "PR"; in other words, you are requesting that the maintainers pull your changes into the main code repository). In GitHub, go to the fork of the project that you made (https://github.com/«GITHUB_USERNAME»/MITgcm.git). There is a button for "Compare and Pull" in your newly created branch. Click the button! Now you can add a final succinct summary description of what you've done in your commit(s), flag up any issues, and respond to the remaining questions on the PR template form. If you have made non-trivial changes to the code or documentation, we will note this in the MITgcm change log, doc/tag-index. Please suggest how to note your changes in doc/tag-index; we will not accept the PR if this field is left blank. The maintainers will now be notified and be able to peruse your changes! In general, the maintainers will try to respond to a new PR within a week. While the PR remains open, you can go back to step #5 and make additional edits, git adds, git commits, and then redo step #6; such changes will be added to the PR (and maintainers re-notified), no need to redo step #7.

Your pull request remains open until either the maintainers fully accept and merge your code changes into the main repository, or decide to reject your changes. Occasionally, the review team will reject changes that are not sufficiently aligned with and do not fit with the code structure; the review team is always happy to discuss their decisions, but wants to avoid people investing extensive additional effort in code that has a fundamental design flaw. But much more likely than outright rejection, you will instead be asked to respond to feedback, modify your code changes in some way, and/or clean up your code to better satisfy our style requirements, etc., and the pull request will remain open. In some cases, the maintainers might take initiative to make some changes to your pull request (such changes can then be incorporated back into your local branch simply by typing `git pull` from your branch), but more typically you will be asked to undertake the majority of the necessary changes.

It is possible for other users (besides the maintainers) to examine or even download your pull request; see *Reviewing pull requests*.

The current review team is Jean-Michel Campin, Ed Doddridge, Chris Hill, Oliver Jahn, and Jeff Scott.

5.3 Coding style guide

Detailed instructions or link to be added.

5.4 Creating MITgcm packages

Optional parts of code are separated from the MITgcm core driver code and organized into packages. The packaging structure provides a mechanism for maintaining suites of code, specific to particular classes of problem, in a way that is cleanly separated from the generic fluid dynamical engine. An overview of available MITgcm packages is presented in Section 8, as illustrated in Figure 8.1. An overview of how to include and use MITgcm packages in your setup is presented in Section 8.1.1, with specific details on using existing packages spread throughout Section 8, Section 9, and Section 10. This sub-section includes information necessary to create your own package for use with MITgcm.

The MITgcm packaging structure is described below using generic package names `${pkg}`. A concrete examples of a package is the code for implementing GM/Redi mixing: this code uses the package names `${PKG} = GMREDI`, `${pkg} = gmredi`, and `${Pkg} = gmRedi`.

5.4.1 Package structure

- Compile-time state: Given that each package is allowed to be compiled or not (e.g., all `${pkg}` listed in `packages.conf` are compiled, see Section 8.1.1.1), genmake2 keeps track of each package's compile-state in PACKAGES_CONFIG.h with CPP option `ALLOW_${PKG}` being defined (`#define`) or not (`#undef`). Therefore, in the MITgcm core code (or code from other included packages), calls to package-specific subroutines and package-specific header file `#include` statements must be protected within `#ifdef ALLOW_${PKG} ... ... #endif /* ALLOW_${PKG} */` (see *below*) to ensure that the model compiles when this ${pkg} is not compiled.
- Run-time state: The core driver part of the model can check for a run-time on/off switch of individual package(s) through the Fortran logical flag `use${Pkg}`. The information is loaded from a global package setup file called `data.pkg`. Note a `use${Pkg}` flag is NOT used within the package-local subroutine code (i.e., `${pkg}_«DO_SOMETHING».F` package source code).
- Each package gets its runtime configuration parameters from a file named `data.${pkg}`. Package runtime configuration options are imported into a common block held in a header file called `${PKG}.h`. Note in some packages, the header file `${PKG}.h` is split into `${PKG}_PARAMS.h`, which contains the package parameters, and `${PKG}_VARS.h` for the field arrays. The `${PKG}.h` header file(s) can be imported by other packages to check dependencies and requirements from other packages (see Section 5.4.2).

In order for a package's run-time state `use${Pkg}` to be set to true (i.e., "on"), the code build must have its compile-time state `ALLOW_${PKG}` defined (i.e., "included"), else mitgcmuv will terminate (cleanly) during initialization. A package's run-time state is not permitted to change during a model run.

Every call to a package routine from **outside** the package requires a check on BOTH compile-time and run-time states:

```
#include "PACKAGES_CONFIG.h"
#include "CPP_OPTIONS.h"
      .
      .
#ifdef ALLOW_${PKG}
#  include "${PKG}_PARAMS.h"
#endif
      .
      .
```

(continues on next page)

(continued from previous page)

```
      .

#ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) THEN
         .
         .
         CALL ${PKG}_DO_SOMETHING(...)
         .
      ENDIF
#endif
```

Within an individual package, the header file `${PKG}_OPTIONS.h` is used to set CPP flags specific to that package. This header file should include `PACKAGES_CONFIG.h` and CPP_OPTIONS.h, as shown in this example:

```
#ifndef ${PKG}_OPTIONS_H
#define ${PKG}_OPTIONS_H
#include "PACKAGES_CONFIG.h"
#include "CPP_OPTIONS.h"

#ifdef ALLOW_${PKG}
      .
      .
      .
#define ${PKG}_SOME_PKG_SPECIFIC_CPP_OPTION
      .
      .
      .
#endif /* ALLOW_${PKG} */
#endif /* ${PKG}_OPTIONS_H */
```

See for example GMREDI_OPTIONS.h.

5.4.2 Package boot sequence

All packages follow a required "boot" sequence outlined here:

```
S/R PACKAGES_BOOT()

S/R PACKAGES_READPARMS()
   #ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) CALL ${PKG}_READPARMS( retCode )
   #endif

S/R PACKAGES_INIT_FIXED()
   #ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) CALL ${PKG}_INIT_FIXED( retCode )
   #endif

S/R PACKAGES_CHECK()
   #ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) CALL ${PKG}_CHECK( retCode )
   #else
      IF ( use${Pkg} ) CALL PACKAGES_CHECK_ERROR('${PKG}')
   #endif
```

(continues on next page)

(continued from previous page)

```
S/R PACKAGES_INIT_VARIABLES()
   #ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) CALL ${PKG}_INIT_VARIA( )
   #endif
```

- **PACKAGES_BOOT()** determines the logical state of all `use${Pkg}` variables, as defined in the file `data.pkg`.
- **${PKG}_READPARMS()** is responsible for reading in the package parameters file `data.${pkg}` and storing the package parameters in `${PKG}.h` (or in `${PKG}_PARAMS.h`). ${PKG}_READPARMS is called in S/R packages_readparms.F, which in turn is called from S/R initialise_fixed.F.
- **${PKG}_INIT_FIXED()** is responsible for completing the internal setup of a package, including adding any package-specific variables available for output in pkg/diagnostics (done in S/R ${PKG}_DIAGNOSTICS_INIT). ${PKG}_INIT_FIXED is called in S/R packages_init_fixed.F, which in turn is called from S/R initialise_fixed.F. Note: some packages instead use `CALL ${PKG}_INITIALISE` (or the old form `CALL ${PKG}_INIT`).
- **${PKG}_CHECK()** is responsible for validating basic package setup and inter-package dependencies. ${PKG}_CHECK can also import parameters from other packages that it may need to check; this is accomplished through header files `${PKG}.h`. (It is assumed that parameters owned by other packages will not be reset during ${PKG}_CHECK !!!) ${PKG}_CHECK is called in S/R packages_check.F, which in turn is called from S/R initialise_fixed.F.
- **${PKG}_INIT_VARIA()** is responsible for initialization of all package variables, called after the core model state has been completely initialized but before the core model timestepping starts. This routine calls ${PKG}_READ_PICKUP, where any package variables required to restart the model will be read from a pickup file. ${PKG}_INIT_VARIA is called in packages_init_variables.F, which in turn is called from S/R initialise_varia.F. Note: the name ${PKG}_INIT_VARIA is not yet standardized across all packages; one can find other S/R names such as ${PKG}_INI_VARS or ${PKG}_INIT_VARIABLES or ${PKG}_INIT.

5.4.3 Package S/R calls

Calls to package subroutines within the core code timestepping loop can vary. Below we show an example of calls to do calculations, generate output and dump the package state (for pickup):

```
S/R DO_OCEANIC_PHYS()
   #ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) CALL ${PKG}_DO_SOMETHING( )
   #endif

S/R DO_THE_MODEL_IO()
   #ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) CALL ${PKG}_OUTPUT( )
   #endif

S/R PACKAGES_WRITE_PICKUP()
   #ifdef ALLOW_${PKG}
      IF ( use${Pkg} ) CALL ${PKG}_WRITE_PICKUP( )
   #endif
```

- **${PKG}_DO_SOMETHING()** refers to any local package source code file, which may be called from any model/src routine (or, from any subroutine in another package). An specific example would be the S/R call gmredi_calc_tensor.F from within the core S/R model/src/do_oceanic_phys.F.

- **${PKG}_OUTPUT()** is responsible for writing time-average fields to output files (although the cumulating step is done within other package subroutines). May also call other output routines (e.g., CALL ${PKG}_MONITOR) and write snapshot fields that are held in common blocks. Other temporary fields are directly dumped to file where they are available. Note that pkg/diagnostics output of ${PKG} variables is generated in pkg/diagnostics subroutines. ${PKG}_OUTPUT() is called in S/R do_the_model_io.F NOTE: 1) the S/R ${PKG}_DIAGS is used in some packages but is being replaced by ${PKG}_OUTPUT to avoid confusion with pkg/diagnostics functionality. 2) the output part is not yet in a standard form.
- **${PKG}_WRITE_PICKUP()** is responsible for writing a package pickup file, used in packages where such is necessary for a restart. ${PKG}_WRITE_PICKUP is called in packages_write_pickup.F which in turn is called from the_model_main.F.

Note: In general, subroutines in one package (pkgA) that only contains code which is connected to a 2nd package (pkgB) will be named pkgA_pkgB_something.F (e.g., gmredi_diagnostics_init.F).

5.4.4 Package "mypackage"

In order to simply creating the infrastructure required for a new package, we have created pkg/mypackage as essentially an existing package (i.e., all package variables defined, proper boot sequence, output generated) that does not do anything. Thus, we suggest you start with this "blank" package's code infrastructure and add your new package functionality to it, perusing the existing mypackage routines and editing as necessary, rather than creating a new package from scratch.

5.5 MITgcm code testing protocols

verification directory includes many examples intended for regression testing (some of which are tutorial experiments presented in detail in Section 4). Each one of these test-experiment directories contains "known-good" standard output files (see Section 5.5.2.1) along with all the input (including both code and data files) required for their re-calculation. Also included in verification is the shell script testreport to perform regression tests.

5.5.1 Test-experiment directory content

Each test-experiment directory («TESTDIR», see verification for the full list of choices) contains several standard subdirectories and files which testreport recognizes and uses when running a regression test. The directories and files that testreport uses are different for a forward test and an adjoint test (`testreport -adm`, see Section 5.5.2) and some test-experiments are set-up for only one type of regression test whereas others allow both types of tests (forward and adjoint). Also some test-experiments allow, using the same MITgcm executable, multiple tests using different parameters and input files, with a primary input set-up (e.g., `input/` or `input_ad/`) and corresponding results (e.g., `results/output.txt` or `results/output_adm.txt`) and with one or several secondary inputs (e.g., `input.«OTHER»/` or `input_ad.«OTHER»/`) and corresponding results (e.g., `results/output.«OTHER».txt` or `results/output_adm.«OTHER».txt`).

directory TESTDIR/code/ Contains the test-experiment specific source code (i.e., files that have been modified from the standard MITgcm repository version) used to build the MITgcm executable (`mitgcmuv`) for forward-test (using `genmake2 -mods=../code`).

It can also contain specific source files with the suffix `_mpi` to be used in place of the corresponding file (without suffix) for an MPI test (see Section 5.5.2). The presence or absence of `SIZE.h_mpi` determines whether or not an MPI test on this test-experiment is performed or skipped. Note that the original `code/SIZE.h_mpi` is not directly used as SIZE.h to build an MPI-executable; instead, a local copy `build/SIZE.h.mpi` is derived from `code/SIZE.h_mpi` by adjusting the number of processors (nPx, nPy) according to «NUMBER_OF_PROCS» (see Section 5.5.2, `testreport -MPI`); then it is linked to SIZE.h (`ln -s SIZE.h.mpi SIZE.h`) before building the MPI-executable.

directory TESTDIR/code_ad/ Contains the test-experiment specific source code used to build the MITgcm executable (`mitgcmuv_ad`) for adjoint-test (using `genmake2 -mods=../code_ad`). It can also contain specific source files with the suffix `_mpi` (see above).

directory «TESTDIR»/build/ Directory where testreport will build the MITgcm executable for forward and adjoint tests. It is initially empty except in some cases will contain an experiment specific `genmake_local` file (see Section 3.5.2).

directory TESTDIR/input/ Contains the input and parameter files used to run the primary forward test of this test-experiment.

It can also contain specific parameter files with the suffix `.mpi` to be used in place of the corresponding file (without suffix) for MPI tests, or with suffix `.mth` to be used for multi-threaded tests (see Section 5.5.2). The presence or absence of `eedata.mth` determines whether or not a multi-threaded test on this test-experiment is performed or skipped, respectively.

To save disk space and reduce downloading time, multiple copies of the same input file are avoided by using a shell script `prepare_run`. When such a script is found in `TESTDIR/input/`, testreport runs this script in directory `TESTDIR/run/` after linking all the input files from `TESTDIR/input/`.

directory TESTDIR/input_ad/ Contains the input and parameter files used to run the primary adjoint test of this test-experiment. It can also contain specific parameter files with the suffix `.mpi` and shell script `prepare_run` as described above.

directory TESTDIR/input.«OTHER»/ Contains the input and parameter files used to run the secondary OTHER forward test of this test-experiment. It can also contain specific parameter files with suffix `.mpi` or `.mth` and shell script `prepare_run` (see above).

The presence or absence the file `eedata.mth` determines whether or not a secondary multi-threaded test on this test-experiment is performed or skipped.

directory TESTDIR/input_ad.«OTHER»/ Contains the input and parameter files used to run the secondary OTHER adjoint test of this test-experiment. It can also contain specific parameter files with the suffix `.mpi` and shell script `prepare_run` (see above).

directory «TESTDIR»/results/ Contains reference standard output used for test comparison. `results/output.txt` and `results/output_adm.txt`, respectively, correspond to primary forward and adjoint test run on the reference platform (currently baudelaire.mit.edu) on one processor (no MPI, single thread) using the reference compiler (currently the GNU Fortran compiler gfortran). The presence of these output files determines whether or not testreport is testing or skipping this test-experiment. Reference standard output for secondary tests (`results/output.«OTHER».txt` or `results/output_adm.«OTHER».txt`) are also expected here.

directory TESTDIR/run/ Initially empty directory where testreport will run the MITgcm executable for primary forward and adjoint tests.

Symbolic links (using command `ln -s`) are made for input and parameter files (from `../input/` or from `../input_ad/`) and for MITgcm executable (from `../build/`) before the run proceeds. The sequence of links (function `linkdata` within shell script testreport) for a forward test is:

- link and rename or remove links to special files with suffix `.mpi` or `.mth` from `../input/`
- link files from ../input/
- execute `../input/prepare_run` (if it exists)

The sequence for an adjoint test is similar, with `../input_ad/` replacing `../input/`.

directory TESTDIR/tr_run.«OTHER»/ Directory created by testreport to run the MITgcm executable for secondary "OTHER" forward or adjoint tests.

The sequence of links for a forward secondary test is:

- link and rename or remove links to special files with suffix `.mpi` or `.mth` from `../input.OTHER/`
- link files from `../input.OTHER/`
- execute `../input.OTHER/prepare_run` (if it exists)
- link files from `../input/`
- execute `../input/prepare_run` (if it exists)

The sequence for an adjoint test is similar, with `../input_ad.OTHER/` and `../input_ad/` replacing `../input.OTHER/` and `../input/`.

5.5.2 The testreport utility

The shell script testreport, which was written to work with genmake2, can be used to build different versions of MITgcm code, run the various examples, and compare the output. On some systems, the testreport script can be run with a command line as simple as:

```
% cd verification
% ./testreport -optfile ../tools/build_options/linux_amd64_gfortran
```

The testreport script accepts a number of command-line options which can be listed using the `-help` option. The most important ones are:

`-ieee` (default) / `-fast` If allowed by the compiler (as defined in the specified optfile), use IEEE arithmetic (`genmake2 -ieee`). In contrast, `-fast` uses the optfile default for compiler flags.

`-devel` Use optfile development flags (assumes specified in optfile).

`-optfile «/PATH/FILENAME»` (or `-optfile '«/PATH/F1» «/PATH/F2» ...'`) This specifies a list of "options files" that will be passed to genmake2. If multiple options files are used (for example, to test different compilers or different sets of options for the same compiler), then each options file will be used with each of the test directories.

`-tdir «TESTDIR»` (or `-tdir '«TDIR1» «TDIR2» ...'`) This option specifies the test directory or list of test directories that should be used. Each of these entries should exactly match (note: they are case sensitive!) the names of directories in verification. If this option is omitted, then all directories that are properly formatted (that is, containing an input subdirectory and a `results/output.txt` file) will be used.

`-skipdir «TESTDIR»` (or `-skipdir '«TDIR1» «TDIR2» ...'`) This option specifies a test directory or list of test directories to skip. The default is to test **ALL** directories in verification.

`-MPI «NUMBER_OF_PROCS»` (or `-mpi`) If the necessary file «TESTDIR»`/code/SIZE.h_mpi` exists, then use it (and all `TESTDIR/code/*_mpi` files) for an MPI-enabled run. The option `-MPI` followed by the maximum number of processors enables to build and run each test-experiment using different numbers of MPI processors (specific number chosen by: multiple of nPx*nPy from «TESTDIR»`/code/SIZE.h_mpi` and not larger than «NUMBER_OF_PROCS»). The short option (`-mpi`) can only be used to build and run on 2 MPI processors (equivalent to `-MPI 2`).

Note that the use of MPI typically requires a special command option (see "-command" below) to invoke the MPI executable.

`-command='«SOME COMMANDS TO RUN»'` For some tests, particularly MPI runs, a specific command might be needed to run the executable. This option allows a more general command (or shell script) to be invoked.

The default here is for «SOME COMMANDS TO RUN» to be replaced by `mpirun -np TR_NPROC mitgcmuv`. If on your system you require something other than `mpirun`, you will need to use the option and specify your computer's syntax. Because the number of MPI processors varies according to each test-experiment, the keyword TR_NPROC will be replaced by its effective value, the actual number of MPI processors needed to run the current test-experiment.

-mth Compile with `genmake2 -omp` and run with multiple threads (using `eedata.mth`).

-adm Compile and test the adjoint suite of verification runs using TAF.

-clean Clean out all files/progress from any previously executed testreport runs.

-match «NUMBER» Set matching criteria to «NUMBER» of significant digits (default is 10 digits).

Additional testreport options are available to pass options to genmake2 (called during testreport execution) as well as additional options to skip specific steps of the testreport shell script. See `testreport -help` for a detailed list.

In the verification/ directory, the testreport script will create an output directory «tr_NAME_DATE_N», with your computer hostname substituted for NAME, the current date for DATE, followed by a suffix number N to distinguish from previous testreport output directories. Unless you specify otherwise using the `-tdir` or `-skipdir` options described above, all sub-directories (i.e., TESTDIR experiments) in verification will be tested. testreport writes progress to the screen (stdout) and reports into the «tr_NAME_DATE_N/TESTDIR» sub-directories as it runs. In particular, one can find, in each TESTDIR subdirectory, a `summary.txt` file in addition to log and/or error file(s) (depending how the run failed, if this occurred). `summary.txt` contains information about the run and a comparison of the current output with "reference output" (see *below* for information on how this reference output is generated). The test comparison involves several output model variables. By default, for a forward test, these are the 2D solver initial residual `cg2d_init_res` and 3D state variables (T, S, U, V) from pkg/monitor output; by default for an adjoint test, the cost-function and gradient-check. However, some test-experiments use some package-specific variables from pkg/monitor according to the file «`TESTDIR`»`/input[_ad][.`«`OTHER`»`]/tr_checklist` specification. Note that at this time, the only variables that are compared by testreport are those dumped in standard output via pkg/monitor, not output produced by pkg/diagnostics. Monitor output produced from **ALL** run time steps are compared to assess significant digit match; the worst match is reported. At the end of the testing process, a composite `summary.txt` file is generated in the top «tr_NAME_DATE_N» directory as a compact, combined version of the `summary.txt` files located in all TESTDIR sub-directories (a slightly more condensed version of this information is also written to file `tr_out.txt` in the top verification/ directory; note this file is overwritten upon subsequent testreport runs). Figure 5.2 shows an excerpt from the composite `summary.txt`, created by running the full testreport suite (in the example here, on a linux cluster, using gfortran):

The four columns on the left are build/run results (successful=Y, unsuccessful=N). Explanation of these columns is as follows:

- Gen2: did genmake2 build the makefile for this experiment without error?
- Dpnd: did the `make depend` for this experiment complete without error?
- Make: did the `make` successfully generate a `mitgcmuv` executable for this experiment?
- Run: did execution of this experiment startup and complete successfully?

The next sets of columns shows the number of significant digits matched from the monitor output "cg2d", "min", "max", "mean", and "s d" (standard deviation) for variables T, S, U, and V (see column headings), as compared with the reference output. NOTE: these column heading labels are for the default list of variables, even if different variables are specified in a `tr_checklist` file (for reference, the list of actual variables tested for a specific TESTDIR experiment is output near the end of the file `summary.txt` appearing in the specific TESTDIR experiment directory). For some experiments, additional variables are tested, as shown in "PTR 01", "PTR 02" sets of columns; testreport will detect if tracers are active in a given experiment and check digit match on their concentration values. A match to near-full machine precision is 15-16 digits; this generally will occur when a similar type of computer, similar operating system, and similar version of Fortran compiler are used for the test. Otherwise, different round-off can occur, and due to the chaotic nature of ocean and climate models, fewer digits (typically, 10-13 digits) are matched. A match of 22 digits generally is due to output being exactly 0.0. In some experiments, some variables may not be used or meaningful, which causes the '0' and '4' match results in several of the adjustment experiments above.

While the significant digit match for many variables is tested and displayed in `summary.txt`, only one of these is used to assess pass/fail (output to the right of the match test results) – the number bracketed by > and <. For example, see above for experiment advect_cs the pass/fail test occurs on variable "T: s d" (i.e., standard deviation of potential temperature), the first variable in the list specified in verification/advect_cs/input/tr_checklist. By default (i.e., if no

```
run: ./testreport -of ../tools/build_options/linux_amd64_gfortran
on : Linux c072 4.11.9-100.fc24.x86_64 #1 SMP Wed Jul 5 16:34:07 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

  OPTFILE=/home/jscott/MITgcm_fortesting/MITgcm/tools/build_options/linux_amd64_gfortran

default 10  ----T-----  ----S-----  ----U-----  ----V-----  --PTR 01--  --PTR 02--
G D M    c        m  s        m  s        m  s        m  s        m  s        m  s
e p a R  g  m  m  e  .  m  m  e  .  m  m  e  .  m  m  e  .  m  m  e  .  m  m  e  .
n n k u  2  i  a  a  d  i  a  a  d  i  a  a  d  i  a  a  d  i  a  a  d  i  a  a  d
2 d e n  d  n  x  n  .  n  x  n  .  n  x  n  .  n  x  n  .  n  x  n  .  n  x  n  .

Y Y Y Y 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 22 16 16>16<22 pass  1D_ocean_ice_column
Y Y Y Y>14<16 16 16 16 22 22 22 22 22 22 22 22 13  4 13 13  .  .  .  .  .  .  .  . pass  adjustment.128x64x1
Y Y Y Y>14<16 16 16 16 22 22 22 22 16 16 13 16 16 16  4 16  .  .  .  .  .  .  .  . pass  adjustment.cs-32x32x1
Y Y Y Y>14<16 16 16  0 22 22 22 22 16 16  0 14 16 16  0 16  .  .  .  .  .  .  .  . pass  adjustment.cs-32x32x1.nlfs
Y Y Y Y -- 16 16 16>16<16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  advect_cs
Y Y Y Y -- 14 14 16>16<16 16 16 16 16 16 16 22 16 16 16 22  .  .  .  .  .  .  .  . pass  advect_xy
Y Y Y Y -- 16 16 16>16<16 16 16 16 16 16 16 22 16 16 16 22  .  .  .  .  .  .  .  . pass  advect_xy.ab3_c4
Y Y Y Y -- 16 16 16>16<16 16 16 16 16 16 16 16 22 22 22 22  .  .  .  .  .  .  .  . pass  advect_xz
Y Y Y Y -- 16 16 16>16<16 14 16 14 16 16 16 16 22 22 22 22  .  .  .  .  .  .  .  . pass  advect_xz.nlfs
Y Y Y Y -- 13 14 16>14<12 16 16 16 16 16 16 16 22 22 22 22  .  .  .  .  .  .  .  . pass  advect_xz.pqm
Y Y Y Y>14<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  aim.5l_cs
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  . pass  aim.5l_cs.thSI
Y Y Y Y>14<16 16 13 14 16 16 13 13 16 16 13 13 16 16 13 14  .  .  .  .  .  .  .  . pass  aim.5l_Equatorial_Channel
Y Y Y Y>14<16 16 13 13 16 16 13 13 16 16 13 13 16 16 13 13  .  .  .  .  .  .  .  . pass  aim.5l_LatLon
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 pass  cfc_example
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  cheapAML_box
Y Y Y Y>14<16 16 16 12 22 22 22 22 13 16 12 16 13 14 12 16  .  .  .  .  .  .  .  . pass  deep_anelastic
Y Y Y Y>14<16 16 16 16 16 16 16 16 16 16 16 16 16 13 14 16  .  .  .  .  .  .  .  . pass  dome
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  exp2
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  exp2.rigidLid
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  . pass  exp4
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  . pass  exp4.nlfs
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  exp4.stevens
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  fizhi-cs-32x32x40
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  fizhi-cs-aqualev20
Y Y Y Y>16<16 16 16 16 22 22 22 22 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  fizhi-gridalt-hs
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  flt_example
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  front_relax
Y Y Y Y>16<16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16  .  .  .  .  .  .  .  . pass  front_relax.bvp
...
```

Figure 5.2: Example output from testreport `summary.txt`

file `tr_checklist` is present), pass/fail is assessed on the cg2d monitor output. See the testreport script for a list of permissible variables to test and a guide to their abbreviations. See `tr_checklist` files in the input subdirectories of several TESTDIR experiments (e.g., verification/advect_xz/input/tr_checklist) for examples of syntax (note, a + after a variable in a `tr_checklist file` is shorthand to compare the mean, minimum, maximum, and standard deviation for the variable).

5.5.2.1 Reference Output

Reference output is currently generated using the linux server `baudelaire.mit.edu` which employs an Intel Xeon Westmere processor running Fedora Core 13. For each verification experiment in the MITgcm repository, this reference output is stored in the file «`TESTDIR`»`/results/output.txt`, which is the standard output generated by running testreport (using a single process) on `baudelaire.mit.edu` using the gfortran (GNU Fortran) compiler version 4.4.5.

Using a different gfortran version (or a different Fortran compiler entirely), and/or running with MPI, a different operating system, or a different processor (cpu) type will generally result in output that differs to machine precision. The greater the number of such differences between your platform and this reference platform, typically the fewer digits of matching output precision.

5.5.3 The do_tst_2+2 utility

The shell script tools/do_tst_2+2 can be used to check the accuracy of the restart procedure. For each experiment that has been run through testreport, do_tst_2+2 executes three additional short runs using the tools/tst2+2 script. The first run makes use of the pickup files output from the run executed by testreport to restart and run for four time steps, writing pickup files upon completion. The second run is similar except only two time steps are executed, writing pickup files. The third run restarts from the end of the second run, executing two additional time steps, writing pickup files upon completion. In order to successfully pass do_tst_2+2, not only must all three runs execute and complete successfully, but the pickups generated at the end the first run must be identical to the pickup files from the end of the third run. Note that a prerequisite to running do_tst_2+2 is running testreport, both to build the executables used by do_tst_2+2, and to generate the pickup files from which do_tst_2+2 begins execution.

The tools/do_tst_2+2 script should be called from the verification/ directory, e.g.:

```
% cd verification
% ../tools/do_tst_2+2
```

The do_tst_2+2 script accepts a number of command-line options which can be listed using the `-help` option. The most important ones are:

`-t «TESTDIR»` Similar to testreport option `-tdir`, specifies the test directory or list of test directories that should be used. If omitted, the test is attempted in all sub-directories.

`-skd «TESTDIR»` Similar to testreport option `-skipdir`, specifies a test directory or list of test directories to skip.

`-mpi` Run the tests using MPI; requires the prerequisite testreport run to have been executed with the `-mpi` or `-MPI` «`NUMBER_OF_PROCS`» flag. No argument is necessary, as the do_tst_2+2 script will determine the correct number of processes to use for your executable.

`-clean` Clean up any output generated from the do_tst_2+2. This step is necessary if one wants to do additional testreport runs from these directories.

Upon completion, do_tst_2+2 will generate a file `tst_2+2_out.txt` in the verification/ directory which summarizes the results. The top half of the file includes information from the composite `summary.txt` file from the prerequisite testreport run. In the bottom half, new results from each verification experiment are given: each line starts with four Y/N indicators indicating if pickups from the testreport run were available, and whether runs 1, 2

and 3, completely successfully, respectively, followed by a pass or fail from the output pickup file comparison test, followed by the TESTDIR experiment name. In each «TESTDIR»/run subdirectory do_tst_2+2 also creates a log file tst_2+2_out.log which contains additional information. During do_tst_2+2 execution a separate directory of summary information, including log files for all failed tests, is created in an output directory «rs_NAME_DATE_N» similar to the syntax for the testreport output directory name. Note however this directory is deleted by default upon do_tst_2+2 completion, but can be saved by adding the do_tst_2+2 command line option -a NONE.

5.5.4 Daily Testing of MITgcm

On a daily basis, MITgcm runs a full suite of testreport (i.e., forward and adjoint runs, single process, single-threaded and mpi) on an array of different clusters, running using different operating systems, testing several different Fortran compilers. The reference machine baudelaire.mit.edu is one of such daily test machines. When changes in output occur from previous runs, even if as minor as changes in numeric output to machine precision, MITgcm maintainers are automatically notified.

Links to summary results from the daily testing are posted at http://mitgcm.org/public/testing.html.

5.5.5 Required Testing for MITgcm Code Contributors

5.5.5.1 Using testreport to check your new code

Before submitting your pull request for approval, if you have made any changes to MITgcm code, however trivial, you **MUST** complete the following:

- Run testreport (on all experiments) on an unmodified master branch of MITgcm. We suggest using the -devel option and gfortran (typically installed in most linux environments) although neither is strictly necessary for this test. Depending how different your platform is from our reference machine setup, typically most tests will pass but some match tests may fail; it is possible one or more experiments might not even build or run successfully. But even if there are multiple experiment fails or unsuccessful builds or runs, do not despair, the purpose at this stage is simply to generate a reference report on your local platform using the master code. It may take one or more hours for testreport to complete.
- Save a copy of this summary output from running testreport on the mastrer branch: from the verification directory, type cp tr_out.txt tr_out_master.txt. The file tr_out.txt is simply a condensed version of the composite summary.txt file located in the «tr_NAME_DATE_N» directory. Note we are not making this file "git-aware", as we have no desire to check this into the repo, so we are using an old-fashioned copy to save the output here for later comparison.
- Switch to your pull request branch, and repeat the testreport sequence using the same options.
- From the verification directory, type diff tr_out_master.txt tr_out.txt which will report any differences in testreport output from the above tests. If no differences occur (other than timestamp-related), see below if you are required to do a do_tst_2+2 test; otherwise, you are clear for submitting your pull request.

Differences might occur due to one or more of the following reasons:

- Your modified code no longer builds properly in one or more experiments. This is likely due to a Fortran syntax error; examine output and log files in the failed experiment TESTDIR to identify and fix the problem.
- The run in the modified code branch terminates due to a numerical exception error. This too requires further investigation into the cause of the error, and a remedy, before the pull request should be submitted.
- You have made changes which require changes to input parameters (e.g., renaming a namelist parameter, changing the units or function of an input parameter, etc.) This by definition is a "breaking change", which must be noted when completing the PR template – but should not deter you from submitting your PR. Ultimately, you and the maintainers will likely have to make changes to one or more verification experiments, but as a first step we will want to review your PR.

- You have made algorithmic changes which change model output in some or all setups; this too is a "breaking change" that should be noted in the PR template. As usual recourse, if the PR is accepted, the maintainers will re-generate reference output and push to the affected «TESTDIR»/results/ directories when the PR is merged.

Most typically, running testreport using a single process is a sufficient test. However, any code changes which call MITgcm routines (such as eesupp/src/global_sum.F) employing low-level MPI-directives should run testreport with the -mpi option enabled.

5.5.5.2 Using do_tst_2+2 to check your new code

If you make any kind of algorithmic change to the code, or modify anything related to generating or reading pickup files, you are also required to also complete a do_tst_2+2. Again, run the test on both the unmodified master branch and your pull request branch (after you have run testreport on both branches). Verify that the output tst_2+2_out.txt file is identical between branches, similar to the above procedure for the file tr_out.txt. If the files differ, attempt to identify and fix what is causing the problem.

5.5.5.3 Automatic testing with Travis-CI

Once your PR is submitted onto GitHub, the continuous integration service Travis-CI runs additional tests on your PR submission. On the 'Pull request' tab in GitHub (https://github.com/MITgcm/MITgcm/pulls), find your pull request; initially you will see a yellow circle to the right of your PR title, indicating testing in progress. Eventually this will change to a green checkmark (pass) or a red X (fail). If you get a red X, click the X and then click on 'Details' to list specifics tests that failed; these can be clicked to produce a screenshot with error messages.

Note that Travis-CI builds documentation (both html and latex) in addition to code testing, so if you have introduced syntax errors into the documentation files, these will be flagged at this stage. Follow the same procedure as above to identify the error messages so the problem(s) can be fixed. Make any appropriate edits to your pull request, re-`git add` and re-`git commit` any newly modified files, re-`git push`. Anytime changes are pushed to the PR, Travis-CI will re-run its tests.

The maintainers will not review your PR until all Travis-CI tests pass.

5.6 Contributing to the manual

Whether you are simply correcting typos or describing undocumented packages, we welcome all contributions to the manual. The following information will help you make sure that your contribution is consistent with the style of the MITgcm documentation. (We know that not all of the current documentation follows these guidelines - we're working on it)

The manual is written in **rst** format, which is short for ReStructuredText directives. rst offers many wonderful features: it automatically does much of the formatting for you, it is reasonably well documented on the web (e.g., primers available here and here), it can accept raw latex syntax and track equation labelling for you, in addition to numerous other useful features. On the down side however, it can be very fussy about formatting, requiring exact spacing and indenting, and seemingly innocuous things such as blank spaces at ends of lines can wreak havoc. We suggest looking at the existing rst files in the manual to see exactly how something is formatted, along with the syntax guidelines specified in this section, prior to writing and formatting your own manual text.

The manual can be viewed either of two ways: interactively (i.e., web-based), as hosted by read-the-docs (https://readthedocs.org/), requiring an html format build, or downloaded as a pdf file. When you have completed your documentation edits, you should double check both versions are to your satisfaction, particularly noting that figure sizing and placement may be rendered differently in the pdf build.

5.6.1 Section headings

- Chapter headings - these are the main headings with integer numbers - underlined with ****
- section headings - headings with number format X.Y - underlined with ====
- Subsection headings - headings with number format X.Y.Z - underlined with ----
- Subsubsection headings - headings with number format X.Y.Z.A - underlined with ~~~~
- Paragraph headings - headings with no numbers - underlined with ^^^^

N.B. all underlinings should be the same length as the heading. If they are too short an error will be produced.

5.6.2 Internal document references

rst allows internal referencing of figures, tables, section headings, and equations, i.e. clickable links that bring the reader to the respective figure etc. in the manual. To be referenced, a unique label is required. To reference figures, tables, or section headings by number, the rst (inline) directive is `:numref:`«LABELNAME»``. For example, this syntax would write out `Figure XX` on a line (assuming «LABELNAME» referred to a figure), and when clicked, would relocate your position in the manual to figure XX. Section headings can also be referenced so that the name is written out instead of the section number, instead using this directive `:ref:`«LABELNAME»``.

Equation references have a slightly different inline syntax: `:eq:`«LABELNAME»`` will produce a clickable equation number reference, surrounded by parentheses.

For instructions how to assign a label to tables and figures, see *below*. To label a section heading, labels go above the section heading they refer to, with the format `.. _«LABELNAME»:`. Note the necessary leading underscore. You can also place a clickable link to *any* spot in the text (e.g., mid-section), using this same syntax to make the label, using the syntax `:ref:`«SOME TEXT TO CLICK ON» <«LABELNAME»>`` for the link.

5.6.3 Citations

In the text, references should be given using the standard "Author(s) (Year)" shorthand followed by a link to the full reference in the manual bibliography. This link is accomplished using the syntax `:cite:`«BIB_REFERENCE»``; this will produce clickable text, usually some variation on the authors' initials or names, surrounded by brackets.

Full references are specified in the file doc/manual_references.bib using standard BibTeX format. Even if unfamiliar with BibTeX, it is relatively easy to add a new reference by simply examining other entries. Furthermore, most publishers provide a means to download BibTex formatted references directly from their website. Note this file is in approximate alphabetic order by author name. For all new references added to the manual, please include a DOI or a URL in addition to journal name, volume and other standard reference infomation. An example JGR journal article reference is reproduced below; note the «BIB_REFERENCE» here is "bryan:79" so the syntax in the rst file format would be "`Bryan and Lewis (1979) :cite:`bryan:79``, which will appear in the manual as Bryan and Lewis (1979) [BL79].

```
@Article{bryan:79,
    author = {Bryan, K. and L.J. Lewis},
    title = {A water mass model of the world ocean},
    journal = jgr,
    volume = 84,
    number = {C5},
    pages = {2503–2517},
    doi = {10.1029/JC084iC05p02503},
```

year = 1979,
}

5.6.4 Other embedded links

Hyperlinks: to reference a (clickable) URL, simply enter the full URL. If you want to have a different, clickable text link instead of displaying the full URL, the syntax is `` `«CLICKABLE TEXT» <«URL»>`_ `` (the '<' and '>' are literal characters, and note the trailing underscore). For this kind of link, the clickable text has to be unique for each URL. If you would like to use non-unique text (like 'click here'), you should use an 'anonymous reference' with a double trailing underscore: `` `«CLICKABLE TEXT» <«URL»>`__ ``.

File references: to create a link to pull up MITgcm code (or any file in the repo) in a code browser window, the syntax is `:filelink:`«PATH/FILENAME»``. If you want to have a different text link to click on (e.g., say you didn't want to display the full path), the syntax is `:filelink:`«CLICKABLE TEXT» <«PATH/FILENAME»>`` (again, the '<' and '>' are literal characters). The top directory here is https://github.com/MITgcm/MITgcm , so if for example you wanted to pop open the file dynamics.F from the main model source directory, you would specify `model/src/dynamics.F` in place of «PATH/FILENAME».

Variable references: to create a link to bring up a webpage displaying all MITgcm repo references to a particular variable name (for this purpose we are using the LXR Cross Referencer), the syntax is `:varlink:`«NAME_OF_VARIABLE»``. This will work on CPP options as well as FORTRAN identifiers (e.g., common block names, subroutine names).

5.6.5 Symbolic Notation

Inline math is done with `:math:`«LATEX_HERE»``

Separate equations, which will be typeset on their own lines, are produced with:

```
.. math::
   «LATEX_HERE»
   :label: «EQN_LABEL_HERE»
```

Labelled separate equations are assigned an equation number, which may be referenced elsewhere in the document (see Section 5.6.2). Omitting the `:label:` above will still produce an equation on its own line, except without an equation label. Note that using latex formatting `\begin{aligned} ... \end{aligned}` across multiple lines of equations will not work in conjunction with unique equation labels for each separate line (any embedded formatting `&` characters will cause errors too). Latex alignment will work however if you assign a single label for the multiple lines of equations.

There is a software tool 'universal document converter' named pandoc that we have found helpful in converting raw latex documents into rst format. To convert a `.tex` file into `.rst`, from a terminal window type:

```
% pandoc -f latex -t rst -o «OUTPUT_FILENAME».rst «INPUT_FILENAME».tex
```

Additional conversion options are available, for example if you have your equations or text in another format; see the pandoc documentation.

Note however we have found that a fair amount of clean-up is still required after conversion, particularly regarding latex equations/labels (pandoc has the unfortunate tendency to add extra spaces, sometimes confusing the rst `:math:` directive, other times creating issues with indentation).

5.6.6 Figures

The syntax to insert a figure is as follows:

```
.. figure:: «PATHNAME/FILENAME».*
   :width: 80%
   :align: center
   :alt: «TEXT DESCRIPTION OF FIGURE HERE»
   :name: «MY_FIGURE_NAME»

   The figure caption goes here as a single line of text.
```

`figure::` The figure file is located in subdirectory `pathname` above; in practice, we have located figure files in subdirectories `figs` off each manual chapter subdirectory. The wild-card `*` is used here so that different file formats can be used in the build process. For vector graphic images, save a `pdf` for the pdf build plus a `svg` file for the html build. For bitmapped images, `gif`, `png`, or `jpeg` formats can be used for both builds, no wild-card necessary, just substitute the actual extension (see here for more info on compatible formats). [Note: A repository for figure source .eps needs to be created]

`:width:` used to scale the size of the figure, here specified as 80% scaling factor (check sizing in both the pdf and html builds, as you may need to adjust the figure size within the pdf file independently).

`:align:` can be right, center, or left.

`:name:` use this name when you refer to the figure in the text, i.e. `:numref:`«MY_FIGURE_NAME»``.

Note the indentation and line spacing employed above.

5.6.7 Tables

There are two syntaxes for tables in reStructuredText. Grid tables are more flexible but cumbersome to create. Simple tables are easy to create but limited (no row spans, etc.). The raw rst syntax is shown first, then the output.

Grid Table Example:

```
+------------+------------+-----------+
| Header 1   | Header 2   | Header 3  |
+============+============+===========+
| body row 1 | column 2   | column 3  |
+------------+------------+-----------+
| body row 2 | Cells may span columns.|
+------------+------------+-----------+
| body row 3 | Cells may  | - Cells   |
+------------+ span rows. | - contain |
| body row 4 |            | - blocks. |
+------------+------------+-----------+
```

Header 1	Header 2	Header 3
body row 1	column 2	column 3
body row 2	Cells may span columns.	
body row 3	Cells may span rows.	• Cells • contain • blocks.
body row 4		

Simple Table Example:

```
=====  =====  ======
   Inputs      Output
-----------  ------
  A      B    A or B
=====  =====  ======
False  False  False
True   False  True
False  True   True
True   True   True
=====  =====  ======
```

Inputs		Output
A	B	A or B
False	False	False
True	False	True
False	True	True
True	True	True

Note that the spacing of your tables in your `.rst` file(s) will not match the generated output; rather, when you build the final output, the rst builder (Sphinx) will determine how wide the columns need to be and space them appropriately.

5.6.8 Other text blocks

Conventionally, we have used the rst 'inline literal' syntax around any literal computer text (commands, labels, literal computer syntax etc.) Surrounding text with double back-quotes `` results in output html `like this`.

To set several lines apart in an whitespace box, e.g. useful for showing lines in from a terminal session, rst uses `::` to set off a 'literal block'. For example:

```
::

    % unix_command_foo
    % unix_command_fum
```

(note the `::` would not appear in the output html or pdf) A splashier way to outline a block, including a box label, is to employ what is termed in rst as an 'admonition block'. In the manual these are used to show calling trees and for describing subroutine inputs and outputs. An example of a subroutine input/output block is as follows:

This is an admonition block showing subroutine in/out syntax

.. admonition:: SUBROUTINE_NAME
:class: note

| $var1$: **VAR1** (WHERE_VAR1_DEFINED.h)
| $var2$: **VAR1** (WHERE_VAR2_DEFINED.h)
| $var3$: **VAR1** (WHERE_VAR3_DEFINED.h)

An example of a subroutine in/out admonition box in the documentation is *here*.

An example of a calling tree in the documentation is *here*.

To show text from a separate file (e.g., to show lines of code, show comments from a Fortran file, show a parameter file etc.), use the `literalinclude` directive. Example usage is shown here:

```
.. literalinclude:: «FILE_TO_SHOW»
    :start-at: String indicating where to start grabbing text
    :end-at: String indicating where to stop grabbing text
```

Unlike the `:filelink:` and `:varlink:` directives, which assume a file path starting at the top of the MITgcm repository, one must specify the path relative to the current directory of the file (for example, from the doc directory, it would require `../../` at the start of the file path to specify the base directory of the MITgcm repository). Note one can instead use `:start-after:` and `:end-before:` to get text from the file between (not including) those lines. If one omits the `start-at` or `start-after`, etc. options the whole file is shown. More details for this directive can be found here. Example usage in this documentation is *here*, where the lines to generate this are:

```
.. literalinclude:: ../../model/src/the_model_main.F
    :start-at: C Invocation from WRAPPER level...
    :end-at: C    |                 :: events.
```

5.6.9 Other style conventions

Units should be typeset in normal text, with a space between a numeric value and the unit, and exponents added with the `:sup:` command.

```
9.8 m/s\ :sup:`2`
```

will produce 9.8 m/s^2. If the exponent is negative use two dashes `--` to make the minus sign sufficiently long. The backslash removes the space between the unit and the exponent. Similarly, for subscripts the command is `:sub:`.

Alternatively, latex `:math:` directives (see *above*) may also be used to display units, using the `\text{}` syntax to display non-italic characters.

- Todo: determine how to break up sections into smaller files
- discuss | lines

5.6.10 Building the manual

Once you've made your changes to the manual, you should build it locally to verify that it works as expected. To do this you will need a working python installation with the following modules installed (use `pip install «MODULE»` in the terminal):

- sphinx
- sphinxcontrib-bibtex
- sphinx_rtd_theme

Once these modules are installed you can build the html version of the manual by running `make html` in the `doc` directory.

To build the pdf version of the manual you will also need a working version of LaTeX that includes several packages that are not always found in minimal LaTeX installations. The command to build the pdf version is `make latexpdf`, which should also be run in the `doc` directory.

5.7 Reviewing pull requests

The only people with write access to the main repository are a small number of core MITgcm developers. They are the people that will eventually merge your pull requests. However, before your PR gets merged, it will undergo the automated testing on Travis-CI, and it will be assessed by the MITgcm community.

Everyone can review and comment on pull requests. Even if you are not one of the core developers you can still comment on a pull request.

To test pull requests locally you should download the pull request branch. You can do this either by cloning the branch from the pull request:

```
git clone -b «THEIR_DEVELOPMENT_BRANCHNAME» https://github.com/«THEIR_GITHUB_
↪USERNAME»/MITgcm.git
```

where «THEIR_GITHUB_USERNAME» is replaced by the username of the person proposing the pull request, and «THEIR_DEVELOPMENT_BRANCHNAME» is the branch from the pull request.

Alternatively, you can add the repository of the user proposing the pull request as a remote to your existing local repository. Navigate to your local repository and type

```
git remote add «THEIR_GITHUB_USERNAME» https://github.com/«THEIR_GITHUB_USERNAME»/
↪MITgcm.git
```

where «THEIR_GITHUB_USERNAME» is replaced by the user name of the person who has made the pull request. Then download their pull request changes

```
git fetch «THEIR_GITHUB_USERNAME»
```

and switch to the desired branch

```
git checkout --track «THEIR_GITHUB_USERNAME»/«THEIR_DEVELOPMENT_BRANCHNAME»
```

You now have a local copy of the code from the pull request and can run tests locally. If you have write access to the main repository you can push fixes or changes directly to the pull request.

None of these steps, apart from pushing fixes back to the pull request, require write access to either the main repository or the repository of the person proposing the pull request. This means that anyone can review pull requests. However, unless you are one of the core developers you won't be able to directly push changes. You will instead have to make a comment describing any problems you find.

CHAPTER 6

Software Architecture

This chapter focuses on describing the **WRAPPER** environment within which both the core numerics and the pluggable packages operate. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The tutorial examples in this manual (see Section 4) contain more succinct, step-by-step instructions on running basic numerical experiments, of various types, both sequentially and in parallel. For many projects, simply starting from an example code and adapting it to suit a particular situation will be all that is required. The first part of this chapter discusses the MITgcm architecture at an abstract level. In the second part of the chapter we described practical details of the MITgcm implementation and the current tools and operating system features that are employed.

6.1 Overall architectural goals

Broadly, the goals of the software architecture employed in MITgcm are three-fold:

- To be able to study a very broad range of interesting and challenging rotating fluids problems;
- The model code should be readily targeted to a wide range of platforms; and
- On any given platform, performance should be comparable to an implementation developed and specialized specifically for that platform.

These points are summarized in Figure 6.1, which conveys the goals of the MITgcm design. The goals lead to a software architecture which at the broadest level can be viewed as consisting of:

1. A core set of numerical and support code. This is discussed in detail in Section 2.
2. A scheme for supporting optional "pluggable" **packages** (containing for example mixed-layer schemes, biogeochemical schemes, atmospheric physics). These packages are used both to overlay alternate dynamics and to introduce specialized physical content onto the core numerical code. An overview of the package scheme is given at the start of Section 8.
3. A support framework called WRAPPER (Wrappable Application Parallel Programming Environment Resource), within which the core numerics and pluggable packages operate.

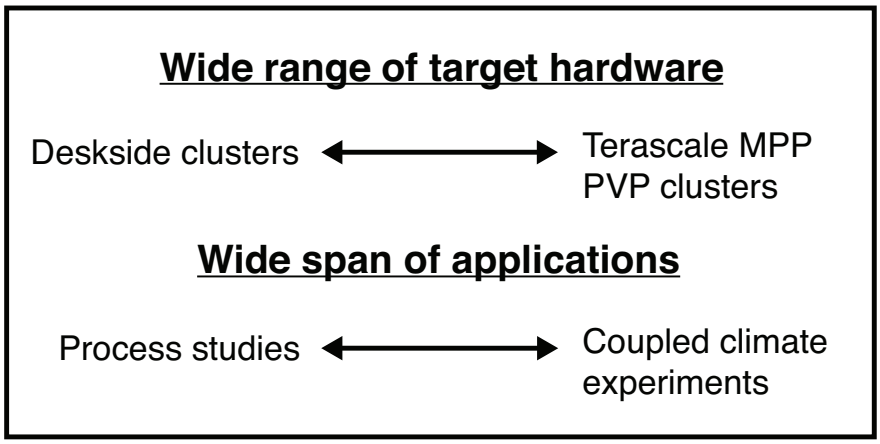

Figure 6.1: The MITgcm architecture is designed to allow simulation of a wide range of physical problems on a wide range of hardware. The computational resource requirements of the applications targeted range from around 10^7 bytes ($\approx$ 10 megabytes) of memory to 10^{11} bytes ($\approx$ 100 gigabytes). Arithmetic operation counts for the applications of interest range from 10^9 floating point operations to more than 10^{17} floating point operations.

This chapter focuses on describing the WRAPPER environment under which both the core numerics and the pluggable packages function. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The "Getting Started" chapter of this manual (Section 3) contains more succinct, step-by-step instructions on running basic numerical experiments both sequentially and in parallel. For many projects simply starting from an example code and adapting it to suit a particular situation will be all that is required.

6.2 WRAPPER

A significant element of the software architecture utilized in MITgcm is a software superstructure and substructure collectively called the WRAPPER (Wrappable Application Parallel Programming Environment Resource). All numerical and support code in MITgcm is written to "fit" within the WRAPPER infrastructure. Writing code to fit within the WRAPPER means that coding has to follow certain, relatively straightforward, rules and conventions (these are discussed further in Section 6.3.1).

The approach taken by the WRAPPER is illustrated in Figure 6.2, which shows how the WRAPPER serves to insulate code that fits within it from architectural differences between hardware platforms and operating systems. This allows numerical code to be easily retargeted.

6.2.1 Target hardware

The WRAPPER is designed to target as broad as possible a range of computer systems. The original development of the WRAPPER took place on a multi-processor, CRAY Y-MP system. On that system, numerical code performance and scaling under the WRAPPER was in excess of that of an implementation that was tightly bound to the CRAY system's proprietary multi-tasking and micro-tasking approach. Later developments have been carried out on uniprocessor and multiprocessor Sun systems with both uniform memory access (UMA) and non-uniform memory access (NUMA) designs. Significant work has also been undertaken on x86 cluster systems, Alpha processor based clustered SMP systems, and on cache-coherent NUMA (CC-NUMA) systems such as Silicon Graphics Altix systems. The MITgcm code, operating within the WRAPPER, is also routinely used on large scale MPP systems (for example, Cray T3E and IBM SP systems). In all cases, numerical code, operating within the WRAPPER, performs and scales very

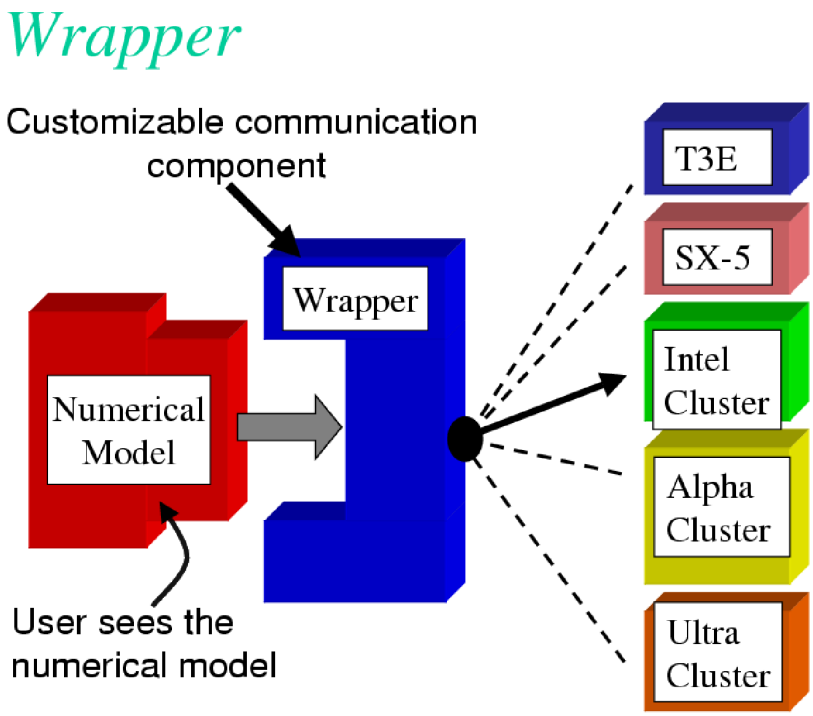

Figure 6.2: Numerical code is written to fit within a software support infrastructure called WRAPPER. The WRAPPER is portable and can be specialized for a wide range of specific target hardware and programming environments, without impacting numerical code that fits within the WRAPPER. Codes that fit within the WRAPPER can generally be made to run as fast on a particular platform as codes specially optimized for that platform.

competitively with equivalent numerical code that has been modified to contain native optimizations for a particular system (see Hoe et al. 1999) [HHA99] .

6.2.2 Supporting hardware neutrality

The different systems mentioned in Section 6.2.1 can be categorized in many different ways. For example, one common distinction is between shared-memory parallel systems (SMP and PVP) and distributed memory parallel systems (for example x86 clusters and large MPP systems). This is one example of a difference between compute platforms that can impact an application. Another common distinction is between vector processing systems with highly specialized CPUs and memory subsystems and commodity microprocessor based systems. There are numerous other differences, especially in relation to how parallel execution is supported. To capture the essential differences between different platforms the WRAPPER uses a *machine model*.

6.2.3 WRAPPER machine model

Applications using the WRAPPER are not written to target just one particular machine (for example an IBM SP2) or just one particular family or class of machines (for example Parallel Vector Processor Systems). Instead the WRAPPER provides applications with an abstract *machine model*. The machine model is very general; however, it can easily be specialized to fit, in a computationally efficient manner, any computer architecture currently available to the scientific computing community.

6.2.4 Machine model parallelism

Codes operating under the WRAPPER target an abstract machine that is assumed to consist of one or more logical processors that can compute concurrently. Computational work is divided among the logical processors by allocating "ownership" to each processor of a certain set (or sets) of calculations. Each set of calculations owned by a particular processor is associated with a specific region of the physical space that is being simulated, and only one processor will be associated with each such region (domain decomposition).

In a strict sense the logical processors over which work is divided do not need to correspond to physical processors. It is perfectly possible to execute a configuration decomposed for multiple logical processors on a single physical processor. This helps ensure that numerical code that is written to fit within the WRAPPER will parallelize with no additional effort. It is also useful for debugging purposes. Generally, however, the computational domain will be subdivided over multiple logical processors in order to then bind those logical processors to physical processor resources that can compute in parallel.

6.2.4.1 Tiles

Computationally, the data structures (e.g., arrays, scalar variables, etc.) that hold the simulated state are associated with each region of physical space and are allocated to a particular logical processor. We refer to these data structures as being **owned** by the processor to which their associated region of physical space has been allocated. Individual regions that are allocated to processors are called **tiles**. A processor can own more than one tile. Figure 6.3 shows a physical domain being mapped to a set of logical processors, with each processor owning a single region of the domain (a single tile). Except for periods of communication and coordination, each processor computes autonomously, working only with data from the tile that the processor owns. If instead multiple tiles were allotted to a single processor, each of these tiles would be computed on independently of the other allotted tiles, in a sequential fashion.

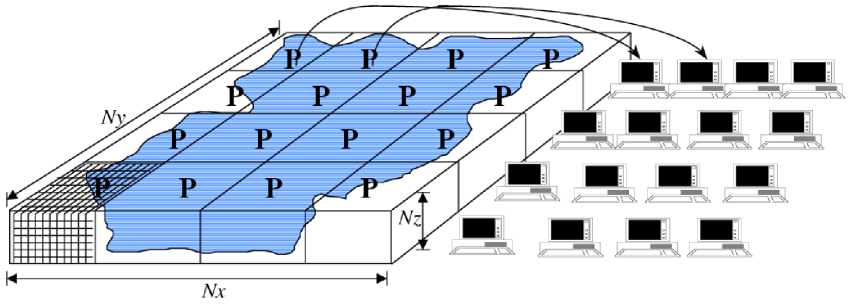

Figure 6.3: The WRAPPER provides support for one and two dimensional decompositions of grid-point domains. The figure shows a hypothetical domain of total size $N_x N_y N_z$. This hypothetical domain is decomposed in two-dimensions along the N_x and N_y directions. The resulting tiles are owned by different processors. The owning processors perform the arithmetic operations associated with a tile. Although not illustrated here, a single processor can own several tiles. Whenever a processor wishes to transfer data between tiles or communicate with other processors it calls a WRAPPER supplied function.

6.2.4.2 Tile layout

Tiles consist of an interior region and an overlap region. The overlap region of a tile corresponds to the interior region of an adjacent tile. In Figure 6.4 each tile would own the region within the black square and hold duplicate information for overlap regions extending into the tiles to the north, south, east and west. During computational phases a processor will reference data in an overlap region whenever it requires values that lie outside the domain it owns. Periodically processors will make calls to WRAPPER functions to communicate data between tiles, in order to keep the overlap regions up to date (see Section 6.2.6). The WRAPPER functions can use a variety of different mechanisms to communicate data between tiles.

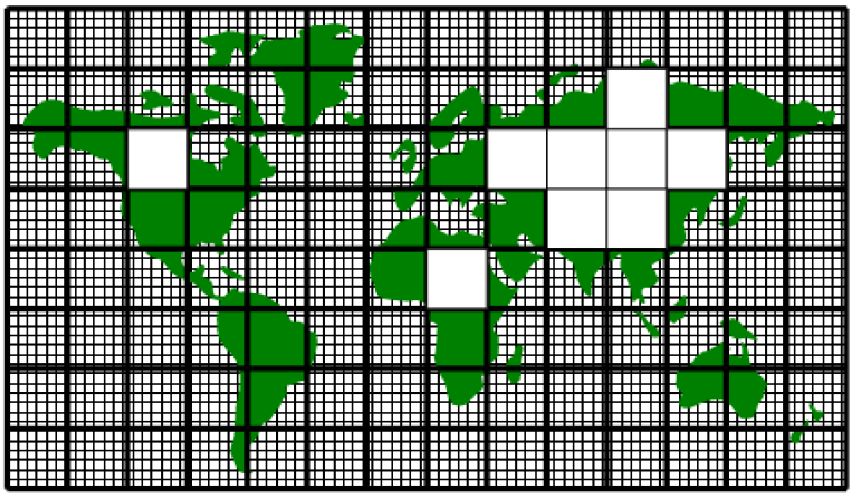

Figure 6.4: A global grid subdivided into tiles. Tiles contain a interior region and an overlap region. Overlap regions are periodically updated from neighboring tiles.

6.2.5 Communication mechanisms

Logical processors are assumed to be able to exchange information between tiles (and between each other) using at least one of two possible mechanisms, shared memory or distributed memory communication. The WRAPPER assumes that communication will use one of these two styles. The underlying hardware and operating system support for the style used is not specified and can vary from system to system.

6.2.5.1 Shared memory communication

Under this mode of communication, data transfers are assumed to be possible using direct addressing of regions of memory. In the WRAPPER shared memory communication model, simple writes to an array can be made to be visible to other CPUs at the application code level. So, as shown below, if one CPU (CPU1) writes the value 8 to element 3 of array a, then other CPUs (here, CPU2) will be able to see the value 8 when they read from a(3). This provides a very low latency and high bandwidth communication mechanism. Thus, in this way one CPU can communicate information to another CPU by assigning a particular value to a particular memory location.

```
  CPU1                      |         CPU2
  ====                      |         ====
                            |
a(3) = 8                    |         WHILE ( a(3) .NE. 8 )
                            |          WAIT
                            |         END WHILE
                            |
```

Under shared communication independent CPUs are operating on the exact same global address space at the application level. This is the model of memory access that is supported at the basic system design level in "shared-memory" systems such as PVP systems, SMP systems, and on distributed shared memory systems (e.g., SGI Origin, SGI Altix, and some AMD Opteron systems). On such systems the WRAPPER will generally use simple read and write statements to access directly application data structures when communicating between CPUs.

In a system where assignments statements map directly to hardware instructions that transport data between CPU and memory banks, this can be a very efficient mechanism for communication. In such case multiple CPUs can communicate simply be reading and writing to agreed locations and following a few basic rules. The latency of this sort of communication is generally not that much higher than the hardware latency of other memory accesses on the system. The bandwidth available between CPUs communicating in this way can be close to the bandwidth of the systems main-memory interconnect. This can make this method of communication very efficient provided it is used appropriately.

Memory consistency

When using shared memory communication between multiple processors, the WRAPPER level shields user applications from certain counter-intuitive system behaviors. In particular, one issue the WRAPPER layer must deal with is a systems memory model. In general the order of reads and writes expressed by the textual order of an application code may not be the ordering of instructions executed by the processor performing the application. The processor performing the application instructions will always operate so that, for the application instructions the processor is executing, any reordering is not apparent. However, machines are often designed so that reordering of instructions is not hidden from other second processors. This means that, in general, even on a shared memory system two processors can observe inconsistent memory values.

The issue of memory consistency between multiple processors is discussed at length in many computer science papers. From a practical point of view, in order to deal with this issue, shared memory machines all provide some mechanism to enforce memory consistency when it is needed. The exact mechanism employed will vary between systems. For communication using shared memory, the WRAPPER provides a place to invoke the appropriate mechanism to ensure memory consistency for a particular platform.

Cache effects and false sharing

Shared-memory machines often have local-to-processor memory caches which contain mirrored copies of main memory. Automatic cache-coherence protocols are used to maintain consistency between caches on different processors. These cache-coherence protocols typically enforce consistency between regions of memory with large granularity (typically 128 or 256 byte chunks). The coherency protocols employed can be expensive relative to other memory accesses and so care is taken in the WRAPPER (by padding synchronization structures appropriately) to avoid unnecessary coherence traffic.

Operating system support for shared memory

Applications running under multiple threads within a single process can use shared memory communication. In this case *all* the memory locations in an application are potentially visible to all the compute threads. Multiple threads operating within a single process is the standard mechanism for supporting shared memory that the WRAPPER utilizes. Configuring and launching code to run in multi-threaded mode on specific platforms is discussed in Section 6.3.2.1. However, on many systems, potentially very efficient mechanisms for using shared memory communication between multiple processes (in contrast to multiple threads within a single process) also exist. In most cases this works by making a limited region of memory shared between processes. The MMAP and IPC facilities in UNIX systems provide this capability as do vendor specific tools like LAPI and IMC. Extensions exist for the WRAPPER that allow these mechanisms to be used for shared memory communication. However, these mechanisms are not distributed with the default WRAPPER sources, because of their proprietary nature.

6.2.5.2 Distributed memory communication

Under this mode of communication there is no mechanism, at the application code level, for directly addressing regions of memory owned and visible to another CPU. Instead a communication library must be used, as illustrated below. If one CPU (here, CPU1) writes the value 8 to element 3 of array a, then at least one of CPU1 and/or CPU2 will need to call a function in the API of the communication library to communicate data from a tile that it owns to a tile that another CPU owns. By default the WRAPPER binds to the MPI communication library for this style of communication (see https://computing.llnl.gov/tutorials/mpi/ for more information about the MPI Standard).

```
  CPU1                        |         CPU2
  ====                        |         ====
                              |
a(3) = 8                      |         WHILE ( a(3) .NE. 8 )
CALL SEND( CPU2,a(3) )        |          CALL RECV( CPU1, a(3) )
                              |         END WHILE
                              |
```

Many parallel systems are not constructed in a way where it is possible or practical for an application to use shared memory for communication. For cluster systems consisting of individual computers connected by a fast network, there is no notion of shared memory at the system level. For this sort of system the WRAPPER provides support for communication based on a bespoke communication library. The default communication library used is MPI. It is relatively straightforward to implement bindings to optimized platform specific communication libraries. For example the work described in Hoe et al. (1999) [HHA99] substituted standard MPI communication for a highly optimized library.

6.2.6 Communication primitives

Optimized communication support is assumed to be potentially available for a small number of communication operations. It is also assumed that communication performance optimizations can be achieved by optimizing a small number of communication primitives. Three optimizable primitives are provided by the WRAPPER.

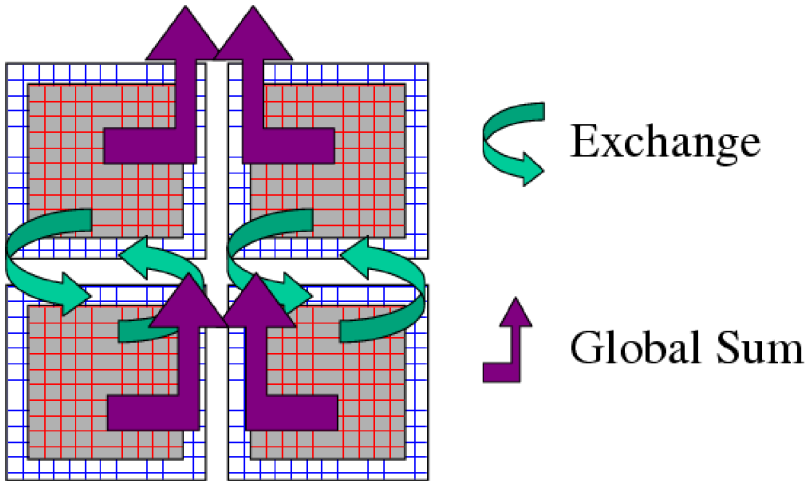

Figure 6.5: Three performance critical parallel primitives are provided by the WRAPPER. These primitives are always used to communicate data between tiles. The figure shows four tiles. The curved arrows indicate exchange primitives which transfer data between the overlap regions at tile edges and interior regions for nearest-neighbor tiles. The straight arrows symbolize global sum operations which connect all tiles. The global sum operation provides both a key arithmetic primitive and can serve as a synchronization primitive. A third barrier primitive is also provided, which behaves much like the global sum primitive.

- **EXCHANGE** This operation is used to transfer data between interior and overlap regions of neighboring tiles. A number of different forms of this operation are supported. These different forms handle:
  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.
  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.
  - Transformation operations required when transporting data between different grid regions. Transferring data between faces of a cube-sphere grid, for example, involves a rotation of vector components.
  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.
- **GLOBAL SUM** The global sum operation is a central arithmetic operation for the pressure inversion phase of the MITgcm algorithm. For certain configurations, scaling can be highly sensitive to the performance of the global sum primitive. This operation is a collective operation involving all tiles of the simulated domain. Different forms of the global sum primitive exist for handling:
  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.
  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.
  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.
- **BARRIER** The WRAPPER provides a global synchronization function called barrier. This is used to synchronize computations over all tiles. The **BARRIER** and **GLOBAL SUM** primitives have much in common and in some cases use the same underlying code.

6.2.7 Memory architecture

The WRAPPER machine model is aimed to target efficient systems with highly pipelined memory architectures and systems with deep memory hierarchies that favor memory reuse. This is achieved by supporting a flexible tiling strategy as shown in Figure 6.6. Within a CPU, computations are carried out sequentially on each tile in turn. By reshaping tiles according to the target platform it is possible to automatically tune code to improve memory performance. On a vector machine a given domain might be subdivided into a few long, thin regions. On a commodity microprocessor based system, however, the same region could be simulated use many more smaller sub-domains.

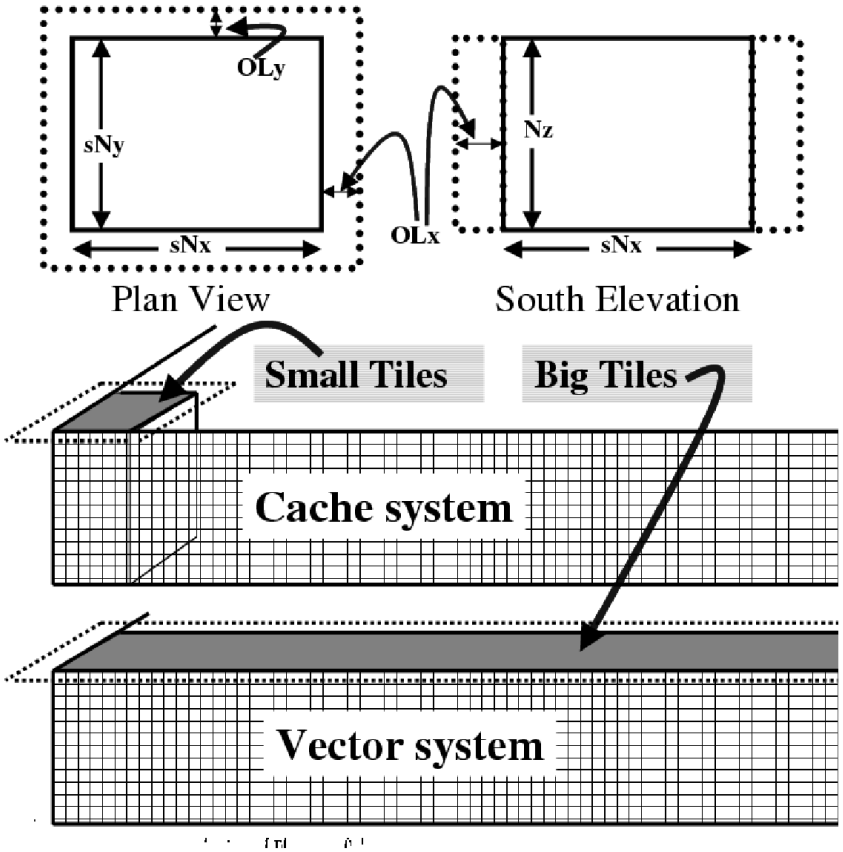

Figure 6.6: The tiling strategy that the WRAPPER supports allows tiles to be shaped to suit the underlying system memory architecture. Compact tiles that lead to greater memory reuse can be used on cache based systems (upper half of figure) with deep memory hierarchies, whereas long tiles with large inner loops can be used to exploit vector systems having highly pipelined memory systems.

6.2.8 Summary

Following the discussion above, the machine model that the WRAPPER presents to an application has the following characteristics:

- The machine consists of one or more logical processors.
- Each processor operates on tiles that it owns.
- A processor may own more than one tile.

- Processors may compute concurrently.
- Exchange of information between tiles is handled by the machine (WRAPPER) not by the application.

Behind the scenes this allows the WRAPPER to adapt the machine model functions to exploit hardware on which:

- Processors may be able to communicate very efficiently with each other using shared memory.
- An alternative communication mechanism based on a relatively simple interprocess communication API may be required.
- Shared memory may not necessarily obey sequential consistency, however some mechanism will exist for enforcing memory consistency.
- Memory consistency that is enforced at the hardware level may be expensive. Unnecessary triggering of consistency protocols should be avoided.
- Memory access patterns may need to be either repetitive or highly pipelined for optimum hardware performance.

This generic model, summarized in Figure 6.7, captures the essential hardware ingredients of almost all successful scientific computer systems designed in the last 50 years.

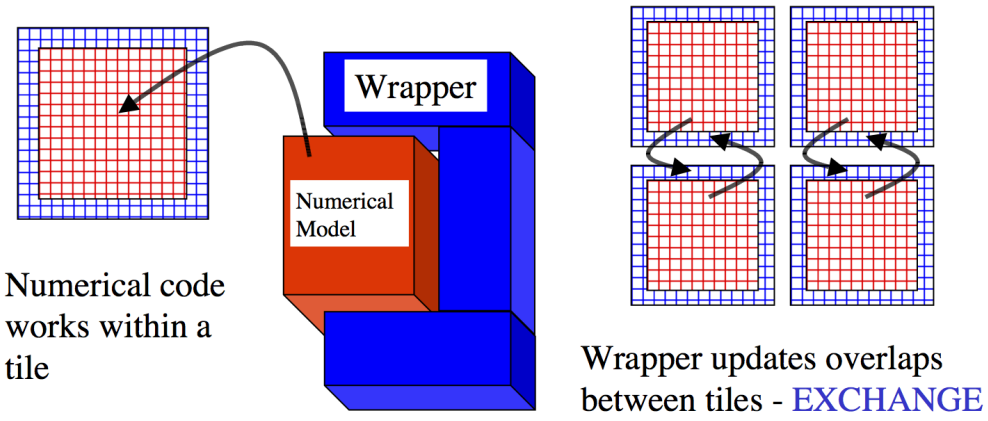

Figure 6.7: Summary of the WRAPPER machine model.

6.3 Using the WRAPPER

In order to support maximum portability the WRAPPER is implemented primarily in sequential Fortran 77. At a practical level the key steps provided by the WRAPPER are:

1. specifying how a domain will be decomposed
2. starting a code in either sequential or parallel modes of operations
3. controlling communication between tiles and between concurrently computing CPUs.

This section describes the details of each of these operations. Section 6.3.1 explains the way a domain is decomposed (or composed) is expressed. Section 6.3.2 describes practical details of running codes in various different parallel modes on contemporary computer systems. Section 6.3.3 explains the internal information that the WRAPPER uses to control how information is communicated between tiles.

6.3.1 Specifying a domain decomposition

At its heart, much of the WRAPPER works only in terms of a collection of tiles which are interconnected to each other. This is also true of application code operating within the WRAPPER. Application code is written as a series of compute operations, each of which operates on a single tile. If application code needs to perform operations involving data associated with another tile, it uses a WRAPPER function to obtain that data. The specification of how a global domain is constructed from tiles or alternatively how a global domain is decomposed into tiles is made in the file SIZE.h. This file defines the following parameters:

File: model/inc/SIZE.h

Parameter: sNx, sNx
Parameter: OLx, OLy
Parameter: nSx, nSy
Parameter: nPx, nPy

Together these parameters define a tiling decomposition of the style shown in Figure 6.8. The parameters `sNx` and `sNx` define the size of an individual tile. The parameters `OLx` and `OLy` define the maximum size of the overlap extent. This must be set to the maximum width of the computation stencil that the numerical code finite-difference operations require between overlap region updates. The maximum overlap required by any of the operations in the MITgcm code distributed at this time is four grid points (some of the higher-order advection schemes require a large overlap region). Code modifications and enhancements that involve adding wide finite-difference stencils may require increasing `OLx` and `OLy`. Setting `OLx` and `OLy` to a too large value will decrease code performance (because redundant computations will be performed), however it will not cause any other problems.

The parameters `nSx` and `nSy` specify the number of tiles that will be created within a single process. Each of these tiles will have internal dimensions of `sNx` and `sNy`. If, when the code is executed, these tiles are allocated to different threads of a process that are then bound to different physical processors (see the multi-threaded execution discussion in Section 6.3.2), then computation will be performed concurrently on each tile. However, it is also possible to run the same decomposition within a process running a single thread on a single processor. In this case the tiles will be computed over sequentially. If the decomposition is run in a single process running multiple threads but attached to a single physical processor, then, in general, the computation for different tiles will be interleaved by system level software. This too is a valid mode of operation.

The parameters `sNx`, `sNy`, `OLx`, `OLy`, `nSx` and `nSy` are used extensively by numerical code. The settings of `sNx`, `sNy`, `OLx`, and `OLy` are used to form the loop ranges for many numerical calculations and to provide dimensions for arrays holding numerical state. The `nSx` and `nSy` are used in conjunction with the thread number parameter `myThid`. Much of the numerical code operating within the WRAPPER takes the form:

```
DO bj=myByLo(myThid),myByHi(myThid)
 DO bi=myBxLo(myThid),myBxHi(myThid)
    :
    a block of computations ranging
    over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
    :
 ENDDO
ENDDO

communication code to sum a number or maybe update
tile overlap regions

DO bj=myByLo(myThid),myByHi(myThid)
 DO bi=myBxLo(myThid),myBxHi(myThid)
```

(continues on next page)

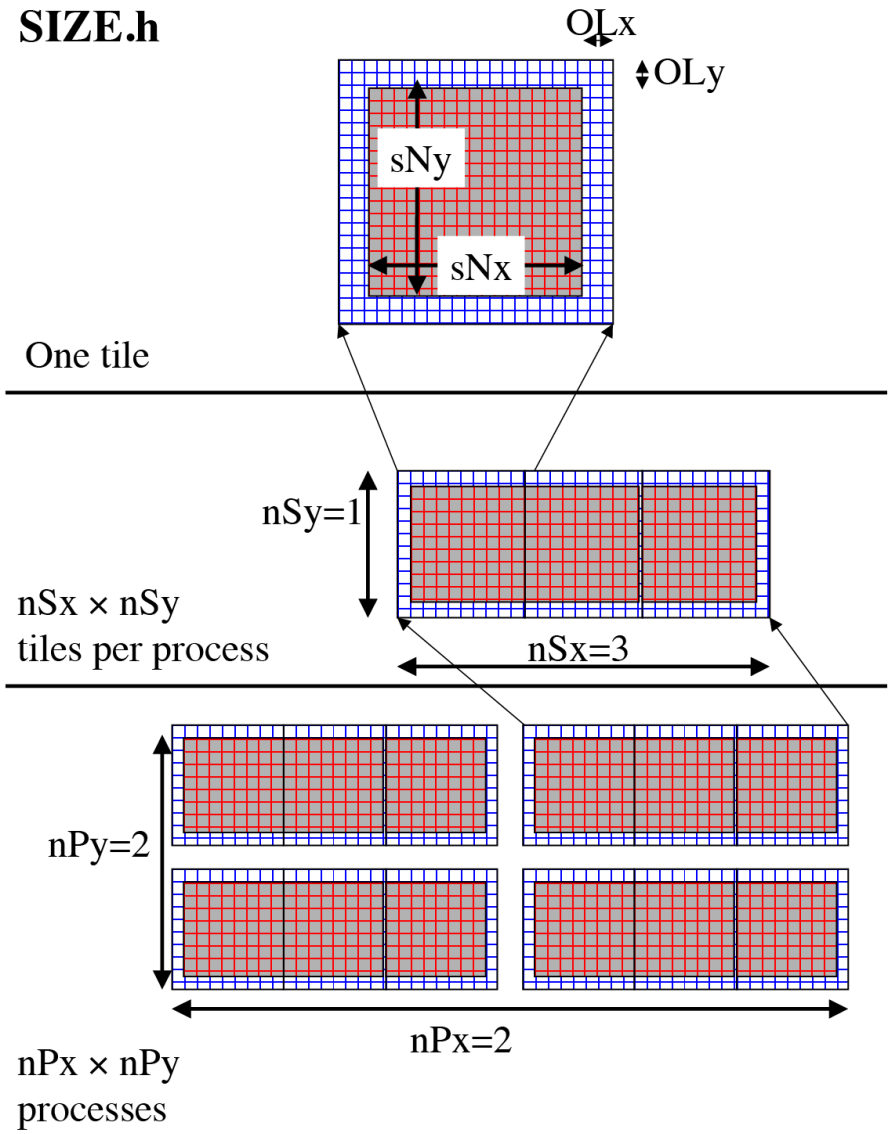

Figure 6.8: The three level domain decomposition hierarchy employed by the WRAPPER. A domain is composed of tiles. Multiple tiles can be allocated to a single process. Multiple processes can exist, each with multiple tiles. Tiles within a process can be spread over multiple compute threads.

(continued from previous page)

```
    :
    another block of computations ranging
    over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
    :
 ENDDO
ENDDO
```

The variables `myBxLo(myThid)`, `myBxHi(myThid)`, `myByLo(myThid)` and `myByHi(myThid)` set the bounds of the loops in `bi` and `bj` in this schematic. These variables specify the subset of the tiles in the range `1, nSx` and `1, nSy1` that the logical processor bound to thread number `myThid` owns. The thread number variable `myThid` ranges from 1 to the total number of threads requested at execution time. For each value of `myThid` the loop scheme above will step sequentially through the tiles owned by that thread. However, different threads will have different ranges of tiles assigned to them, so that separate threads can compute iterations of the `bi`, `bj` loop concurrently. Within a `bi`, `bj` loop, computation is performed concurrently over as many processes and threads as there are physical processors available to compute.

An exception to the the use of `bi` and `bj` in loops arises in the exchange routines used when the *exch2 package* is used with the cubed sphere. In this case `bj` is generally set to 1 and the loop runs from `1, bi`. Within the loop `bi` is used to retrieve the tile number, which is then used to reference exchange parameters.

The amount of computation that can be embedded in a single loop over `bi` and `bj` varies for different parts of the MITgcm algorithm. Consider a code extract from the 2-D implicit elliptic solver:

```
REAL*8  cg2d_r(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
REAL*8  err
    :
    :
  other computations
    :
    :
err = 0.
DO bj=myByLo(myThid),myByHi(myThid)
 DO bi=myBxLo(myThid),myBxHi(myThid)
  DO J=1,sNy
   DO I=1,sNx
     err = err + cg2d_r(I,J,bi,bj)*cg2d_r(I,J,bi,bj)
   ENDDO
  ENDDO
 ENDDO
ENDDO

CALL GLOBAL_SUM_R8( err   , myThid )
err = SQRT(err)
```

This portion of the code computes the L_2-Norm of a vector whose elements are held in the array `cg2d_r`, writing the final result to scalar variable `err`. Notice that under the WRAPPER, arrays such as cg2d_r have two extra trailing dimensions. These right most indices are tile indexes. Different threads with a single process operate on different ranges of tile index, as controlled by the settings of `myByLo(myThid)`, `myByHi(myThid)`, `myBxLo(myThid)` and `myBxHi(myThid)`. Because the L_2-Norm requires a global reduction, the `bi`, `bj` loop above only contains one statement. This computation phase is then followed by a communication phase in which all threads and processes must participate. However, in other areas of the MITgcm, code entries subsections of code are within a single `bi`, `bj` loop. For example the evaluation of all the momentum equation prognostic terms (see dynamics.F) is within a single `bi`, `bj` loop.

The final decomposition parameters are `nPx` and `nPy`. These parameters are used to indicate to the WRAPPER level how many processes (each with `nSx`×`nSy` tiles) will be used for this simulation. This information is needed during

initialization and during I/O phases. However, unlike the variables sNx, sNy, OLx, OLy, nSx and nSy the values of nPx and nPy are absent from the core numerical and support code.

6.3.1.1 Examples of SIZE.h specifications

The following different SIZE.h parameter setting illustrate how to interpret the values of sNx, sNy, OLx, OLy, nSx, nSy, nPx and nPy.

1.
```
 PARAMETER (
&           sNx =  90,
&           sNy =  40,
&           OLx =   3,
&           OLy =   3,
&           nSx =   1,
&           nSy =   1,
&           nPx =   1,
&           nPy =   1)
```

   This sets up a single tile with *x*-dimension of ninety grid points, *y*-dimension of forty grid points, and *x* and *y* overlaps of three grid points each.

2.
```
 PARAMETER (
&           sNx =  45,
&           sNy =  20,
&           OLx =   3,
&           OLy =   3,
&           nSx =   1,
&           nSy =   1,
&           nPx =   2,
&           nPy =   2)
```

   This sets up tiles with *x*-dimension of forty-five grid points, *y*-dimension of twenty grid points, and *x* and *y* overlaps of three grid points each. There are four tiles allocated to four separate processes (nPx=2, nPy=2) and arranged so that the global domain size is again ninety grid points in *x* and forty grid points in *y*. In general the formula for global grid size (held in model variables Nx and Ny) is

```
Nx  = sNx*nSx*nPx
Ny  = sNy*nSy*nPy
```

3.
```
 PARAMETER (
&           sNx =  90,
&           sNy =  10,
&           OLx =   3,
&           OLy =   3,
&           nSx =   1,
&           nSy =   2,
&           nPx =   1,
&           nPy =   2)
```

   This sets up tiles with *x*-dimension of ninety grid points, *y*-dimension of ten grid points, and *x* and *y* overlaps of three grid points each. There are four tiles allocated to two separate processes (nPy=2) each of which has two separate sub-domains nSy=2. The global domain size is again ninety grid points in *x* and forty grid points in *y*. The two sub-domains in each process will be computed sequentially if they are given to a single thread within a single process. Alternatively if the code is invoked with multiple threads per process the two domains in y may be computed concurrently.

4.
```
 PARAMETER (
&           sNx =  32,
&           sNy =  32,
&           OLx =   3,
&           OLy =   3,
&           nSx =   6,
&           nSy =   1,
&           nPx =   1,
&           nPy =   1)
```

This sets up tiles with x-dimension of thirty-two grid points, y-dimension of thirty-two grid points, and x and y overlaps of three grid points each. There are six tiles allocated to six separate logical processors (`nSx=6`). This set of values can be used for a cube sphere calculation. Each tile of size 32×32 represents a face of the cube. Initializing the tile connectivity correctly (see Section 6.3.3.3. allows the rotations associated with moving between the six cube faces to be embedded within the tile-tile communication code.

6.3.2 Starting the code

When code is started under the WRAPPER, execution begins in a main routine eesupp/src/main.F that is owned by the WRAPPER. Control is transferred to the application through a routine called model/src/the_model_main.F once the WRAPPER has initialized correctly and has created the necessary variables to support subsequent calls to communication routines by the application code. The main stages of the WRAPPER startup calling sequence are as follows:

```
MAIN
|
|--EEBOOT               :: WRAPPER initialization
|  |
|  |-- EEBOOT_MINMAL    :: Minimal startup. Just enough to
|  |                       allow basic I/O.
|  |-- EEINTRO_MSG      :: Write startup greeting.
|  |
|  |-- EESET_PARMS      :: Set WRAPPER parameters
|  |
|  |-- EEWRITE_EEENV    :: Print WRAPPER parameter settings
|  |
|  |-- INI_PROCS        :: Associate processes with grid regions.
|  |
|  |-- INI_THREADING_ENVIRONMENT   :: Associate threads with grid regions.
|       |
|       |--INI_COMMUNICATION_PATTERNS :: Initialize between tile
|                                     :: communication data structures
|
|
|--CHECK_THREADS    :: Validate multiple thread start up.
|
|--THE_MODEL_MAIN   :: Numerical code top-level driver routine
```

The steps above preceeds transfer of control to application code, which occurs in the procedure the_main_model.F

6.3.2.1 Multi-threaded execution

Prior to transferring control to the procedure the_main_model.F the WRAPPER may cause several coarse grain threads to be initialized. The routine the_main_model.F is called once for each thread and is passed a single stack argument

which is the thread number, stored in the myThid. In addition to specifying a decomposition with multiple tiles per process (see Section 6.3.1) configuring and starting a code to run using multiple threads requires the following steps.

Compilation

First the code must be compiled with appropriate multi-threading directives active in the file eesupp/src/main.F and with appropriate compiler flags to request multi-threading support. The header files eesupp/inc/MAIN_PDIRECTIVES1.h and eesupp/inc/MAIN_PDIRECTIVES2.h contain directives compatible with compilers for Sun, Compaq, SGI, Hewlett-Packard SMP systems and CRAY PVP systems. These directives can be activated by using compile time directives `-DTARGET_SUN`, `-DTARGET_DEC`, `-DTARGET_SGI`, `-DTARGET_HP` or `-DTARGET_CRAY_VECTOR` respectively. Compiler options for invoking multi-threaded compilation vary from system to system and from compiler to compiler. The options will be described in the individual compiler documentation. For the Fortran compiler from Sun the following options are needed to correctly compile multi-threaded code

```
-stackvar -explicitpar -vpara -noautopar
```

These options are specific to the Sun compiler. Other compilers will use different syntax that will be described in their documentation. The effect of these options is as follows:

1. **-stackvar** Causes all local variables to be allocated in stack storage. This is necessary for local variables to ensure that they are private to their thread. Note, when using this option it may be necessary to override the default limit on stack-size that the operating system assigns to a process. This can normally be done by changing the settings of the command shell's `stack-size`. However, on some systems changing this limit will require privileged administrator access to modify system parameters.
2. **-explicitpar** Requests that multiple threads be spawned in response to explicit directives in the application code. These directives are inserted with syntax appropriate to the particular target platform when, for example, the `-DTARGET_SUN` flag is selected.
3. **-vpara** This causes the compiler to describe the multi-threaded configuration it is creating. This is not required but it can be useful when troubleshooting.
4. **-noautopar** This inhibits any automatic multi-threaded parallelization the compiler may otherwise generate.

An example of valid settings for the `eedata` file for a domain with two subdomains in *y* and running with two threads is shown below

```
nTx=1,nTy=2
```

This set of values will cause computations to stay within a single thread when moving across the `nSx` sub-domains. In the *y*-direction, however, sub-domains will be split equally between two threads.

Despite its appealing programming model, multi-threaded execution remains less common than multi-process execution (described in Section 6.3.2.2). One major reason for this is that many system libraries are still not "thread-safe". This means that, for example, on some systems it is not safe to call system routines to perform I/O when running in multi-threaded mode (except, perhaps, in a limited set of circumstances). Another reason is that support for multi-threaded programming models varies between systems.

6.3.2.2 Multi-process execution

Multi-process execution is more ubiquitous than multi-threaded execution. In order to run code in a multi-process configuration, a decomposition specification (see Section 6.3.1) is given (in which at least one of the parameters `nPx` or `nPy` will be greater than one). Then, as for multi-threaded operation, appropriate compile time and run time steps must be taken.

Compilation

Multi-process execution under the WRAPPER assumes that portable, MPI libraries are available for controlling the start-up of multiple processes. The MPI libraries are not required, although they are usually used, for performance critical communication. However, in order to simplify the task of controlling and coordinating the start up of a large number (hundreds and possibly even thousands) of copies of the same program, MPI is used. The calls to the MPI multi-process startup routines must be activated at compile time. Currently MPI libraries are invoked by specifying the appropriate options file with the `-of` flag when running the genmake2 script, which generates the Makefile for compiling and linking MITgcm. (Previously this was done by setting the `ALLOW_USE_MPI` and `ALWAYS_USE_MPI` flags in the CPP_EEOPTIONS.h file.) More detailed information about the use of genmake2 for specifying local compiler flags is located in Section 3.5.2.

Execution

The mechanics of starting a program in multi-process mode under MPI is not standardized. Documentation associated with the distribution of MPI installed on a system will describe how to start a program using that distribution. For the open-source MPICH system, the MITgcm program can be started using a command such as

```
mpirun -np 64 -machinefile mf ./mitgcmuv
```

In this example the text `-np 64` specifies the number of processes that will be created. The numeric value 64 must be equal to (or greater than) the product of the processor grid settings of `nPx` and `nPy` in the file SIZE.h. The option `-machinefile mf` specifies that a text file called `mf` will be read to get a list of processor names on which the sixty-four processes will execute. The syntax of this file is specified by the MPI distribution.

6.3.2.3 Environment variables

On some systems multi-threaded execution also requires the setting of a special environment variable. On many machines this variable is called `PARALLEL` and its values should be set to the number of parallel threads required. Generally the help or manual pages associated with the multi-threaded compiler on a machine will explain how to set the required environment variables.

6.3.2.4 Runtime input parameters

Finally the file `eedata` needs to be configured to indicate the number of threads to be used in the *x* and *y* directions:

```
# Example "eedata" file
# Lines beginning "#" are comments
# nTx - No. threads per process in X
# nTy - No. threads per process in Y
 &EEPARMS
 nTx=1,
 nTy=1,
 &
```

The product of `nTx` and `nTy` must be equal to the number of threads spawned, i.e., the setting of the environment variable `PARALLEL`. The value of `nTx` must subdivide the number of sub-domains in *x* (`nSx`) exactly. The value of `nTy` must subdivide the number of sub-domains in *y* (`nSy`) exactly. The multi-process startup of the MITgcm executable `mitgcmuv` is controlled by the routines eeboot_minimal.F and ini_procs.F. The first routine performs basic steps required to make sure each process is started and has a textual output stream associated with it. By default two output files are opened for each process with names `STDOUT.NNNN` and `STDERR.NNNN`. The *NNNNN* part of the name is filled in with the process number so that process number 0 will create output files `STDOUT.0000` and

STDERR.0000, process number 1 will create output files STDOUT.0001 and STDERR.0001, etc. These files are used for reporting status and configuration information and for reporting error conditions on a process-by-process basis. The eeboot_minimal.F procedure also sets the variables myProcId and MPI_COMM_MODEL. These variables are related to processor identification and are used later in the routine ini_procs.F to allocate tiles to processes.

Allocation of processes to tiles is controlled by the routine ini_procs.F. For each process this routine sets the variables myXGlobalLo and myYGlobalLo. These variables specify, in index space, the coordinates of the southernmost and westernmost corner of the southernmost and westernmost tile owned by this process. The variables pidW, pidE, pidS and pidN are also set in this routine. These are used to identify processes holding tiles to the west, east, south and north of a given process. These values are stored in global storage in the header file EESUPPORT.h for use by communication routines. The above does not hold when the *exch2 package* is used. The *exch2 package* sets its own parameters to specify the global indices of tiles and their relationships to each other. See the documentation on the *exch2 package* for details.

6.3.3 Controlling communication

The WRAPPER maintains internal information that is used for communication operations and can be customized for different platforms. This section describes the information that is held and used.

1. **Tile-tile connectivity information** For each tile the WRAPPER sets a flag that sets the tile number to the north, south, east and west of that tile. This number is unique over all tiles in a configuration. Except when using the cubed sphere and the *exch2 package*, the number is held in the variables tileNo (this holds the tiles own number), tileNoN, tileNoS, tileNoE and tileNoW. A parameter is also stored with each tile that specifies the type of communication that is used between tiles. This information is held in the variables tileCommModeN, tileCommModeS, tileCommModeE and tileCommModeW. This latter set of variables can take one of the following values COMM_NONE, COMM_MSG, COMM_PUT and COMM_GET. A value of COMM_NONE is used to indicate that a tile has no neighbor to communicate with on a particular face. A value of COMM_MSG is used to indicate that some form of distributed memory communication is required to communicate between these tile faces (see Section 6.2.5.2). A value of COMM_PUT or COMM_GET is used to indicate forms of shared memory communication (see Section 6.2.5.1). The COMM_PUT value indicates that a CPU should communicate by writing to data structures owned by another CPU. A COMM_GET value indicates that a CPU should communicate by reading from data structures owned by another CPU. These flags affect the behavior of the WRAPPER exchange primitive (see Figure 6.5). The routine ini_communication_patterns.F is responsible for setting the communication mode values for each tile.

   When using the cubed sphere configuration with the *exch2 package*, the relationships between tiles and their communication methods are set by the *exch2 package* and stored in different variables. See the *exch2 package* documentation for details.

2. **MP directives** The WRAPPER transfers control to numerical application code through the routine the_model_main.F. This routine is called in a way that allows for it to be invoked by several threads. Support for this is based on either multi-processing (MP) compiler directives or specific calls to multi-threading libraries (e.g., POSIX threads). Most commercially available Fortran compilers support the generation of code to spawn multiple threads through some form of compiler directives. Compiler directives are generally more convenient than writing code to explicitly spawn threads. On some systems, compiler directives may be the only method available. The WRAPPER is distributed with template MP directives for a number of systems.

   These directives are inserted into the code just before and after the transfer of control to numerical algorithm code through the routine the_model_main.F. An example of the code that performs this process for a Silicon Graphics system is as follows:

```
C--
C--  Parallel directives for MIPS Pro Fortran compiler
C--
C      Parallel compiler directives for SGI with IRIX
C$PAR  PARALLEL DO
C$PAR&  CHUNK=1,MP_SCHEDTYPE=INTERLEAVE,
C$PAR&  SHARE(nThreads),LOCAL(myThid,I)
C
      DO I=1,nThreads
        myThid = I

C--     Invoke nThreads instances of the numerical model
        CALL THE_MODEL_MAIN(myThid)

      ENDDO
```

Prior to transferring control to the procedure the_model_main.F the WRAPPER may use MP directives to spawn multiple threads. This code is extracted from the files main.F and eesupp/inc/MAIN_PDIRECTIVES1.h. The variable nThreads specifies how many instances of the routine the_model_main.F will be created. The value of nThreads is set in the routine ini_threading_environment.F. The value is set equal to the the product of the parameters nTx and nTy that are read from the file `eedata`. If the value of nThreads is inconsistent with the number of threads requested from the operating system (for example by using an environment variable as described in Section 6.3.2.1) then usually an error will be reported by the routine check_threads.F.

3. **memsync flags** As discussed in Section 6.2.5.1, a low-level system function may be need to force memory consistency on some shared memory systems. The routine memsync.F is used for this purpose. This routine should not need modifying and the information below is only provided for completeness. A logical parameter exchNeedsMemSync set in the routine ini_communication_patterns.F controls whether the memsync.F primitive is called. In general this routine is only used for multi-threaded execution. The code that goes into the memsync.F routine is specific to the compiler and processor used. In some cases, it must be written using a short code snippet of assembly language. For an Ultra Sparc system the following code snippet is used

```
asm("membar #LoadStore|#StoreStore");
```

For an Alpha based system the equivalent code reads

```
asm("mb");
```

while on an x86 system the following code is required

```
asm("lock; addl $0,0(%%esp)": : :"memory")
```

4. **Cache line size** As discussed in Section 6.2.5.1, multi-threaded codes explicitly avoid penalties associated with excessive coherence traffic on an SMP system. To do this the shared memory data structures used by the global_sum.F, global_max.F and barrier.F routines are padded. The variables that control the padding are set in the header file EEPARAMS.h. These variables are called cacheLineSize, lShare1, lShare4 and lShare8. The default values should not normally need changing.

5. **_BARRIER** This is a CPP macro that is expanded to a call to a routine which synchronizes all the logical processors running under the WRAPPER. Using a macro here preserves flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure barrier.F. The default setting for the _BARRIER macro is given in the file CPP_EEMACROS.h.

6. **_GSUM** This is a CPP macro that is expanded to a call to a routine which sums up a floating point number over all the logical processors running under the WRAPPER. Using a macro here provides extra flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure GLOBAL_SUM_R8() for 64-bit floating point operands or GLOBAL_SUM_R4() for 32-bit floating point operand (located in file global_sum.F). The default setting for the _GSUM macro is given in the file CPP_EEMACROS.h. The _GSUM macro is a performance critical operation, especially for large processor count, small tile size configurations. The custom communication example discussed in Section 6.3.3.2 shows how the macro is used to invoke a custom global sum routine for a specific set of hardware.

7. **_EXCH** The _EXCH CPP macro is used to update tile overlap regions. It is qualified by a suffix indicating whether overlap updates are for two-dimensional (_EXCH_XY) or three dimensional (_EXCH_XYZ) physical fields and whether fields are 32-bit floating point (_EXCH_XY_R4, _EXCH_XYZ_R4) or 64-bit floating point (_EXCH_XY_R8, _EXCH_XYZ_R8). The macro mappings are defined in the header file CPP_EEMACROS.h. As with _GSUM, the _EXCH operation plays a crucial role in scaling to small tile, large logical and physical processor count configurations. The example in Section 6.3.3.2 discusses defining an optimized and specialized form on the _EXCH operation.

   The _EXCH operation is also central to supporting grids such as the cube-sphere grid. In this class of grid a rotation may be required between tiles. Aligning the coordinate requiring rotation with the tile decomposition allows the coordinate transformation to be embedded within a custom form of the _EXCH primitive. In these cases _EXCH is mapped to exch2 routines, as detailed in the *exch2 package* documentation.

8. **Reverse Mode** The communication primitives _EXCH and _GSUM both employ hand-written adjoint forms (or reverse mode) forms. These reverse mode forms can be found in the source code directory pkg/autodiff. For the global sum primitive the reverse mode form calls are to GLOBAL_ADSUM_R4() and GLOBAL_ADSUM_R8() (located in file global_sum_ad.F). The reverse mode form of the exchange primitives are found in routines prefixed ADEXCH. The exchange routines make calls to the same low-level communication primitives as the forward mode operations. However, the routine argument theSimulationMode is set to the value REVERSE_SIMULATION. This signifies to the low-level routines that the adjoint forms of the appropriate communication operation should be performed.

9. **MAX_NO_THREADS** The variable MAX_NO_THREADS is used to indicate the maximum number of OS threads that a code will use. This value defaults to thirty-two and is set in the file EEPARAMS.h. For single

threaded execution it can be reduced to one if required. The value is largely private to the WRAPPER and application code will not normally reference the value, except in the following scenario.

For certain physical parametrization schemes it is necessary to have a substantial number of work arrays. Where these arrays are allocated in heap storage (for example COMMON blocks) multi-threaded execution will require multiple instances of the COMMON block data. This can be achieved using a Fortran 90 module construct. However, if this mechanism is unavailable then the work arrays can be extended with dimensions using the tile dimensioning scheme of nSx and nSy (as described in Section 6.3.1). However, if the configuration being specified involves many more tiles than OS threads then it can save memory resources to reduce the variable MAX_NO_THREADS to be equal to the actual number of threads that will be used and to declare the physical parameterization work arrays with a single MAX_NO_THREADS extra dimension. An example of this is given in the verification experiment verification/aim.5l_cs. Here the default setting of MAX_NO_THREADS is altered to

```
INTEGER MAX_NO_THREADS
PARAMETER ( MAX_NO_THREADS =     6 )
```

and several work arrays for storing intermediate calculations are created with declarations of the form.

```
common /FORCIN/ sst1(ngp,MAX_NO_THREADS)
```

This declaration scheme is not used widely, because most global data is used for permanent, not temporary, storage of state information. In the case of permanent state information this approach cannot be used because there has to be enough storage allocated for all tiles. However, the technique can sometimes be a useful scheme for reducing memory requirements in complex physical parameterizations.

6.3.3.1 Specializing the Communication Code

The isolation of performance critical communication primitives and the subdivision of the simulation domain into tiles is a powerful tool. Here we show how it can be used to improve application performance and how it can be used to adapt to new gridding approaches.

6.3.3.2 JAM example

On some platforms a big performance boost can be obtained by binding the communication routines _EXCH and _GSUM to specialized native libraries (for example, the shmem library on CRAY T3E systems). The LETS_MAKE_JAM CPP flag is used as an illustration of a specialized communication configuration that substitutes for standard, portable forms of _EXCH and _GSUM. It affects three source files eeboot.F, CPP_EEMACROS.h and cg2d.F. When the flag is defined is has the following effects.

- An extra phase is included at boot time to initialize the custom communications library (see ini_jam.F).
- The _GSUM and _EXCH macro definitions are replaced with calls to custom routines (see gsum_jam.F and exch_jam.F)
- a highly specialized form of the exchange operator (optimized for overlap regions of width one) is substituted into the elliptic solver routine cg2d.F.

Developing specialized code for other libraries follows a similar pattern.

6.3.3.3 Cube sphere communication

Actual _EXCH routine code is generated automatically from a series of template files, for example exch2_rx1_cube.template. This is done to allow a large number of variations of the exchange process to be maintained. One set of variations supports the cube sphere grid. Support for a cube sphere grid in MITgcm is based on

having each face of the cube as a separate tile or tiles. The exchange routines are then able to absorb much of the detailed rotation and reorientation required when moving around the cube grid. The set of _EXCH routines that contain the word cube in their name perform these transformations. They are invoked when the run-time logical parameter useCubedSphereExchange is set .TRUE.. To facilitate the transformations on a staggered C-grid, exchange operations are defined separately for both vector and scalar quantities and for grid-centered and for grid-face and grid-corner quantities. Three sets of exchange routines are defined. Routines with names of the form exch2_rx are used to exchange cell centered scalar quantities. Routines with names of the form exch2_uv_rx are used to exchange vector quantities located at the C-grid velocity points. The vector quantities exchanged by the exch_uv_rx routines can either be signed (for example velocity components) or un-signed (for example grid-cell separations). Routines with names of the form exch_z_rx are used to exchange quantities at the C-grid vorticity point locations.

6.4 MITgcm execution under WRAPPER

Fitting together the WRAPPER elements, package elements and MITgcm core equation elements of the source code produces the calling sequence shown below.

6.4.1 Annotated call tree for MITgcm and WRAPPER

WRAPPER layer.

```
MAIN
|
|--EEBOOT               :: WRAPPER initialization
|  |
|  |-- EEBOOT_MINMAL    :: Minimal startup. Just enough to
|  |                       allow basic I/O.
|  |-- EEINTRO_MSG      :: Write startup greeting.
|  |
|  |-- EESET_PARMS      :: Set WRAPPER parameters
|  |
|  |-- EEWRITE_EEENV    :: Print WRAPPER parameter settings
|  |
|  |-- INI_PROCS        :: Associate processes with grid regions.
|  |
|  |-- INI_THREADING_ENVIRONMENT   :: Associate threads with grid regions.
|       |
|       |--INI_COMMUNICATION_PATTERNS :: Initialize between tile
|                                      :: communication data structures
|
|
|--CHECK_THREADS    :: Validate multiple thread start up.
|
|--THE_MODEL_MAIN   :: Numerical code top-level driver routine
```

Core equations plus packages.

```
C Invocation from WRAPPER level...
C
C  |
C  |-THE_MODEL_MAIN :: Primary driver for the MITgcm algorithm
C    |              :: Called from WRAPPER level numerical
C    |              :: code invocation routine. On entry
C    |              :: to THE_MODEL_MAIN separate thread and
C    |              :: separate processes will have been established.
```

(continues on next page)

(continued from previous page)

```
C     |              :: Each thread and process will have a unique ID
C     |              :: but as yet it will not be associated with a
C     |              :: specific region in decomposed discrete space.
C     |
C     |-INITIALISE_FIXED :: Set fixed model arrays such as topography,
C     | |                :: grid, solver matrices etc..
C     | |
C     | |-INI_PARMS :: Routine to set kernel model parameters.
C     | |           :: Kernel parameters are read from file "data"
C     | |           :: in directory in which code executes.
C     | |
C     | |-PACKAGES_BOOT       :: Start up the optional package environment.
C     | |                     :: Runtime selection of active packages.
C     | |-CPL_IMPORT_CPLPARMS :: Import coupling parameters from/to
C     | |                     :: the coupler layer
C     | |-PACKAGES_READPARMS  :: Read each package input parameter file
C     | | |- ${PKG}_READPARMS
C     | |
C     | |-SET_PARMS :: Finalise model parameter setting (if fct of pkg usage)
C     | |
C     | |-INI_MODEL_IO   :: Initialise Input/Output setting
C     | |  |-MNC_INIT    :: Initialise MITgcm NetCDF interface (MNC)(see pkg/mnc)
C     | |  |-MNC_CW_INIT :: Initialise MNC grid and variable types  (see pkg/mnc)
C     | |  |-MON_INIT    :: Initialises monitor package ( see pkg/monitor )
C     | |
C     | |-INI_GRID       :: Control grid array (vert. and horiz.) initialisation.
C     | | |              :: Grid arrays are held and described in GRID.h.
C     | | |-LOAD_GRID_SPACING    :: Load grid spacing (vector) from files
C     | | |-INI_VERTICAL_GRID    :: Set up vertical grid and coordinate
C     | | |-INI_CARTESIAN_GRID   :: Cartesian horiz. grid initialisation
C     | | |                      :: (calculate grid from kernel parameters).
C     | | |-INI_SPHERICAL_POLAR_GRID :: Spherical polar horiz. grid setting
C     | | |                          :: (calculate grid from kernel parameters).
C     | | |-INI_CURVILINEAR_GRID :: General orthogonal, structured horiz. grid
C     | | |                      :: initialisation; input from raw grid files
C     | | |                      :: (LONC.bin, LATC.bin, DXF.bin, ... ) or per
C     | | |                      :: face file: horizGridFile(.faceXXX.bin)
C     | | |-INI_CYLINDER_GRID    :: Cylindrical horiz. grid setting
C     | |
C     | |-LOAD_REF_FILES   :: Read-in reference vertical profiles (T,S,Rho)
C     | |-INI_EOS          :: Initialise Equation Of State (EOS) coefficients
C     | |-SET_REF_STATE    :: Set reference pressure/geopotential, reference
C     | |                  :: stratification (for implicit IGW), vertical
C     | |                  :: velocity scaling factor and anelastic ref. density
C     | |-SET_GRID_FACTORS :: Set grid factors (fct of k) for deep-atmosphere
C     | |
C     | |-INI_DEPTHS    :: Read (from "bathyFile") or set bathymetry/orography.
C     | |-INI_MASKS_ETC :: Derive horizontal and vertical cell fractions and
C     | |               :: land masking for solid-fluid boundaries.
C     | |
C     | |-PACKAGES_INIT_FIXED  :: do all packages fixed-initialisation setting
C     | | |- ${PKG}_INIT_FIXED
C     | |
C     | |-INI_GLOBAL_DOMAIN :: Initialise domain related (global) quantities.
C     | |-INI_LINEAR_PHISURF :: Set ref. surface Bo_surf
C     | |
C     | |-INI_CORI          :: Set coriolis term. zero, f-plane, beta-plane,
```

(continues on next page)

(continued from previous page)

```
C     | |                   :: sphere options are coded.
C     | |-INI_CG2D          :: 2D conjugate grad solver initialisation.
C     | |-INI_CG3D          :: 3D conjugate grad solver initialisation.
C     | |
C     | |-CONFIG_SUMMARY    :: Provide synopsis of kernel setup. Includes
C     | |                   :: annotated table of kernel parameter settings.
C     | |
C     | |-PACKAGES_CHECK    :: call each package configuration checking S/R
C     | | |- ${PKG}_CHECK
C     | |
C     | |-CONFIG_CHECK      :: Check config and parameter consistency.
C     | |
C     | |-WRITE_GRID        :: write grid fields to output files
C     | |-CPL_EXCH_CONFIGS  :: exchange config with coupler-interface
C     |
C     |-CTRL_UNPACK         :: Control vector support package. see pkg/ctrl
C     |-COST_DEPENDENT_INIT :: ( see pkg/cost )
C     |
C     |-ADTHE_MAIN_LOOP :: Derivative evaluating form of main time stepping loop
C     !                 :: Automatically generated by TAMC/TAF.
C     |
C     |-THE_MAIN_LOOP   :: Main timestepping loop routine.
C     | |
C     | |-INITIALISE_VARIA :: Set the initial conditions for time evolving fields
C     | | |
C  #ifdef ALLOW_AUTODIFF
C     | | |-INI_DEPTHS         \
C     | | |-CTRL_DEPTH_INI      \
C     | | |-UPDATE_MASKS_ETC     } ALLOW_DEPTH_CONTROL case
C     | | |-UPDATE_CG2D         /
C  #endif
C     | | |-INI_NLFS_VARS :: Initialise all Non-Lin Free-Surf arrays (SURFACE.h)
C     | | |-INI_DYNVARS   :: Initialise to zero all DYNVARS.h arrays
C     | | |-INI_NH_VARS   :: Initialise to zero all NH_VARS.h arrays
C     | | |-INI_FFIELDS   :: Initialise forcing fields in FFIELDS.h to zero
C     | | |
C     | | |-INI_FIELDS    :: Control initialising model fields to non-zero
C     | | | |-INI_VEL     :: Initialize 3D flow field.
C     | | | |-INI_THETA   :: Set model initial temperature field.
C     | | | |-INI_SALT    :: Set model initial salinity field.
C     | | | |-INI_PSURF   :: Set model initial free-surface height/pressure.
C     | | | |-READ_PICKUP :: Read in main model pickup files to restart a run.
C     | | |
C     | | |-INI_MIXING   :: Initialise diapycnal diffusivity.
C     | | |
C     | | |-TAUEDDY_INIT_VARIA :: Initialise eddy (bolus) streamfunction
C     | | |
C     | | |-INI_FORCING  :: Set model initial forcing fields, either
C     | | |   |          :: set in-line or from file as shown here:
C     | | |   |-READ_FLD_XY_RS(zonalWindFile)
C     | | |   |-READ_FLD_XY_RS(meridWindFile)
C     | | |   |-READ_FLD_XY_RS(surfQnetFile)
C     | | |   |-READ_FLD_XY_RS(EmPmRfile)
C     | | |   |-READ_FLD_XY_RS(thetaClimFile)
C     | | |   |-READ_FLD_XY_RS(saltClimFile)
C     | | |   |-READ_FLD_XY_RS(surfQswFile)
C     | | |
```

(continues on next page)

(continued from previous page)

```
C     | | |-AUTODIFF_INIT_VARIA :: (see pkg/autodiff )
C     | | |
C     | | |-PACKAGES_INIT_VARIABLES :: Does initialisation of time evolving
C     | | | | ${PKG}_INIT_VARIA     :: package data.
C     | | |
C     | | |-COST_INIT_VARIA     :: ( see pkg/cost )
C     | | |-CONVECTIVE_ADJUSTMENT_INI :: Apply conv. adjustment to initial state
C     | | |
C     | | |-CALC_R_STAR    :: Calculate the new level thickness factor (r* coord)
C     | | |-UPDATE_R_STAR  :: Update the level thickness fraction (r* coord).
C     | | |-UPDATE_SIGMA   :: Update the level thickness fraction (sigma-coord).
C     | | |-CALC_SURF_DR   :: Calculate the new surface level thickness.
C     | | |-UPDATE_SURF_DR :: Update the surface-level thickness fraction.
C     | | |
C     | | |-UPDATE_CG2D    :: Update 2D conjugate grad. for Free-Surf.
C     | | |
C     | | |-INTEGR_CONTINUITY :: Integrate the continuity Equation
C     | | | |-INTEGRATE_FOR_W :: Integrate for vertical velocity
C     | | | |-OBCS_APPLY_W    :: Open boundary package (see pkg/obcs).
C     | | | |-DUMMY_FOR_ETAN  :: For printing adEtaN (see pkg/autodiff).
C     | | | |-UPDATE_ETAH     :: Update Surface height/pressure
C     | | |
C     | | |-CALC_R_STAR    :: Calculate the new level thickness factor (r* coord)
C     | | |-CALC_SURF_DR   :: Calculate the new surface level thickness.
C     | | |
C     | | |-STATE_SUMMARY    :: Summarise model prognostic variables.
C     | | |
C     | | |-MONITOR          :: Monitor state (see pkg/monitor)
C     | | |
C     | | |-DO_STATEVARS_TAVE :: Time averaging package ( see pkg/timeave ).
C     | | |  |-TIMEAVE_STATVARS :: Accumulate main model state variables
C     | | |  |-PTRACERS_TIMEAVE :: Accumulate passive tracers variables
C     | | |
C     | | |-DO_THE_MODEL_IO  :: Controlling routine for IO
C     | | | |-WRITE_STATE    ::  Write model state variables.
C     | | | |-TIMEAVE_STATV_WRITE :: Write Time averaged output (see pkg/timeave)
C     | | | |-FIZHI_WRITE_STATE :: Write Fizhi pkg output (see pkg/fizhi)
C     | | | |-AIM_WRITE_TAVE    :: Write AIM  pkg output (see pkg/aim_v23)
C     | | | |-LAND_OUTPUT       :: Write Land pkg output (see pkg/land)
C     | | | |-OBCS_OUTPUT       :: Write OBCS pkg output (see pkg/obcs)
C     | | | |-GMREDI_OUTPUT     :: Write GM-Redi pkg output (see pkg/gmredi)
C     | | | |-KPP_OUTPUT        :: Write KPP  pkg output (see pkg/kpp)
C     | | | |-PP81_OUTPUT       :: Write PP81 pkg output (see pkg/pp81)
C     | | | |-KL10_OUTPUT       :: Write KL10 pkg output (see pkg/kl10)
C     | | | |-MY82_OUTPUT       :: Write MY82 pkg output (see pkg/my82)
C     | | | |-OPPS_OUTPUT       :: Write OPPS pkg output (see pkg/opps)
C     | | | |-GGL90_OUTPUT      :: Write GGL90 pkg output (see pkg/ggl90)
C     | | | |-SBO_CALC          :: Compute SBO diagnostics (see pkg/sbo)
C     | | | |-SBO_OUTPUT        :: Write SBO  pkg output   (see pkg/sbo)
C     | | | |-SEAICE_OUTPUT     :: Write SeaIce pkg output (see pkg/seaice)
C     | | | |-SHELFICE_OUTPUT   :: Write ShelfIce pkg output (see pkg/shelfice)
C     | | | |-BULKF_OUTPUT      :: Write Bulk-Force output (see pkg/bulK_force)
C     | | | |-THSICE_OUTPUT     :: Write ThSIce pkg output (see pkg/thsice)
C     | | | |-PTRACERS_OUTPUT   :: Write pTracers pkg output (see pkg/ptracers)
C     | | | |-MATRIX_OUTPUT     :: Write Matrix pkg output (see pkg/matrix)
C     | | | |-GCHEM_OUTPUT      :: Write Geochemistry pkg output (see pkg/gchem)
C     | | | |-CPL_OUTPUT        :: Write Coupler-Interface output (see
```

(continues on next page)

(continued from previous page)

```
C     | | | |                   :: pkg/atm_compon_interf, pkg/ocn_compon_interf)
C     | | | |-LAYERS_CALC       :: Calculate layers diagnostics (see pkg/layers)
C     | | | |-LAYERS_OUTPUT     :: Write Layers pkg output (see pkg/layers)
C     | | | |-DIAGNOSTICS_WRITE :: Write pkg/diagnostics output
C     | | |
C====|>| ***************************
C====|>| BEGIN MAIN TIMESTEPPING LOOP
C====|>| ***************************
C     | |-COST_AVERAGESFIELDS :: time-averaged Cost function terms (see pkg/cost)
C     | |-PROFILES_INLOOP     :: ( see pkg/profiles )
C     | /
C     | |-MAIN_DO_LOOP    :: Open-AD case: Main timestepping loop routine
C     | \                    otherwise: just call FORWARD_STEP
C     | |
C/\   | |-FORWARD_STEP        :: Step forward a time-step ( AT LAST !!! )
C/\   | | |
C/\   | | |-AUTODIFF_INADMODE_UNSET :: Set/reset some adjoint flags
C/\   | | |-RESET_NLFS_VARS   :: Reset some Non-Lin Free-Surf vars (Adjoint)
C/\   | | |-UPDATE_R_STAR     :: Reset r-star factor variables     (Adjoint)
C/\   | | |-UPDATE_SURF_DR    :: Reset NLFS surface thickness vars (Adjoint)
C/\   | | |
C/\   | | |-PTRACERS_SWITCH_ONOFF    :: Set/reset pTracers time-stepping switch
C/\   | | |-DIAGNOSTICS_SWITCH_ONOFF :: Activate/de-activate diagnostics
C/\   | | |-DO_STATEVARS_DIAGS ( 0 ) :: fill-up state variable diagnostics
C/\   | | |
C/\   | | |-NEST_CHILD_SETMEMO :: Nesting interface
C/\   | | |-NEST_PARENT_IO_1   :: Nesting interface
C/\   | | |
C/\   | | |-LOAD_FIELDS_DRIVER :: Control loading of input fields from files
C/\   | | |
C/\   | | |-BULKF_FORCING      :: Calculate surface forcing (see pkg/bulk_force)
C/\   | | |-CHEAPAML           :: Cheap AML driver ( see pkg/cheapaml )
C/\   | | |-CTRL_MAP_FORCING   :: Control vector support package. (see pkg/ctrl)
C/\   | | |-DUMMY_IN_STEPPING  :: Autodiff package ( pkg/autodiff ).
C/\   | | |
C/\   | | |-CPL_EXPORT_MY_DATA :: Send coupling fields to coupler
C/\   | | |-CPL_IMPORT_EXTERNAL_DATA :: Receive coupling fields from coupler
C/\   | | |
C/\   | | |-OASIS_PUT     :: Oasis coupler interface
C/\   | | |-OASIS_GET     :: Oasis coupler interface
C/\   | | |
C/\   | | |-EBM_DRIVER    :: Calculate EBM type atmospheric forcing (see pkg/ebm)
C/\   | | |
C/\   | | |-DO_ATMOSPHERIC_PHYS :: Atmospheric physics computation
C/\   | | | |
C/\   | | | |-UPDATE_OCEAN_EXPORTS     :: ( see pkg/fizhi )
C/\   | | | |-UPDATE_EARTH_EXPORTS     :: ( see pkg/fizhi )
C/\   | | | |-UPDATE_CHEMISTRY_EXPORTS :: ( see pkg/fizhi )
C/\   | | | |-FIZHI_WRAPPER            :: ( see pkg/fizhi )
C/\   | | | |-STEP_FIZHI_FG            :: ( see pkg/fizhi )
C/\   | | | |-FIZHI_UPDATE_TIME        :: ( see pkg/fizhi )
C/\   | | | |
C/\   | | | |-ATM_PHYS_DRIVER          :: ( see pkg/atm_phys )
C/\   | | | |
C/\   | | | |-AIM_DO_PHYSICS           :: ( see pkg/aim_v23 )
C/\   | | | |
C/\   | | |-DO_OCEANIC_PHYS     :: Oceanic (& seaice) physics computation
```

(continues on next page)

(continued from previous page)

```
C/\  | | | |
C/\  | | | |-OBCS_CALC         :: Open boundary. package (see pkg/obcs).
C/\  | | | |
C/\  | | | |-FRAZIL_CALC_RHS   :: Compute FRAZIL tendencies ( see pkg/frazil )
C/\  | | | |-THSICE_MAIN       :: Thermodynamic sea-ice driver (see pkg/thsice)
C/\  | | | |-SEAICE_MODEL      :: Sea-ice model driver (see pkg/seaice )
C/\  | | | |-SEAICE_COST_SENSI   :: Sea-ice cost-function (see pkg/seaice )
C/\  | | | |-SHELFICE_THERMODYNAMICS :: Compute ShelfIce thermo (pkg/shelfice)
C/\  | | | |-ICEFRONT_THERMODYNAMICS :: Compute IceFront thermo (pkg/icefront)
C/\  | | | |
C/\  | | | |-SALT_PLUME_DO_EXCH   :: (see pkg/salt_plume )
C/\  | | | |-FREEZE_SURFACE       :: Prevent SST to fall below TFreeze
C/\  | | | |-OCN_APPLY_IMPORT     :: Apply imported fields from coupler
C/\  | | | |-EXTERNAL_FORCING_SURF:: Compute appropriately dimensioned
C/\  | | | |                      :: surface forcing terms.
C/\  | | | |-FIND_RHO_2D @ p(k)   :: Calculate [rho(T,S,p)-Rho_0] of a slice
C/\  | | | |-FIND_RHO_2D @ p(k-1) :: Calculate [rho(T,S,p)-Rho_0] of a slice
C/\  | | | |-GRAD_SIGMA           :: Calculate isoneutral gradients
C/\  | | | |-CALC_IVDC       :: Set Implicit Vertical Diffusivity for Convection
C/\  | | | |-CALC_OCE_MXLAYER        :: Diagnose Oceanic Mixed Layer depth
C/\  | | | |
C/\  | | | |-SALT_PLUME_CALC_DEPTH   :: (see pkg/salt_plume )
C/\  | | | |-SALT_PLUME_VOLFRAC      :: (see pkg/salt_plume )
C/\  | | | |-SALT_PLUME_APPLY (Temp) :: (see pkg/salt_plume )
C/\  | | | |-SALT_PLUME_APPLY (Salt) :: (see pkg/salt_plume )
C/\  | | | |-SALT_PLUME_FORCING_SURF :: (see pkg/salt_plume )
C/\  | | | |-KPP_CALC           :: Compute KPP  vertical mixing ( see pkg/kpp )
C/\  | | | |-PP81_CALC          :: Compute PP81 vertical mixing ( see pkg/pp81 )
C/\  | | | |-KL10_CALC          :: Compute KL10 vertical mixing ( see pkg/kl10 )
C/\  | | | |-MY82_CALC          :: Compute MY82 vertical mixing ( see pkg/kl10 )
C/\  | | | |-GGL90_CALC         :: Compute GGL90 vertical mixing (see pkg/ggl10)
C/\  | | | |-GMREDI_CALC_TENSOR :: Compute GM-Redi tensor ( see pkg/gmredi )
C/\  | | | |-DWNSLP_CALC_FLOW   :: Compute Down-Slope flow  (see pkg/down_slope)
C/\  | | | |-BBL_CALC_RHS       :: Compute BBL tendencies ( see pkg/bbl )
C/\  | | | |-MYPACKAGE_CALC_RHS :: Compute mypackage tendencies (pkg/mypackage)
C/\  | | | |
C/\  | | | |-GMREDI_DO_EXCH     :: ( see pkg/gmredi )
C/\  | | | |-KPP_DO_EXCH        :: ( see pkg/kpp )
C/\  | | | |-DIAGS_RHO_G        :: Compute some density related diagnostics
C/\  | | | |-DIAGS_OCEANIC_SURF_FLUX :: Diagnose oceanic surface fluxes
C/\  | | | |-SALT_PLUME_DIAGNOSTICS_FILL :: (see pkg/salt_plume )
C/\  | | | |-ECCO_PHYS          :: ( see pkg/ecco )
C/\  | | | |
C/\  | | |-STREAMICE_TIMESTEP   :: ( see pkg/streamice )
C/\  | | | |
C/\  | | |-GCHEM_CALC_TENDENCY  :: geochemistry driver routine (see pkg/gchem)
C/\  | | | |
C/\  | | |-LONGSTEP_AVERAGE        :: Averaging state vars ( see pkg/longstep )
C/\  | | |-LONGSTEP_THERMODYNAMICS :: Step forward tracers ( see pkg/longstep )
C/\  | | | |
C/\  | | |-THERMODYNAMICS       :: theta, salt + tracer equations driver.
C/\  | | | |                        (synchronous time-stepping case)
C/\  | | | |-CALC_WSURF_TR          :: Compute T & S Linear-Free-Surf correction
C/\  | | | |-PTRACERS_CALC_WSURF_TR :: Compute Tracers Linear-Free-Surf correct.
C/\  | | | | |
C/\  | | | |-GMREDI_RESIDUAL_FLOW :: Get the flow field used to advect tracers
C/\  | | | | |
```

(continues on next page)

(continued from previous page)

```
C/\  | | | |-TEMP_INTEGRATE       :: Step forward Prognostic Eq for Temperature.
C/\  | | | | |
C/\  | | | | |-ADAMS_BASHFORTH3   :: Extrapolate tracer forward in time (AB-3)
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tracer forward in time (AB-2)
C/\  | | | | |-CALC_3D_DIFFUSIVITY :: set vertical diffusivity
C/\  | | | | |
C/\  | | | | |-GAD_SOM_ADVECT     :: Second Order Moment (SOM) advection
C/\  | | | | |-GAD_ADVECTION      :: Generalised advection driver (multi-dim
C/\  | | | | |                         advection case) (see pkg/gad).
C/\  | | | | |-CALC_ADV_FLOW      :: set 3-D flow field to advect tracer
C/\  | | | | |-APPLY_FORCING_T    :: Problem specific forcing for temperature.
C/\  | | | | |-GAD_CALC_RHS       :: Calculate Advection-Diffusion tendency terms
C/\  | | | | |
C/\  | | | | |-ADAMS_BASHFORTH3   :: Extrapolate tendency forward in time (AB-3)
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tendency forward in time (AB-2)
C/\  | | | | |-FREESURF_RESCALE_G :: Re-scale Gt for free-surface height.
C/\  | | | | |-DWNSLP_APPLY       :: Add pkg/down_slope tendency
C/\  | | | | |
C/\  | | | | |-TIMESTEP_TRACER    :: Step tracer field forward in time
C/\  | | | | |
C/\  | | | | |-GAD_IMPLICIT_R     :: Solve vertical implicit Advect-Diffus. eqn.
C/\  | | | | |-IMPLDIFF           :: Solve vertical implicit diffusion equation.
C/\  | | | | |-CYCLE_AB_TRACER    :: Cycle time-stepping arrays for tracer field
C/\  | | | | |-CYCLE_TRACER       :: Cycle time-stepping arrays for tracer field
C/\  | | | | |
C/\  | | | |-SALT_INTEGRATE       :: Step forward Prognostic Eq for Salinity.
C/\  | | | | | |                    same sequence of calls as in TEMP_INTEGRATE
C/\  | | | | |
C/\  | | | |-PTRACERS_INTEGRATE   :: Integrate other tracer(s) (see pkg/ptracers).
C/\  | | | | | |                    same sequence of calls as in TEMP_INTEGRATE
C/\  | | | | |-OBCS_APPLY_PTRACER :: Open boundary package for pTracers
C/\  | | | | |
C/\  | | | |-OBCS_APPLY_TS        :: Open boundary package (see pkg/obcs ).
C/\  | | | |
C/\  | | |-LONGSTEP_AVERAGE        :: Averaging state vars ( see pkg/longstep )
C/\  | | |-LONGSTEP_THERMODYNAMICS :: Step forward tracers ( see pkg/longstep )
C/\  | | |
C/\  | | |-DO_STAGGER_FIELDS_EXCHANGES :: Update overlap regions of arrays
C/\  | | |                                Theta & Salt (implicit IGW case)
C/\  | | |
C/\  | | |-DYNAMICS       :: Momentum equations driver.
C/\  | | | |
C/\  | | | |-CALC_GRAD_PHI_SURF :: Calculate the gradient of the surface
C/\  | | | |                       Potential anomaly.
C/\  | | | |-CALC_VISCOSITY   :: Calculate net vertical viscosity
C/\  | | | |-MOM_CALC_3D_STRAIN :: Calculates the strain tensor of 3D flow field
C/\  | | | |-OBCS_COPY_UV_N   :: for Stevens bndary Conditions (see pkg/obcs)
C/\  | | | |
C/\  | | | |-CALC_PHI_HYD     :: Integrate the hydrostatic relation.
C/\  | | | |-MOM_FLUXFORM     :: Flux Form momentum eqn. (pkg/mom_fluxform)
C/\  | | | |-MOM_VECINV       :: Vector Invariant momentum eqn (pkg/mom_vecinv)
C/\  | | | |-MOM_CALC_SMAG_3D :: Calculate Smagorinsky 3D (harmonic) viscosities
C/\  | | | |-MOM_UV_SMAG_3D   :: Calculate U,V mom. tendency due to Smag 3D Visc
C/\  | | | |-TIMESTEP         :: Step horizontal momentum fields forward in time
C/\  | | | |
C/\  | | | |-MOM_U_IMPLICIT_R :: Solve implicitly vertical Adv-Diffus equation.
C/\  | | | |-IMPLDIFF         :: Solve vertical implicit diffusion equation.
```

(continues on next page)

(continued from previous page)

```
C/\  | | | |-OBCS_SAVE_UV_N   :: for Stevens bndary Conditions (see pkg/obcs)
C/\  | | | |-OBCS_APPLY_UV    :: Apply Open bndary Conditions to provisional U,V
C/\  | | | |-IMPLDIFF         :: (CD-Scheme) Solve vertical impl. diffus. eqn
C/\  | | | |
C/\  | | | |-CALC_GW          :: Vert. momentum tendency terms (Non-Hydrostatic)
C/\  | | | | |-MOM_W_SMAG_3D  :: Calculate W mom. tendency due to Smag 3D Visc
C/\  | | | |-TIMESTEP_WVEL    :: Step vert mom forward in time (Non-Hydrostatic)
C/\  | | |
C/\  | | |-MNC_UPDATE_TIME    :: Update MNC time record (see pkg/mnc)
C/\  | | |
C/\  | | |-UPDATE_R_STAR  :: Update the level thickness fraction (r* coord).
C/\  | | |-UPDATE_SIGMA   :: Update the level thickness fraction (sigma-coord).
C/\  | | |-UPDATE_R_STAR  :: Update the level thickness fraction.
C/\  | | |-UPDATE_SURF_DR :: Update the surface-level thickness fraction.
C/\  | | |-UPDATE_CG2D    :: Update 2D conjugate grad. for Free-Surf.
C/\  | | |
C/\  | | |-SHAP_FILT_APPLY_UV  :: Apply Shapiro Filter to provisional velocity
C/\  | | |-ZONAL_FILT_APPLY_UV :: Apply  Zonal  Filter to provisional velocity
C/\  | | |
C/\  | | |-SOLVE_FOR_PRESSURE  :: Find surface pressure.
C/\  | | | |-CALC_DIV_GHAT     :: Form the RHS of the surface pressure eqn.
C/\  | | | |-CG2D              :: Two-dim pre-con. conjugate-gradient.
C/\  | | | |-PRE_CG3D          :: Finish to set the RHS of the 3-D pressure eqn.
C/\  | | | |-CG3D              :: Three-dim pre-con. conjugate-gradient solver.
C/\  | | | |-POST_CG3D         :: finalise solution of NH and Free-Surf pressure
C/\  | | |
C/\  | | |-MOMENTUM_CORRECTION_STEP :: Finalise momentum stepping
C/\  | | | |-CALC_GRAD_PHI_SURF  :: Return DDx and DDy of surface pressure
C/\  | | | |-CORRECTION_STEP     :: Pressure correction to momentum
C/\  | | | |-OBCS_APPLY_UV       :: Open boundary package (see pkg/obcs).
C/\  | | | |-SHAP_FILT_APPLY_UV  :: Apply Shapiro Filter to latest velocity
C/\  | | | |-ZONAL_FILT_APPLY_UV :: Apply  Zonal  Filter to latest velocity
C/\  | | |
C/\  | | |-INTEGR_CONTINUITY   :: Integrate continuity equation (see above)
C/\  | | |
C/\  | | |-CALC_R_STAR    :: Calculate the new level thickness factor (r* coord)
C/\  | | |-CALC_SURF_DR   :: Calculate the new surface level thickness.
C/\  | | |
C/\  | | |-DO_STAGGER_FIELDS_EXCHANGES :: Update overlap regions of arrays
C/\  | | |                             uVel,vVel & wVel (stagger-time-step case)
C/\  | | |
C/\  | | |-DO_STATEVARS_DIAGS ( 1 ) :: fill-up state variable diagnostics
C/\  | | |
C/\  | | |-THERMODYNAMICS       :: theta, salt + tracer Eq. driver (see above).
C/\  | | |                         (staggered time-stepping case)
C/\  | | |
C/\  | | |-TRACERS_CORRECTION_STEP :: Finalise tracer stepping:
C/\  | | | |                       ::  apply filter, conv.adjustment
C/\  | | | |-TRACERS_IIGW_CORRECTION :: apply Implicit IGW adjustment to T & S
C/\  | | | |-SHAP_FILT_APPLY_TS      :: Apply Shapiro Filter to latest T & S
C/\  | | | |-ZONAL_FILT_APPLY_TS     :: Apply  Zonal  Filter to latest T & S
C/\  | | | |-PTRACERS_ZONAL_FILT_APPLY :: Apply  Zonal Filter to pTracers
C/\  | | | |-SALT_FILL               :: Fill up negative Salt
C/\  | | | |-OPPS_INTERFACE          :: ( see pkg/opps )
C/\  | | | |-CONVECTIVE_ADJUSTMENT   :: Apply convective adjustment
C/\  | | | |-MATRIX_STORE_TENDENCY_IMP :: ( see pkg/matrix )
C/\  | | |
```

(continues on next page)

(continued from previous page)

```
C/\  | | |-LONGSTEP_AVERAGE        :: Averaging state vars ( see pkg/longstep )
C/\  | | |-LONGSTEP_THERMODYNAMICS :: Step forward tracers ( see pkg/longstep )
C/\  | | |
C/\  | | |-GCHEM_FORCING_SEP :: Tracer forcing for gchem pkg (if tracer
C/\  | | |                  :: dependent tendencies calculated separately)
C/\  | | |
C/\  | | |-DO_FIELDS_BLOCKING_EXCHANGES :: Sync up overlap regions.
C/\  | | |
C/\  | | |-DO_STATEVARS_DIAGS ( 2 ) :: fill-up state variable diagnostics
C/\  | | |
C/\  | | |-GRIDALT_UPDATE    :: ( see pkg/gridalt )
C/\  | | |-STEP_FIZHI_CORR   :: ( see pkg/fizhi )
C/\  | | |
C/\  | | |-FLT_MAIN          :: Step forward Floats (see pkg/flt)
C/\  | | |
C/\  | | |-DO_STATEVARS_TAVE :: Time averaging package (see above)
C/\  | | |
C/\  | | |-NEST_PARENT_IO_2  :: Nesting interface
C/\  | | |-NEST_CHILD_TRANSP :: Nesting interface
C/\  | | |
C/\  | | |-MONITOR          :: Monitor package (pkg/monitor).
C/\  | | |
C/\  | | |-COST_TILE        :: ( see pkg/cost )
C/\  | | |
C/\  | | |-DO_THE_MODEL_IO  :: Controlling routine for IO (see above)
C/\  | | |
C/\  | | |-PTRACERS_RESET   :: Re-initialize PTRACERS ( see pkg/ptracers )
C/\  | | |
C/\  | | |-DO_WRITE_PICKUP  :: Controlling routine for writing files to restart
C/\  | | | |-PACKAGES_WRITE_PICKUP :: Write pickup files for each package
C/\  | | | | |                     ::  which needs it to restart
C/\  | | | | |-GAD_WRITE_PICKUP       :: Write Generic AdvDiff pickups for SOM
C/\  | | | | |                        :: advection scheme (pkg/generic_advdiff)
C/\  | | | | |-CD_CODE_WRITE_PICKUP   :: Write CD-code pickups (see pkg/cd_code)
C/\  | | | | |-OBCS_WRITE_PICKUP      :: Write OBCS pickups    (see pkg/obcs)
C/\  | | | | |-GGL90_WRITE_PICKUP     :: Write GGL90 pickups   (see pkg/ggl90)
C/\  | | | | |-BBL_WRITE_PICKUP       :: Write BBL pickups     (see pkg/bbl)
C/\  | | | | |-CHEAPAML_WRITE_PICKUP  :: Write CheapAML pickups (pkg/cheapaml)
C/\  | | | | |-FLT_WRITE_PICKUP       :: Write Floats pickups  (see pkg/flt)
C/\  | | | | |-PTRACERS_WRITE_PICKUP  :: Write pTracers pickups (pkg/ptracers)
C/\  | | | | |-GCHEM_WRITE_PICKUP     :: Write Geo-Chem pickups (see pkg/gchem)
C/\  | | | | |-SEAICE_WRITE_PICKUP    :: Write SeaIce pickups  (see pkg/seaice)
C/\  | | | | |-STREAMICE_WRITE_PICKUP :: Write StreamIce pickups (pkg/streamice)
C/\  | | | | |-SHELFICE_WRITE_PICKUP :: Write ShelfIce pickups (pkg/shelfice)
C/\  | | | | |-THSICE_WRITE_PICKUP    :: Write ThSIce pickups  (see pkg/thsice)
C/\  | | | | |-LAND_WRITE_PICKUP      :: Write Land   pickups  (see pkg/land)
C/\  | | | | |-ATM_PHYS_WRITE_PICKUP  :: Write Atm-Phys pickups (pkg/atm_phys)
C/\  | | | | |-FIZHI_WRITE_PICKUP     :: Write Fizhi pickups   (see pkg/fizhi)
C/\  | | | | |-FIZHI_WRITE_VEGTILES   :: Write Fizhi VegTiles  (see pkg/fizhi)
C/\  | | | | |-FIZHI_WRITE_DATETIME   :: Write Fizhi DateTime  (see pkg/fizhi)
C/\  | | | | |-CPL_WRITE_PICKUP       :: Write Coupling-Interface pickups
C/\  | | | | |-MYPACKAGE_WRITE_PICKUP :: Write pkg/mypackage pickups
C/\  | | | | |
C/\  | | | |-WRITE_PICKUP          :: Write main model pickup files.
C/\  | | | |
C/\  | | |-AUTODIFF_INADMODE_SET   :: Set/reset some adjoint flags
C    | |
```

(continues on next page)

(continued from previous page)

```
C<===|=| *************************
C<===|=| END MAIN TIMESTEPPING LOOP
C<===|=| *************************
C    | |
C    | |-COST_AVERAGESFIELDS :: Time-averaged Cost function terms (see pkg/cost)
C    | |-PROFILES_INLOOP     :: ( see pkg/profiles )
C    | |-COST_FINAL          :: Cost function package (see pkg/cost)
C    |
C    |-CTRL_PACK       :: Control vector support package (see pkg/ctrl)
C    |
C    |-GRDCHK_MAIN     :: Gradient check package (see pkg/grdchk)
C    |
C    |-TIMER_PRINTALL  :: Computational timing summary
C    |
C    |-COMM_STATS      :: Summarise inter-proc and inter-thread communication
C    |                 :: events.
```

6.4.2 Measuring and Characterizing Performance

TO BE DONE (CNH)

6.4.3 Estimating Resource Requirements

TO BE DONE (CNH)

6.4.3.1 Atlantic 1/6 degree example

6.4.3.2 Dry Run testing

6.4.3.3 Adjoint Resource Requirements

6.4.3.4 State Estimation Environment Resources

CHAPTER 7

Automatic Differentiation

Author: Patrick Heimbach

Automatic differentiation (AD), also referred to as algorithmic (or, more loosely, computational) differentiation, involves automatically deriving code to calculate partial derivatives from an existing fully non-linear prognostic code (see Griewank and Walther, 2008 [GW08]). A software tool is used that parses and transforms source files according to a set of linguistic and mathematical rules. AD tools are like source-to-source translators in that they parse a program code as input and produce a new program code as output (we restrict our discussion to source-to-source tools, ignoring operator-overloading tools). However, unlike a pure source-to-source translation, the output program represents a new algorithm, such as the evaluation of the Jacobian, the Hessian, or higher derivative operators. In principle, a variety of derived algorithms can be generated automatically in this way.

MITgcm has been adapted for use with the Tangent linear and Adjoint Model Compiler (TAMC) and its successor TAF (Transformation of Algorithms in Fortran), developed by Ralf Giering (Giering and Kaminski, 1998 [GK98], Giering, 2000 [Gie00]). The first application of the adjoint of MITgcm for sensitivity studies was published by Marotzke et al. (1999) [MGZ+99]. Stammer et al. (1997, 2002) [SWG+97] [SWG+02] use MITgcm and its adjoint for ocean state estimation studies. In the following we shall refer to TAMC and TAF synonymously, except were explicitly stated otherwise.

As of mid-2007 we are also able to generate fairly efficient adjoint code of the MITgcm using a new, open-source AD tool, called OpenAD (see Naumann, 2006 [NUH+06] and Utke et al., 2008 [UNF+08]). This enables us for the first time to compare adjoint models generated from different AD tools, providing an additional accuracy check, complementary to finite-difference gradient checks. OpenAD and its application to MITgcm is described in detail in Section 7.5.

The AD tool exploits the chain rule for computing the first derivative of a function with respect to a set of input variables. Treating a given forward code as a composition of operations – each line representing a compositional element, the chain rule is rigorously applied to the code, line by line. The resulting tangent linear or adjoint code, then, may be thought of as the composition in forward or reverse order, respectively, of the Jacobian matrices of the forward code's compositional elements.

7.1 Some basic algebra

Let $\mathcal{M}$ be a general nonlinear, model, i.e., a mapping from the m-dimensional space $U \subset \mathbb{R}^m$ of input variables $\vec{u} = (u_1, \ldots, u_m)$ (model parameters, initial conditions, boundary conditions such as forcing functions) to the n-dimensional space $V \subset \mathbb{R}^n$ of model output variable $\vec{v} = (v_1, \ldots, v_n)$ (model state, model diagnostics, objective function, ...) under consideration:

$$\begin{aligned} \mathcal{M} : U &\longrightarrow V \\ \vec{u} &\longmapsto \vec{v} = \mathcal{M}(\vec{u}) \end{aligned} \tag{7.1}$$

The vectors $\vec{u} \in U$ and $\vec{v} \in V$ may be represented with respect to some given basis vectors $\mathrm{span}(U) = \{\vec{e}_i\}_{i=1,\ldots,m}$ and $\mathrm{span}(V) = \{\vec{f}_j\}_{j=1,\ldots,n}$ as

$$\vec{u} = \sum_{i=1}^{m} u_i \, \vec{e}_i, \qquad \vec{v} = \sum_{j=1}^{n} v_j \, \vec{f}_j$$

Two routes may be followed to determine the sensitivity of the output variable $\vec{v}$ to its input $\vec{u}$.

7.1.1 Forward or direct sensitivity

Consider a perturbation to the input variables $\delta\vec{u}$ (typically a single component $\delta\vec{u} = \delta u_i \, \vec{e}_i$). Their effect on the output may be obtained via the linear approximation of the model $\mathcal{M}$ in terms of its Jacobian matrix M, evaluated in the point $u^{(0)}$ according to

$$\delta\vec{v} = M|_{\vec{u}^{(0)}} \, \delta\vec{u} \tag{7.2}$$

with resulting output perturbation $\delta\vec{v}$. In components $M_{ji} = \partial\mathcal{M}_j/\partial u_i$, it reads

$$\delta v_j = \sum_i \left. \frac{\partial \mathcal{M}_j}{\partial u_i} \right|_{u^{(0)}} \delta u_i \tag{7.3}$$

(7.2) is the tangent linear model (TLM). In contrast to the full nonlinear model $\mathcal{M}$, the operator M is just a matrix which can readily be used to find the forward sensitivity of $\vec{v}$ to perturbations in u, but if there are very many input variables ($\gg O(10^6)$ for large-scale oceanographic application), it quickly becomes prohibitive to proceed directly as in (7.2), if the impact of each component $\mathbf{e_i}$ is to be assessed.

7.1.2 Reverse or adjoint sensitivity

Let us consider the special case of a scalar objective function $\mathcal{J}(\vec{v})$ of the model output (e.g., the total meridional heat transport, the total uptake of CO_2 in the Southern Ocean over a time interval, or a measure of some model-to-data misfit)

$$\begin{array}{rccccc} \mathcal{J} : & U & \longrightarrow & V & \longrightarrow & \mathbb{R} \\ & \vec{u} & \longmapsto & \vec{v} = \mathcal{M}(\vec{u}) & \longmapsto & \mathcal{J}(\vec{u}) = \mathcal{J}(\mathcal{M}(\vec{u})) \end{array} \tag{7.4}$$

The perturbation of $\mathcal{J}$ around a fixed point $\mathcal{J}_0$,

$$\mathcal{J} = \mathcal{J}_0 + \delta\mathcal{J}$$

can be expressed in both bases of $\vec{u}$ and $\vec{v}$ with respect to their corresponding inner product $\langle \ , \ \rangle$

$$\begin{aligned} \mathcal{J} &= \mathcal{J}|_{\vec{u}^{(0)}} + \left\langle \nabla_u \mathcal{J}^T|_{\vec{u}^{(0)}} \, , \, \delta\vec{u} \right\rangle + O(\delta\vec{u}^2) \\ &= \mathcal{J}|_{\vec{v}^{(0)}} + \left\langle \nabla_v \mathcal{J}^T|_{\vec{v}^{(0)}} \, , \, \delta\vec{v} \right\rangle + O(\delta\vec{v}^2) \end{aligned} \tag{7.5}$$

(note, that the gradient ∇f is a co-vector, therefore its transpose is required in the above inner product). Then, using the representation of $\delta\mathcal{J} = \langle\, \nabla_v \mathcal{J}^T\,,\, \delta\vec{v}\,\rangle$, the definition of an adjoint operator A^* of a given operator A,

$$\langle\, A^*\vec{x}\,,\, \vec{y}\,\rangle = \langle\, \vec{x}\,,\, A\vec{y}\,\rangle$$

which for finite-dimensional vector spaces is just the transpose of A,

$$A^* \;=\; A^T$$

and from (7.2), (7.5), we note that (omitting $|$'s):

$$\delta\mathcal{J} \;=\; \langle\, \nabla_v \mathcal{J}^T\,,\, \delta\vec{v}\,\rangle \;=\; \langle\, \nabla_v \mathcal{J}^T\,,\, M\,\delta\vec{u}\,\rangle \;=\; \langle\, M^T\,\nabla_v \mathcal{J}^T\,,\, \delta\vec{u}\,\rangle \tag{7.6}$$

With the identity (7.5), we then find that the gradient $\nabla_u \mathcal{J}$ can be readily inferred by invoking the adjoint M^* of the tangent linear model M

$$\begin{aligned} \nabla_u \mathcal{J}^T|_{\vec{u}} &= M^T|_{\vec{u}} \cdot \nabla_v \mathcal{J}^T|_{\vec{v}} \\ &= M^T|_{\vec{u}} \cdot \delta\vec{v}^* \\ &= \delta\vec{u}^* \end{aligned} \tag{7.7}$$

(7.7) is the adjoint model (ADM), in which M^T is the adjoint (here, the transpose) of the tangent linear operator M, $\delta\vec{v}^*$ the adjoint variable of the model state $\vec{v}$, and $\delta\vec{u}^*$ the adjoint variable of the control variable $\vec{u}$.

The reverse nature of the adjoint calculation can be readily seen as follows. Consider a model integration which consists of Λ consecutive operations $\mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(.....(\mathcal{M}_\lambda(.....(\mathcal{M}_1(\mathcal{M}_0(\vec{u})))))$, where the $\mathcal{M}$'s could be the elementary steps, i.e., single lines in the code of the model, or successive time steps of the model integration, starting at step 0 and moving up to step Λ, with intermediate $\mathcal{M}_\lambda(\vec{u}) = \vec{v}^{(\lambda+1)}$ and final $\mathcal{M}_\Lambda(\vec{u}) = \vec{v}^{(\Lambda+1)} = \vec{v}$. Let $\mathcal{J}$ be a cost function which explicitly depends on the final state $\vec{v}$ only (this restriction is for clarity reasons only). $\mathcal{J}(u)$ may be decomposed according to:

$$\mathcal{J}(\mathcal{M}(\vec{u})) \;=\; \mathcal{J}(\mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(.....(\mathcal{M}_\lambda(.....(\mathcal{M}_1(\mathcal{M}_0(\vec{u}))))))) \tag{7.8}$$

Then, according to the chain rule, the forward calculation reads, in terms of the Jacobi matrices (we've omitted the $|$'s which, nevertheless are important to the aspect of *tangent* linearity; note also that by definition $\langle\, \nabla_v \mathcal{J}^T\,,\, \delta\vec{v}\,\rangle = \nabla_v \mathcal{J} \cdot \delta\vec{v}\,$)

$$\begin{aligned} \nabla_v \mathcal{J}(M(\delta\vec{u})) &= \nabla_v \mathcal{J} \cdot M_\Lambda \cdot ..... \cdot M_\lambda \cdot ..... \cdot M_1 \cdot M_0 \cdot \delta\vec{u} \\ &= \nabla_v \mathcal{J} \cdot \delta\vec{v} \end{aligned} \tag{7.9}$$

whereas in reverse mode we have

$$\boxed{\begin{aligned} M^T(\nabla_v \mathcal{J}^T) &= M_0^T \cdot M_1^T \cdot ..... \cdot M_\lambda^T \cdot ..... \cdot M_\Lambda^T \cdot \nabla_v \mathcal{J}^T \\ &= M_0^T \cdot M_1^T \cdot ..... \cdot \nabla_{v^{(\lambda)}} \mathcal{J}^T \\ &= \nabla_u \mathcal{J}^T \end{aligned}} \tag{7.10}$$

clearly expressing the reverse nature of the calculation. (7.10) is at the heart of automatic adjoint compilers. If the intermediate steps λ in (7.8) – (7.10) represent the model state (forward or adjoint) at each intermediate time step as noted above, then correspondingly, $M^T(\delta\vec{v}^{(\lambda)\,*}) = \delta\vec{v}^{(\lambda-1)\,*}$ for the adjoint variables. It thus becomes evident that the adjoint calculation also yields the adjoint of each model state component $\vec{v}^{(\lambda)}$ at each intermediate step λ, namely

$$\boxed{\begin{aligned} \nabla_{v^{(\lambda)}} \mathcal{J}^T|_{\vec{v}^{(\lambda)}} &= M_\lambda^T|_{\vec{v}^{(\lambda)}} \cdot ..... \cdot M_\Lambda^T|_{\vec{v}^{(\lambda)}} \cdot \delta\vec{v}^* \\ &= \delta\vec{v}^{(\lambda)\,*} \end{aligned}}$$

in close analogy to (7.7) we note in passing that the $\delta\vec{v}^{(\lambda)\,*}$ are the Lagrange multipliers of the model equations which determine $\vec{v}^{(\lambda)}$.

In components, (7.7) reads as follows. Let

$$\begin{aligned} \delta\vec{u} &= (\delta u_1, \ldots, \delta u_m)^T, \qquad \delta\vec{u}^* = \nabla_u \mathcal{J}^T = \left(\frac{\partial \mathcal{J}}{\partial u_1}, \ldots, \frac{\partial \mathcal{J}}{\partial u_m}\right)^T \\ \delta\vec{v} &= (\delta v_1, \ldots, \delta u_n)^T, \qquad \delta\vec{v}^* = \nabla_v \mathcal{J}^T = \left(\frac{\partial \mathcal{J}}{\partial v_1}, \ldots, \frac{\partial \mathcal{J}}{\partial v_n}\right)^T \end{aligned}$$

denote the perturbations in $\vec{u}$ and $\vec{v}$, respectively, and their adjoint variables; further

$$M = \begin{pmatrix} \frac{\partial \mathcal{M}_1}{\partial u_1} & \cdots & \frac{\partial \mathcal{M}_1}{\partial u_m} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{M}_n}{\partial u_1} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_m} \end{pmatrix}$$

is the Jacobi matrix of $\mathcal{M}$ (an $n \times m$ matrix) such that $\delta\vec{v} = M \cdot \delta\vec{u}$, or

$$\delta v_j = \sum_{i=1}^{m} M_{ji}\, \delta u_i = \sum_{i=1}^{m} \frac{\partial \mathcal{M}_j}{\partial u_i} \delta u_i$$

Then (7.7) takes the form

$$\delta u_i^* = \sum_{j=1}^{n} M_{ji}\, \delta v_j^* = \sum_{j=1}^{n} \frac{\partial \mathcal{M}_j}{\partial u_i} \delta v_j^*$$

or

$$\begin{pmatrix} \frac{\partial}{\partial u_1}\mathcal{J}\Big|_{\vec{u}^{(0)}} \\ \vdots \\ \frac{\partial}{\partial u_m}\mathcal{J}\Big|_{\vec{u}^{(0)}} \end{pmatrix} = \begin{pmatrix} \frac{\partial \mathcal{M}_1}{\partial u_1}\Big|_{\vec{u}^{(0)}} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_1}\Big|_{\vec{u}^{(0)}} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{M}_1}{\partial u_m}\Big|_{\vec{u}^{(0)}} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_m}\Big|_{\vec{u}^{(0)}} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial}{\partial v_1}\mathcal{J}\Big|_{\vec{v}} \\ \vdots \\ \frac{\partial}{\partial v_n}\mathcal{J}\Big|_{\vec{v}} \end{pmatrix}$$

Furthermore, the adjoint $\delta v^{(\lambda)*}$ of any intermediate state $v^{(\lambda)}$ may be obtained, using the intermediate Jacobian (an $n_{\lambda+1} \times n_\lambda$ matrix)

$$M_\lambda = \begin{pmatrix} \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_{n_\lambda}^{(\lambda)}} \\ \vdots & & \vdots \\ \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_{n_\lambda}^{(\lambda)}} \end{pmatrix}$$

and the shorthand notation for the adjoint variables $\delta v_j^{(\lambda)*} = \frac{\partial}{\partial v_j^{(\lambda)}}\mathcal{J}^T$, $j = 1, \ldots, n_\lambda$, for intermediate components, yielding

$$\begin{pmatrix} \delta v_1^{(\lambda)*} \\ \vdots \\ \delta v_{n_\lambda}^{(\lambda)*} \end{pmatrix} = \begin{pmatrix} \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_1^{(\lambda)}} & \cdots \cdots & \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_1^{(\lambda)}} \\ \vdots & & \vdots \\ \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_{n_\lambda}^{(\lambda)}} & \cdots \cdots & \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_{n_\lambda}^{(\lambda)}} \end{pmatrix} \cdot$$

$$\begin{pmatrix} \frac{\partial (\mathcal{M}_{\lambda+1})_1}{\partial v_1^{(\lambda+1)}} & \cdots & \frac{\partial (\mathcal{M}_{\lambda+1})_{n_{\lambda+2}}}{\partial v_1^{(\lambda+1)}} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \frac{\partial (\mathcal{M}_{\lambda+1})_1}{\partial v_{n_{\lambda+1}}^{(\lambda+1)}} & \cdots & \frac{\partial (\mathcal{M}_{\lambda+1})_{n_{\lambda+2}}}{\partial v_{n_{\lambda+1}}^{(\lambda+1)}} \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} \delta v_1^* \\ \vdots \\ \delta v_n^* \end{pmatrix}$$

(7.9) and (7.10) are perhaps clearest in showing the advantage of the reverse over the forward mode if the gradient $\nabla_u \mathcal{J}$, i.e., the sensitivity of the cost function $\mathcal{J}$ with respect to *all* input variables u (or the sensitivity of the cost function with respect to *all* intermediate states $\vec{v}^{(\lambda)}$) are sought. In order to be able to solve for each component of the gradient $\partial\mathcal{J}/\partial u_i$ in (7.9) a forward calculation has to be performed for each component separately, i.e., $\delta\vec{u} = \delta u_i \vec{e}_i$ for the i-th forward calculation. Then, (7.9) represents the projection of $\nabla_u \mathcal{J}$ onto the i-th component. The full gradient is retrieved from the m forward calculations. In contrast, (7.10) yields the full gradient $\nabla_u \mathcal{J}$ (and all intermediate gradients $\nabla_{v^{(\lambda)}} \mathcal{J}$) within a single reverse calculation.

Note, that if $\mathcal{J}$ is a vector-valued function of dimension $l > 1$, (7.10) has to be modified according to

$$M^T \left(\nabla_v \mathcal{J}^T \left(\delta\vec{J} \right) \right) \; = \; \nabla_u \mathcal{J}^T \cdot \delta\vec{J}$$

where now $\delta\vec{J} \in \mathbb{R}^l$ is a vector of dimension l. In this case l reverse simulations have to be performed for each $\delta J_k,\ k = 1, \ldots, l$. Then, the reverse mode is more efficient as long as $l < n$, otherwise the forward mode is preferable. Strictly, the reverse mode is called adjoint mode only for $l = 1$.

A detailed analysis of the underlying numerical operations shows that the computation of $\nabla_u \mathcal{J}$ in this way requires about two to five times the computation of the cost function. Alternatively, the gradient vector could be approximated by finite differences, requiring m computations of the perturbed cost function.

To conclude, we give two examples of commonly used types of cost functions:

7.1.2.1 Example 1: $\mathcal{J} = v_j(T)$

The cost function consists of the j-th component of the model state $\vec{v}$ at time T. Then $\nabla_v \mathcal{J}^T = \vec{f}_j$ is just the j-th unit vector. The $\nabla_u \mathcal{J}^T$ is the projection of the adjoint operator onto the j-th component $\mathbf{f_j}$,

$$\nabla_u \mathcal{J}^T \; = \; M^T \cdot \nabla_v \mathcal{J}^T \; = \; \sum_i M^T_{ji} \, \vec{e}_i$$

7.1.2.2 Example 2: $\mathcal{J} = \langle\, \mathcal{H}(\vec{v}) - \vec{d},\, \mathcal{H}(\vec{v}) - \vec{d} \,\rangle$

The cost function represents the quadratic model vs. data misfit. Here, $\vec{d}$ is the data vector and $\mathcal{H}$ represents the operator which maps the model state space onto the data space. Then, $\nabla_v \mathcal{J}$ takes the form

$$\begin{aligned} \nabla_v \mathcal{J}^T \; &= \; 2 \; H \cdot \left(\mathcal{H}(\vec{v}) - \vec{d} \right) \\ &= \; 2 \sum_j \left\{ \sum_k \frac{\partial \mathcal{H}_k}{\partial v_j} \left(\mathcal{H}_k(\vec{v}) - d_k \right) \right\} \; \vec{f}_j \end{aligned}$$

where $H_{kj} = \partial\mathcal{H}_k/\partial v_j$ is the Jacobi matrix of the data projection operator. Thus, the gradient $\nabla_u \mathcal{J}$ is given by the adjoint operator, driven by the model vs. data misfit:

$$\nabla_u \mathcal{J}^T \; = \; 2 \, M^T \cdot H \cdot \left(\mathcal{H}(\vec{v}) - \vec{d} \right)$$

7.1.3 Storing vs. recomputation in reverse mode

We note an important aspect of the forward vs. reverse mode calculation. Because of the local character of the derivative (a derivative is defined with respect to a point along the trajectory), the intermediate results of the model trajectory $\vec{v}^{(\lambda+1)} = \mathcal{M}_\lambda(v^{(\lambda)})$ may be required to evaluate the intermediate Jacobian $M_\lambda|_{\vec{v}^{(\lambda)}} \, \delta\vec{v}^{(\lambda)}$. This is the case for example for nonlinear expressions (momentum advection, nonlinear equation of state), and state-dependent conditional statements (parameterization schemes). In the forward mode, the intermediate results are required in

the same order as computed by the full forward model $\mathcal{M}$, but in the reverse mode they are required in the reverse order. Thus, in the reverse mode the trajectory of the forward model integration $\mathcal{M}$ has to be stored to be available in the reverse calculation. Alternatively, the complete model state up to the point of evaluation has to be recomputed whenever its value is required.

A method to balance the amount of recomputations vs. storage requirements is called checkpointing (e.g., Griewank, 1992 [Gri92], Restrepo et al., 1998 [RLG98]). It is depicted in Figure 7.1 for a 3-level checkpointing (as an example, we give explicit numbers for a 3-day integration with a 1-hourly timestep in square brackets).

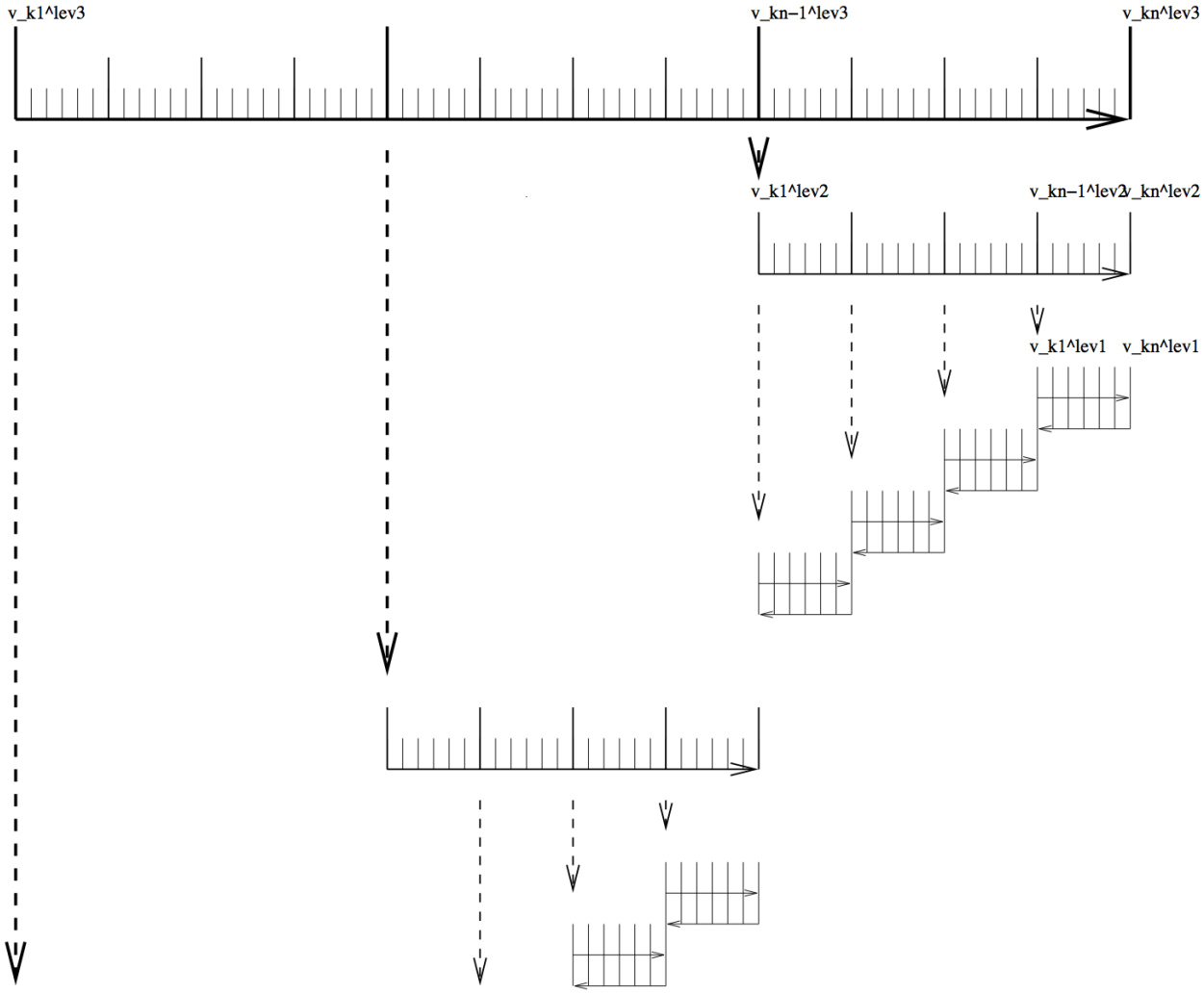

Figure 7.1: Schematic view of intermediate dump and restart for 3-level checkpointing.

- In a first step, the model trajectory is subdivided into n^{lev3} subsections [n^{lev3}=3 1-day intervals], with the label $lev3$ for this outermost loop. The model is then integrated along the full trajectory, and the model state stored to disk only at every k_i^{lev3}-th timestep [i.e. 3 times, at $i = 0, 1, 2$ corresponding to $k_i^{lev3} = 0, 24, 48$]. In addition, the cost function is computed, if needed.
- In a second step each subsection itself is divided into n^{lev2} subsections [n^{lev2}=4 6-hour intervals per subsection]. The model picks up at the last outermost dumped state $v_{k_n^{lev3}}$ and is integrated forward in time along the last subsection, with the label $lev2$ for this intermediate loop. The model state is now stored to disk at every k_i^{lev2}-th timestep [i.e. 4 times, at $i = 0, 1, 2, 3$ corresponding to $k_i^{lev2} = 48, 54, 60, 66$].

- Finally, the model picks up at the last intermediate dump state $v_{k_n^{lev2}}$ and is integrated forward in time along the last subsection, with the label $lev1$ for this intermediate loop. Within this sub-subsection only, parts of the model state are stored to memory at every timestep [i.e. every hour $i = 0, ..., 5$ corresponding to $k_i^{lev1} = 66, 67, \ldots, 71$]. The final state $v_n = v_{k_n^{lev1}}$ is reached and the model state of all preceding timesteps along the last innermost subsection are available, enabling integration backwards in time along the last subsection. The adjoint can thus be computed along this last subsection k_n^{lev2}.

This procedure is repeated consecutively for each previous subsection $k_{n-1}^{lev2}, \ldots, k_1^{lev2}$ carrying the adjoint computation to the initial time of the subsection k_n^{lev3}. Then, the procedure is repeated for the previous subsection k_{n-1}^{lev3} carrying the adjoint computation to the initial time k_1^{lev3}.

For the full model trajectory of $n^{lev3} \cdot n^{lev2} \cdot n^{lev1}$ timesteps the required storing of the model state was significantly reduced to $n^{lev2} + n^{lev3}$ to disk and roughly n^{lev1} to memory (i.e., for the 3-day integration with a total of 72 timesteps the model state was stored 7 times to disk and roughly 6 times to memory). This saving in memory comes at a cost of a required 3 full forward integrations of the model (one for each checkpointing level). The optimal balance of storage vs. recomputation certainly depends on the computing resources available and may be adjusted by adjusting the partitioning among the $n^{lev3},\ n^{lev2},\ n^{lev1}$.

7.2 TLM and ADM generation in general

In this section we describe in a general fashion the parts of the code that are relevant for automatic differentiation using the software tool TAF. Modifications to use OpenAD are described in Section 7.5.

The basic flow is as follows:

```
the_model_main
|
|--- initialise_fixed
|
|--- #ifdef ALLOW_ADJOINT_RUN
|           |
|           |--- ctrl_unpack
|           |
|           |--- adthe_main_loop
|           |    |
|           |    |--- initialise_varia
|           |    |--- ctrl_map_forcing
|           |    |--- do iloop = 1, nTimeSteps
|           |    |       |--- forward_step
|           |    |       |--- cost_tile
|           |    |    end do
|           |    |--- cost_final
|           |    |
|           |    |--- adcost_final
|           |    |--- do iloop = nTimeSteps, 1, -1
|           |    |       |--- adcost_tile
|           |    |       |--- adforward_step
|           |    |    end do
|           |    |--- adctrl_map_forcing
|           |    |--- adinitialise_varia
|           |    o
|           |
|           |--- ctrl_pack
|           |
|--- #else
|           |
```

(continues on next page)

(continued from previous page)

```
|           |--- the_main_loop
|           |
|    #endif
|
|--- #ifdef ALLOW_GRADIENT_CHECK
|           |
|           |--- grdchk_main
|           o
|    #endif
o
```

If CPP option ALLOW_AUTODIFF_TAMC is defined, the driver routine the_model_main.F, instead of calling the_model_loop.F, invokes the adjoint of this routine, `adthe_main_loop.F` (case #define ALLOW_ADJOINT_RUN, or the tangent linear of this routine `g_the_main_loop.F` (case #define ALLOW_TANGENTLINEAR_RUN), which are the toplevel routines in terms of automatic differentiation. The routines `adthe_main_loop.F` or `g_the_main_loop.F` are generated by TAF. It contains both the forward integration of the full model, the cost function calculation, any additional storing that is required for efficient checkpointing, and the reverse integration of the adjoint model.

[DESCRIBE IN A SEPARATE SECTION THE WORKING OF THE TLM]

The above structure of `adthe_main_loop.F` has been strongly simplified to focus on the essentials; in particular, no checkpointing procedures are shown here. Prior to the call of `adthe_main_loop.F`, the routine ctrl_unpack.F is invoked to unpack the control vector or initialize the control variables. Following the call of `adthe_main_loop.F`, the routine ctrl_pack.F is invoked to pack the control vector (cf. Section 7.2.5). If gradient checks are to be performed, the option #define ALLOW_GRDCHK is chosen. In this case the driver routine grdchk_main.F is called after the gradient has been computed via the adjoint (cf. Section 7.3).

7.2.1 General setup

In order to configure AD-related setups the following packages need to be enabled:

- pkg/autodiff
- pkg/ctrl
- pkg/cost
- pkg/grdchk

The packages are enabled by adding them to your experiment-specific configuration file `packages.conf` (see Section ???).

The following AD-specific CPP option files need to be customized:

- ECCO_CPPOPTIONS.h This header file collects CPP options for pkg/autodiff, pkg/cost, pkg/ctrl as well as AD-unrelated options for the external forcing package pkg/exf. (NOTE: These options are not set in their package-specific headers such as COST_OPTIONS.h, but are instead collected in the single header file ECCO_CPPOPTIONS.h. The package-specific header files serve as simple placeholders at this point.)
- tamc.h This header configures the splitting of the time stepping loop with respect to the 3-level checkpointing (see section ???).

7.2.2 Building the AD code using TAF

The build process of an AD code is very similar to building the forward model. However, depending on which AD code one wishes to generate, and on which AD tool is available (TAF or TAMC), the following make targets are

available:

AD-target	*output*	*description*
«MODE»«TOOL»only	«MODE»_«TOOL»_output.f	generates code for «MODE» using «TOOL»
		no make dependencies on .F .h
		useful for compiling on remote platforms
«MODE»«TOOL»	«MODE»_«TOOL»_output.f	generates code for «MODE» using «TOOL»
		includes make dependencies on .F .h
		i.e. input for «TOOL» may be re-generated
«MODE»all	mitgcmuv_«MODE»	generates code for «MODE» using «TOOL»
		and compiles all code
		(use of TAF is set as default)

Here, the following placeholders are used:

- «TOOL»
  - TAF
  - TAMC
- «MODE»
  - ad generates the adjoint model (ADM)
  - ftl generates the tangent linear model (TLM)
  - svd generates both ADM and TLM for singular value decomposition (SVD) type calculations

For example, to generate the adjoint model using TAF after routines (`.F`) or headers (`.h`) have been modified, but without compilation, type `make adtaf`; or, to generate the tangent linear model using TAMC without re-generating the input code, type `make ftltamconly`.

A typical full build process to generate the ADM via TAF would look like follows:

```
% mkdir build
% cd build
% ../../../tools/genmake2 -mods=../code_ad
% make depend
% make adall
```

7.2.3 The AD build process in detail

The `make «MODE»all` target consists of the following procedures:

1. A header file `AD_CONFIG.h` is generated which contains a CPP option on which code ought to be generated. Depending on the `make` target, the contents is one of the following:
   - #define ALLOW_ADJOINT_RUN
   - #define ALLOW_TANGENTLINEAR_RUN
   - #define ALLOW_ECCO_OPTIMIZATION
2. A single file `«MODE»_input_code.f` is concatenated consisting of all `.f` files that are part of the list `AD_FILES` and all `.flow` files that are part of the list `AD_FLOW_FILES`.
3. The AD tool is invoked with the `«MODE»_«TOOL»_FLAGS`. The default AD tool flags in genmake2 can be overwritten by a tools/adjoint_options file (similar to the platform-specific tools/build_options, see Section 3.5.2.2). The AD tool writes the resulting AD code into the file `«MODE»_input_code_ad.f`.

4. A short sed script tools/adjoint_sed is applied to «MODE»_input_code_ad.f to reinstate myThid into the CALL argument list of active file I/O. The result is written to file «MODE»_«TOOL»_output.f.
5. All routines are compiled and an executable is generated.

7.2.3.1 The list AD_FILES and .list files

Not all routines are presented to the AD tool. Routines typically hidden are diagnostics routines which do not influence the cost function, but may create artificial flow dependencies such as I/O of active variables.

genmake2 generates a list (or variable) AD_FILES which contains all routines that are shown to the AD tool. This list is put together from all files with suffix .list that genmake2 finds in its search directories. The list file for the core MITgcm routines is model/src/model_ad_diff.list Note that no wrapper routine is shown to TAF. These are either not visible at all to the AD code, or hand-written AD code is available (see next section).

Each package directory contains its package-specific list file «PKG»_ad_diff.list. For example, pkg/ptracers contains the file ptracers_ad_diff.list. Thus, enabling a package will automatically extend the AD_FILES list of genmake2 to incorporate the package-specific routines. Note that you will need to regenerate the makefile if you enable a package (e.g., by adding it to packages.conf) and a Makefile already exists.

7.2.3.2 The list AD_FLOW_FILES and .flow files

TAMC and TAF can evaluate user-specified directives that start with a specific syntax (CADJ, C$TAF, !$TAF). The main categories of directives are STORE directives and FLOW directives. Here, we are concerned with flow directives, store directives are treated elsewhere.

Flow directives enable the AD tool to evaluate how it should treat routines that are 'hidden' by the user, i.e. routines which are not contained in the AD_FILES list (see previous section), but which are called in part of the code that the AD tool does see. The flow directive tell the AD tool:

- which subroutine arguments are input/output
- which subroutine arguments are active
- which subroutine arguments are required to compute the cost
- which subroutine arguments are dependent

The syntax for the flow directives can be found in the AD tool manuals.

genmake2 generates a list (or variable) AD_FLOW_FILES which contains all files with suffix.flow that it finds in its search directories. The flow directives for the core MITgcm routines of eesupp/src/ and model/src/ reside in pkg/autodiff/. This directory also contains hand-written adjoint code for the MITgcm WRAPPER (Section 6.2).

Flow directives for package-specific routines are contained in the corresponding package directories in the file «PKG»_ad.flow, e.g., ptracers-specific directives are in ptracers_ad.flow.

7.2.3.3 Store directives for 3-level checkpointing

The storing that is required at each period of the 3-level checkpointing is controlled by three top-level headers.

```
do ilev_3 = 1, nchklev_3
#  include ``checkpoint_lev3.h''
   do ilev_2 = 1, nchklev_2
#     include ``checkpoint_lev2.h''
      do ilev_1 = 1, nchklev_1
#        include ``checkpoint_lev1.h''
```

(continues on next page)

(continued from previous page)

```
...

      end do
   end do
end do
```

All files `checkpoint_lev?.h` are contained in directory pkg/autodiff/.

7.2.3.4 Changing the default AD tool flags: ad_options files

7.2.3.5 Hand-written adjoint code

7.2.4 The cost function (dependent variable)

The cost function $\mathcal{J}$ is referred to as the *dependent variable*. It is a function of the input variables $\vec{u}$ via the composition $\mathcal{J}(\vec{u}) = \mathcal{J}(M(\vec{u}))$. The input are referred to as the *independent variables* or *control variables*. All aspects relevant to the treatment of the cost function $\mathcal{J}$ (parameter setting, initialization, accumulation, final evaluation), are controlled by the package pkg/cost. The aspects relevant to the treatment of the independent variables are controlled by the package pkg/ctrl and will be treated in the next section.

```
 the_model_main
 |
 |-- initialise_fixed
 |   |
 |   |-- packages_readparms
 |       |
 |       |-- cost_readparms
 |       o
 |
 |-- the_main_loop
...  |
     |-- initialise_varia
     |   |
     |   |-- packages_init_variables
     |       |
     |       |-- cost_init
     |       o
     |
     |-- do iloop = 1,nTimeSteps
     |      |-- forward_step
     |      |-- cost_tile
     |      |   |
     |      |   |-- cost_tracer
     |   end do
     |
     |-- cost_final
     o
```

7.2.4.1 Enabling the package

pkg/cost is enabled by adding the line `cost` to your file `packages.conf` (see Section ???).

In general the following packages ought to be enabled simultaneously: pkg/autodiff, pkg/ctrl, and pkg/cost. The basic CPP option to enable the cost function is ALLOW_COST. Each specific cost function contribution has its own option. For the present example the option is ALLOW_COST_TRACER. All cost-specific options are set in ECCO_CPPOPTIONS.h Since the cost function is usually used in conjunction with automatic differentiation, the CPP option ALLOW_ADJOINT_RUN (file CPP_OPTIONS.h) and ALLOW_AUTODIFF_TAMC (file ECCO_CPPOPTIONS.h) should be defined.

7.2.4.2 Initialization

The initialization of pkg/cost is readily enabled as soon as the CPP option ALLOW_COST is defined.

- The S/R cost_readparms.F reads runtime flags and parameters from file `data.cost`. For the present example the only relevant parameter read is mult_tracer. This multiplier enables different cost function contributions to be switched on (`= 1.`) or off (`= 0.`) at runtime. For more complex cost functions which involve model vs. data misfits, the corresponding data filenames and data specifications (start date and time, period, ...) are read in this S/R.
- The S/R cost_init_varia.F initializes the different cost function contributions. The contribution for the present example is objf_tracer which is defined on each tile (bi,bj).

7.2.4.3 Accumulation

The 'driver' routine cost_tile.F is called at the end of each time step. Within this 'driver' routine, S/R are called for each of the chosen cost function contributions. In the present example (ALLOW_COST_TRACER), S/R cost_tracer.F is called. It accumulates objf_tracer according to eqn. (ref:ask-the-author).

7.2.4.4 Finalize all contributions

At the end of the forward integration S/R cost_final.F is called. It accumulates the total cost function fc from each contribution and sums over all tiles:

$$\mathcal{J} = \text{fc} = \text{mult\_tracer} \sum_{\text{global sum}} \sum_{bi,\,bj}^{nSx,\,nSy} \text{objf\_tracer}(bi,bj) + \ldots$$

The total cost function fc will be the 'dependent' variable in the argument list for TAF, i.e.,

```
taf -output 'fc' ...
```

```
 *************
 the_main_loop
 *************
 |
 |--- initialise_varia
 |     |
 |    ...
 |     |--- packages_init_varia
 |     |     |
 |     |    ...
 |     |     |--- #ifdef ALLOW_ADJOINT_RUN
 |     |     |           call ctrl_map_ini
 |     |     |           call cost_ini
 |     |     |     #endif
 |     |    ...
```

(continues on next page)

(continued from previous page)

```
 |    |    o
 |   ...
 |    o
...
 |--- #ifdef ALLOW_ADJOINT_RUN
 |          call ctrl_map_forcing
 |    #endif
...
 |--- #ifdef ALLOW_TAMC_CHECKPOINTING
            do ilev_3 = 1,nchklev_3
 |            do ilev_2 = 1,nchklev_2
 |              do ilev_1 = 1,nchklev_1
 |                iloop = (ilev_3-1)*nchklev_2*nchklev_1 +
 |                        (ilev_2-1)*nchklev_1           + ilev_1
 |    #else
 |          do iloop = 1, nTimeSteps
 |    #endif
 |    |
 |    |---       call forward_step
 |    |
 |    |--- #ifdef ALLOW_COST
 |    |           call cost_tile
 |    |    #endif
 |    |
 |    |    enddo
 |    o
 |
 |--- #ifdef ALLOW_COST
 |          call cost_final
 |    #endif
 o
```

7.2.5 The control variables (independent variables)

The control variables are a subset of the model input (initial conditions, boundary conditions, model parameters). Here we identify them with the variable $\vec{u}$. All intermediate variables whose derivative with respect to control variables do not vanish are called active variables. All subroutines whose derivative with respect to the control variables don't vanish are called active routines. Read and write operations from and to file can be viewed as variable assignments. Therefore, files to which active variables are written and from which active variables are read are called active files. All aspects relevant to the treatment of the control variables (parameter setting, initialization, perturbation) are controlled by the package pkg/ctrl.

```
 the_model_main
 |
 |-- initialise_fixed
 |   |
 |   |-- packages_readparms
 |       |
 |       |-- cost_readparms
 |       o
 |
 |-- the_main_loop
...  |
      |-- initialise_varia
```

(continues on next page)

(continued from previous page)

```
|   |
|   |-- packages_init_variables
|       |
|       |-- cost_init
|       o
|
|-- do iloop = 1,nTimeSteps
|      |-- forward_step
|      |-- cost_tile
|      |   |
|      |   |-- cost_tracer
|   end do
|
|-- cost_final
o
```

7.2.5.1 genmake2 and CPP options

Package pkg/ctrl is enabled by adding the line `ctrl` to your file `packages.conf`. Each control variable is enabled via its own CPP option in ECCO_CPPOPTIONS.h.

7.2.5.2 Initialization

- The S/R ctrl_readparms.F reads runtime flags and parameters from file `data.ctrl`. For the present example the file contains the file names of each control variable that is used. In addition, the number of wet points for each control variable and the net dimension of the space of control variables (counting wet points only) nvarlength is determined. Masks for wet points for each tile (bi,bj) and vertical layer k are generated for the three relevant categories on the C-grid: nWetCtile for tracer fields, nWetWtile for zonal velocity fields, nWetStile for meridional velocity fields.
- Two important issues related to the handling of the control variables in MITgcm need to be addressed. First, in order to save memory, the control variable arrays are not kept in memory, but rather read from file and added to the initial fields during the model initialization phase. Similarly, the corresponding adjoint fields which represent the gradient of the cost function with respect to the control variables are written to file at the end of the adjoint integration. Second, in addition to the files holding the 2-D and 3-D control variables and the corresponding cost gradients, a 1-D control vector and gradient vector are written to file. They contain only the wet points of the control variables and the corresponding gradient. This leads to a significant data compression. Furthermore, an option is available (ALLOW_NONDIMENSIONAL_CONTROL_IO) to non-dimensionalize the control and gradient vector, which otherwise would contain different pieces of different magnitudes and units. Finally, the control and gradient vector can be passed to a minimization routine if an update of the control variables is sought as part of a minimization exercise.

The files holding fields and vectors of the control variables and gradient are generated and initialized in S/R ctrl_unpack.F.

7.2.5.3 Perturbation of the independent variables

The dependency flow for differentiation with respect to the controls starts with adding a perturbation onto the input variable, thus defining the independent or control variables for TAF. Three types of controls may be considered:

- Consider as an example the initial tracer distribution pTracer as control variable. After pTracer has been initialized in ptracers_init_varia.F (dynamical variables such as temperature and salinity are initialized in ini_fields.F),

a perturbation anomaly is added to the field in S/R ctrl_map_ini.F:

$$\begin{aligned} u &= u_{[0]} + \Delta u \\ \mathbf{tr1}(...) &= \mathbf{tr1_{ini}}(...) + \mathbf{xx\_tr1}(...) \end{aligned} \tag{7.11}$$

xx_tr1 is a 3-D global array holding the perturbation. In the case of a simple sensitivity study this array is identical to zero. However, it's specification is essential in the context of automatic differentiation since TAF treats the corresponding line in the code symbolically when determining the differentiation chain and its origin. Thus, the variable names are part of the argument list when calling TAF:

```
taf -input 'xx_tr1 ...' ...
```

Now, as mentioned above, MITgcm avoids maintaining an array for each control variable by reading the perturbation to a temporary array from file. To ensure the symbolic link to be recognized by TAF, a scalar dummy variable `xx_tr1_dummy` is introduced and an 'active read' routine of the adjoint support package pkg/autodiff is invoked. The read-procedure is tagged with the variable `xx_tr1_dummy` enabling TAF to recognize the initialization of the perturbation. The modified call of TAF thus reads

```
taf -input 'xx_tr1_dummy ...' ...
```

and the modified operation (to perturb) in the code takes on the form

```
  call active_read_xyz(
&       ..., tmpfld3d, ..., xx_tr1_dummy, ... )

  tr1(...) = tr1(...) + tmpfld3d(...)
```

Note that reading an active variable corresponds to a variable assignment. Its derivative corresponds to a write statement of the adjoint variable, followed by a reset. The 'active file' routines have been designed to support active read and corresponding adjoint active write operations (and vice versa).

- The handling of boundary values as control variables proceeds exactly analogous to the initial values with the symbolic perturbation taking place in S/R ctrl_map_forcing.F. Note however an important difference: Since the boundary values are time dependent with a new forcing field applied at each time step, the general problem may be thought of as a new control variable at each time step (or, if the perturbation is averaged over a certain period, at each N timesteps), i.e.,

$$u_{\mathrm{forcing}} = \{\, u_{\mathrm{forcing}}(t_n) \,\}_{n\,=\,1,\ldots,\mathrm{nTimeSteps}}$$

  In the current example an equilibrium state is considered, and only an initial perturbation to surface forcing is applied with respect to the equilibrium state. A time dependent treatment of the surface forcing is implemented in the ECCO environment, involving the calendar (pkg/cal) and external forcing (pkg/exf) packages.

- This routine is not yet implemented, but would proceed proceed along the same lines as the initial value sensitivity. The mixing parameters diffkr and kapgm are currently added as controls in ctrl_map_ini.F.

7.2.5.4 Output of adjoint variables and gradient

Several ways exist to generate output of adjoint fields.

- In ctrl_map_ini.F, ctrl_map_forcing.F:
  - The control variable fields `xx\_«...»`: before the forward integration, the control variables are read from file «`xx\_ ...`» and added to the model field.
  - The adjoint variable fields `adxx\_«...»`, i.e., the gradient $\nabla_u \mathcal{J}$ for each control variable: after the adjoint integration the corresponding adjoint variables are written to `adxx\_«...»`.

- In ctrl_unpack.F, ctrl_pack.F:
  - The control vector `vector_ctrl`: at the very beginning of the model initialization, the updated compressed control vector is read (or initialized) and distributed to 2-D and 3-D control variable fields.
  - The gradient vector `vector_grad`: at the very end of the adjoint integration, the 2-D and 3-D adjoint variables are read, compressed to a single vector and written to file.
- In addition to writing the gradient at the end of the forward/adjoint integration, many more adjoint variables of the model state at intermediate times can be written using S/R addummy_in_stepping.F. The procedure is enabled using via the CPP-option ALLOW_AUTODIFF_MONITOR (file ECCO_CPPOPTIONS.h). To be part of the adjoint code, the corresponding S/R dummy_in_stepping.F has to be called in the forward model (S/R the_main_loop.F) at the appropriate place. The adjoint common blocks are extracted from the adjoint code via the header file adcommon.h.

  dummy_in_stepping.F is essentially empty, the corresponding adjoint routine is hand-written rather than generated automatically. Appropriate flow directives (dummy_in_stepping.flow) ensure that TAMC does not automatically generate addummy_in_stepping.F by trying to differentiate dummy_in_stepping.F, but instead refers to the hand-written routine.

  dummy_in_stepping.F is called in the forward code at the beginning of each timestep, before the call to model/src/dynamics.F, thus ensuring that addummy_in_stepping.F is called at the end of each timestep in the adjoint calculation, after the call to addummy_in_dynamics.F.

  addummy_in_stepping.F includes the header files adcommon.h. This header file is also hand-written. It contains the common blocks addynvars_r, addynvars_cd, addynvars_diffkr, addynvars_kapgm, adtr1_r, adffields, which have been extracted from the adjoint code to enable access to the adjoint variables.

  WARNING: If the structure of the common blocks dynvars_r, dynvars_cd, etc., changes similar changes will occur in the adjoint common blocks. Therefore, consistency between the TAMC-generated common blocks and those in adcommon.h have to be checked.

7.2.5.5 Control variable handling for optimization applications

In optimization mode the cost function $\mathcal{J}(u)$ is sought to be minimized with respect to a set of control variables $\delta\mathcal{J} = 0$, in an iterative manner. The gradient $\nabla_u \mathcal{J}|_{u_{[k]}}$ together with the value of the cost function itself $\mathcal{J}(u_{[k]})$ at iteration step k serve as input to a minimization routine (e.g. quasi-Newton method, conjugate gradient, ... (Gilbert and Lemaréchal, 1989 [GLemarechal89]) to compute an update in the control variable for iteration step $k+1$:

$$u_{[k+1]} = u_{[0]} + \Delta u_{[k+1]} \quad \text{satisfying} \quad \mathcal{J}\left(u_{[k+1]}\right) < \mathcal{J}\left(u_{[k]}\right)$$

$u_{[k+1]}$ then serves as input for a forward/adjoint run to determine $\mathcal{J}$ and $\nabla_u \mathcal{J}$ at iteration step $k+1$. Figure 7.2 sketches the flow between forward/adjoint model and the minimization routine.

The routines ctrl_unpack.F and ctrl_pack.F provide the link between the model and the minimization routine. As described in Section ref:ask-the-author the ctrl_unpack.F and ctrl_pack.F routines read and write control and gradient vectors which are compressed to contain only wet points, in addition to the full 2-D and 3-D fields. The corresponding I/O flow is shown in Figure 7.3:

ctrl_unpack.F reads the updated control vector `vector_ctrl_<k>`. It distributes the different control variables to 2-D and 3-D files `xx_«...»<k>`. At the start of the forward integration the control variables are read from `xx_«...»<k>` and added to the field. Correspondingly, at the end of the adjoint integration the adjoint fields are written to `adxx_«...»<k>`, again via the active file routines. Finally, ctrl_pack.F collects all adjoint files and writes them to the compressed vector file `vector_grad_<k>`.

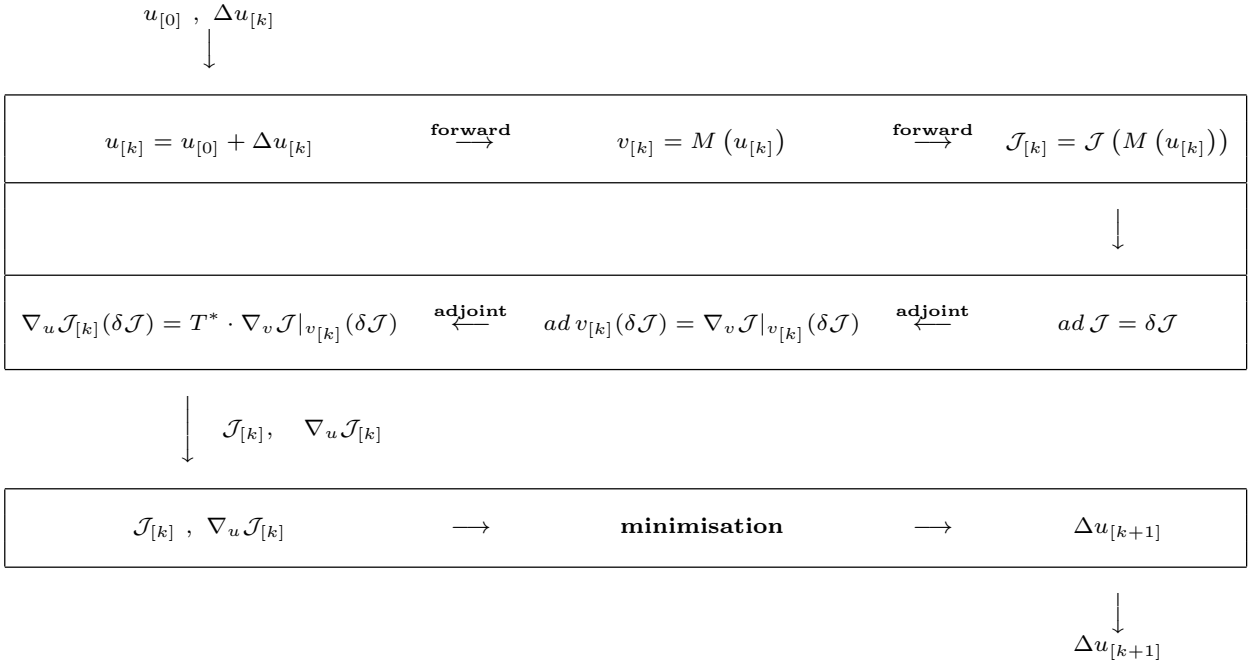

Figure 7.2: Flow between the forward/adjoint model and the minimization routine.

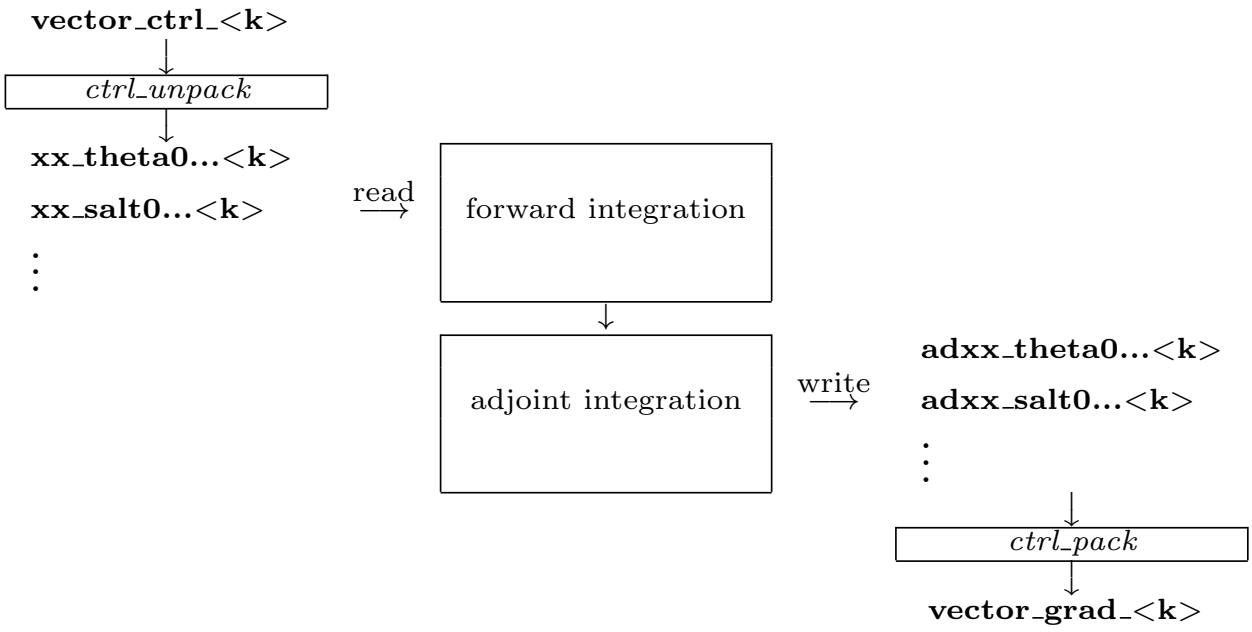

Figure 7.3: Flow chart showing I/O in the forward/adjoint model.

NOTE: These options are not set in their package-specific headers such as COST_OPTIONS.h, but are instead collected in the single header file ECCO_CPPOPTIONS.h. The package-specific header files serve as simple placeholders at this point.

7.3 The gradient check package

An indispensable test to validate the gradient computed via the adjoint is a comparison against finite difference gradients. The gradient check package pkg/grdchk enables such tests in a straightforward and easy manner. The driver routine grdchk_main.F is called from the_model_main.F after the gradient has been computed via the adjoint model (cf. flow chart ???).

The gradient check proceeds as follows: The $i-$th component of the gradient $(\nabla_u \mathcal{J}^T)_i$ is compared with the following finite-difference gradient:

$$\left(\nabla_u \mathcal{J}^T\right)_i \quad \text{vs.} \quad \frac{\partial \mathcal{J}}{\partial u_i} = \frac{\mathcal{J}(u_i + \epsilon) - \mathcal{J}(u_i)}{\epsilon}$$

A gradient check at point u_i may generally considered to be successful if the deviation of the ratio between the adjoint and the finite difference gradient from unity is less than 1 percent,

$$1 - \frac{(\mathrm{grad}\mathcal{J})_i(\text{adjoint})}{(\mathrm{grad}\mathcal{J})_i(\text{finite difference})} < 1\%$$

7.3.1 Code description

7.3.2 Code configuration

The relevant CPP precompile options are set in the following files:

- CPP_OPTIONS.h - Together with the flag ALLOW_ADJOINT_RUN, define the flag ALLOW_GRADIENT_CHECK.

The relevant runtime flags are set in the files:

- `data.pkg` - Set useGrdchk = `.TRUE.`
- `data.grdchk`
  - grdchk_eps
  - nbeg
  - nstep
  - nend
  - grdchkvarindex

```
the_model_main
|
|-- ctrl_unpack
|-- adthe_main_loop            - unperturbed cost function and
|-- ctrl_pack                    adjoint gradient are computed here
|
|-- grdchk_main
    |
    |-- grdchk_init
```

(continues on next page)

(continued from previous page)

```
   |-- do icomp=...           - loop over control vector elements
       |
       |-- grdchk_loc         - determine location of icomp on grid
       |
       |-- grdchk_getxx       - get control vector component from file
       |                        perturb it and write back to file
       |-- grdchk_getadxx     - get gradient component calculated
       |                        via adjoint
       |-- the_main_loop      - forward run and cost evaluation
       |                        with perturbed control vector element
       |-- calculate ratio of adj. vs. finite difference gradient
       |
       |-- grdchk_setxx       - Reset control vector element
       |
       |-- grdchk_print       - print results
```

7.4 Adjoint dump & restart – divided adjoint (DIVA)

Authors: Patrick Heimbach & Geoffrey Gebbie, 07-Mar-2003*

***NOTE:THIS SECTION IS SUBJECT TO CHANGE. IT REFERS TO TAF-1.4.26.**

Previous TAF versions are incomplete and have problems with both TAF options `-pure` and `-mpi`.

The code which is tuned to the DIVA implementation of this TAF version is `checkpoint50` (MITgcm) and `ecco_c50_e28` (ECCO).

7.4.1 Introduction

Most high performance computing (HPC) centers require the use of batch jobs for code execution. Limits in maximum available CPU time and memory may prevent the adjoint code execution from fitting into any of the available queues. This presents a serious limit for large scale / long time adjoint ocean and climate model integrations. The MITgcm itself enables the split of the total model integration into sub-intervals through standard dump/restart of/from the full model state. For a similar procedure to run in reverse mode, the adjoint model requires, in addition to the model state, the adjoint model state, i.e., all variables with derivative information which are needed in an adjoint restart. This adjoint dump & restart is also termed 'divided adjoint (DIVA)'.

For this to work in conjunction with automatic differentiation, an AD tool needs to perform the following tasks:

1. identify an adjoint state, i.e., those sensitivities whose accumulation is interrupted by a dump/restart and which influence the outcome of the gradient. Ideally, this state consists of
   - the adjoint of the model state,
   - the adjoint of other intermediate results (such as control variables, cost function contributions, etc.)
   - bookkeeping indices (such as loop indices, etc.)
2. generate code for storing and reading adjoint state variables
3. generate code for bookkeeping , i.e., maintaining a file with index information
4. generate a suitable adjoint loop to propagate adjoint values for dump/restart with a minimum overhead of adjoint intermediate values.

TAF (but not TAMC!) generates adjoint code which performs the above specified tasks. It is closely tied to the adjoint multi-level checkpointing. The adjoint state is dumped (and restarted) at each step of the outermost checkpointing level

and adjoint integration is performed over one outermost checkpointing interval. Prior to the adjoint computations, a full forward sweep is performed to generate the outermost (forward state) tapes and to calculate the cost function. In the current implementation, the forward sweep is immediately followed by the first adjoint leg. Thus, in theory, the following steps are performed (automatically)

- **1st model call:** This is the case if file `costfinal` does *not* exist. S/R `mdthe_main_loop.f` (generated by TAF) is called.
  1. calculate forward trajectory and dump model state after each outermost checkpointing interval to files `tapelev3`
  2. calculate cost function `fc` and write it to file `costfinal`
- **2nd and all remaining model calls:** This is the case if file costfinal *does* exist. S/R `adthe_main_loop.f` (generated by TAF) is called.
  1. (forward run and cost function call is avoided since all values are known)
     - if 1st adjoint leg: create index file `divided.ctrl` which contains info on current checkpointing index $ilev3$
     - if not i-th adjoint leg: adjoint picks up at $ilev3 = nlev3 - i + 1$ and runs to $nlev3 - i$
  2. perform adjoint leg from $nlev3 - i + 1$ to $nlev3 - i$
  3. dump adjoint state to file `snapshot`
  4. dump index file `divided.ctrl` for next adjoint leg
  5. in the last step the gradient is written.

A few modifications were performed in the forward code, obvious ones such as adding the corresponding TAF-directive at the appropriate place, and less obvious ones (avoid some re-initializations, when in an intermediate adjoint integration interval).

[For TAF-1.4.20 a number of hand-modifications were necessary to compensate for TAF bugs. Since we refer to TAF-1.4.26 onwards, these modifications are not documented here].

7.4.2 Recipe 1: single processor

1. In ECCO_CPPOPTIONS.h set:
   - #define ALLOW_DIVIDED_ADJOINT
   - #undef ALLOW_DIVIDED_ADJOINT_MPI
2. Generate adjoint code. Using the TAF option `-pure`, two codes are generated:
   - `mdthe_main_loop.f`: Is responsible for the forward trajectory, storing of outermost checkpoint levels to file, computation of cost function, and storing of cost function to file (1st step).
   - `adthe_main_loop.f`: Is responsible for computing one adjoint leg, dump adjoint state to file and write index info to file (2nd and consecutive steps).

     for adjoint code generation, e.g., add `-pure` to TAF option list

     ```
     make adtaf
     ```

   - One modification needs to be made to adjoint codes in S/R `adecco_the_main_loop.f` (generated by TAF):

     There's a remaining issue with the `-pure` option. The `call ad...` between `call ad...` and the read of the `snapshot` file should be called only in the first adjoint leg between $nlev3$ and $nlev3 - 1$. In

the ecco-branch, the following lines should be bracketed by an `if (idivbeg .GE. nchklev_3) then`, thus:

```
...
      xx_psbar_mean_dummy = onetape_xx_psbar_mean_dummy_3h(1)
      xx_tbar_mean_dummy = onetape_xx_tbar_mean_dummy_4h(1)
      xx_sbar_mean_dummy = onetape_xx_sbar_mean_dummy_5h(1)
      call barrier( mythid )
cAdd(
      if (idivbeg .GE. nchklev_3) then
cAdd)

      call adcost_final( mythid )
      call barrier( mythid )
      call adcost_sst( mythid )
      call adcost_ssh( mythid )
      call adcost_hyd( mythid )
      call adcost_averagesfields( mytime,myiter,mythid )
      call barrier( mythid )
cAdd(
      endif
cAdd)

C--------------------------------------------
C read snapshot
C--------------------------------------------
      if (idivbeg .lt. nchklev_3) then
        open(unit=77,file='snapshot',status='old',form='unformatted',
     $iostat=iers)
...
```

For the main code, in all likelihood the block which needs to be bracketed consists of `adcost_final.f` (generated by TAF) only.

- Now the code can be copied as usual to `adjoint_model.F` and then be compiled:

```
make adchange
```

then compile

7.4.3 Recipe 2: multi processor (MPI)

1. On the machine where you execute the code (most likely not the machine where you run TAF) find the includes directory for MPI containing `mpif.h`. Either copy `mpif.h` to the machine where you generate the `.f` files before TAF-ing, or add the path to the includes directory to your genmake2 platform setup, TAF needs some MPI parameter settings (essentially `mpi_comm_world` and `mpi_integer`) to incorporate those in the adjoint code.
2. In ECCO_CPPOPTIONS.h set
   - #define ALLOW_DIVIDED_ADJOINT
   - #define ALLOW_DIVIDED_ADJOINT_MPI

   This will include the header file `mpif.h` into the top level routine for TAF.
3. Add the TAF option `-mpi` to the TAF argument list in the makefile.
4. Follow the same steps as in *Recipe 1*.

7.5 Adjoint code generation using OpenAD

Authors: Jean Utke, Patrick Heimbach and Chris Hill

7.5.1 Introduction

The development of OpenAD was initiated as part of the ACTS (Adjoint Compiler Technology & Standards) project funded by the NSF Information Technology Research (ITR) program. The main goals for OpenAD initially defined for the ACTS project are:

1. develop a flexible, modular, open source tool that can generate adjoint codes of numerical simulation programs,
2. establish a platform for easy implementation and testing of source transformation algorithms via a language-independent abstract intermediate representation,
3. support for source code written in C and Fortan, and
4. generate efficient tangent linear and adjoint for the MIT general circulation model.

OpenAD's homepage is at http://www-unix.mcs.anl.gov/OpenAD. A development WIKI is at http://wiki.mcs.anl.gov/OpenAD/index.php/Main_Page. From the WIKI's main page, click on Handling GCM for various aspects pertaining to differentiating the MITgcm with OpenAD.

7.5.2 Downloading and installing OpenAD

The OpenAD webpage has a detailed description on how to download and build OpenAD. From its homepage, please click on Binaries. You may either download pre-built binaries for quick trial, or follow the detailed build process described at http://www.mcs.anl.gov/OpenAD/access.shtml.

7.5.3 Building MITgcm adjoint with OpenAD

17-January-2008

OpenAD was successfully built on head node of `itrda.acesgrid.org`, for following system:

```
> uname -a
Linux itrda 2.6.22.2-42.fc6 #1 SMP Wed Aug 15 12:34:26 EDT 2007 i686 i686 i386 GNU/
↪Linux

> cat /proc/version
Linux version 2.6.22.2-42.fc6 (brewbuilder@hs20-bc2-4.build.redhat.com)
(gcc version 4.1.2 20070626 (Red Hat 4.1.2-13)) #1 SMP Wed Aug 15 12:34:26 EDT 2007

> module load ifc/9.1.036 icc/9.1.042
```

Head of MITgcm branch (`checkpoint59m` with some modifications) was used for building adjoint code. Following routing needed special care (revert to revision 1.1): http://wwwcvs.mitgcm.org/viewvc/MITgcm/MITgcm_contrib/heimbach/OpenAD/OAD_support/active_module.f90?hideattic=0&view=markup.

CHAPTER 8

Packages I - Physical Parameterizations

In this chapter and in the following chapter, the MITgcm 'packages' are described. While you can carry out many experiments with MITgcm by starting from case studies in section Section 4, configuring a brand new experiment or making major changes to an experimental configuration requires some knowledge of the *packages* that make up the full MITgcm code. Packages are used in MITgcm to help organize and layer various code building blocks that are assembled and selected to perform a specific experiment. Each of the specific experiments described in section Section 4 uses a particular combination of packages.

Figure 8.1 shows the full set of packages that are available. As shown in the figure packages are classified into different groupings that layer on top of each other. The top layer packages are generally specialized to specific simulation types. In this layer there are packages that deal with biogeochemical processes, ocean interior and boundary layer processes, atmospheric processes, sea-ice, coupled simulations and state estimation. Below this layer are a set of general purpose numerical and computational packages. The general purpose numerical packages provide code for kernel numerical algorithms that apply to many different simulation types. Similarly, the general purpose computational packages implement non-numerical algorithms that provide parallelism, I/O and time-keeping functions that are used in many different scenarios.

The following sections describe the packages shown in Figure 8.1. Section Section 8.1.1 describes the general procedure for using any package in MITgcm. Sections Section 8 to Section 10 layout the algorithms implemented in specific packages and describe how to use the individual packages. A brief synopsis of the function of each package is given in Figure 8.1. Organizationally package code is assigned a separate subdirectory in the MITgcm code distribution (within the source code directory pkg). The name of this subdirectory is used as the package name in Figure 8.1.

8.1 Overview

8.1.1 Using MITgcm Packages

The set of packages that will be used within a particular model can be configured using a combination of both "compile–time" and "run–time" options. Compile–time options are those used to select which packages will be "compiled in" or implemented within the program. Packages excluded at compile time are completely absent from the executable program(s) and thus cannot be later activated by any set of subsequent run–time options.

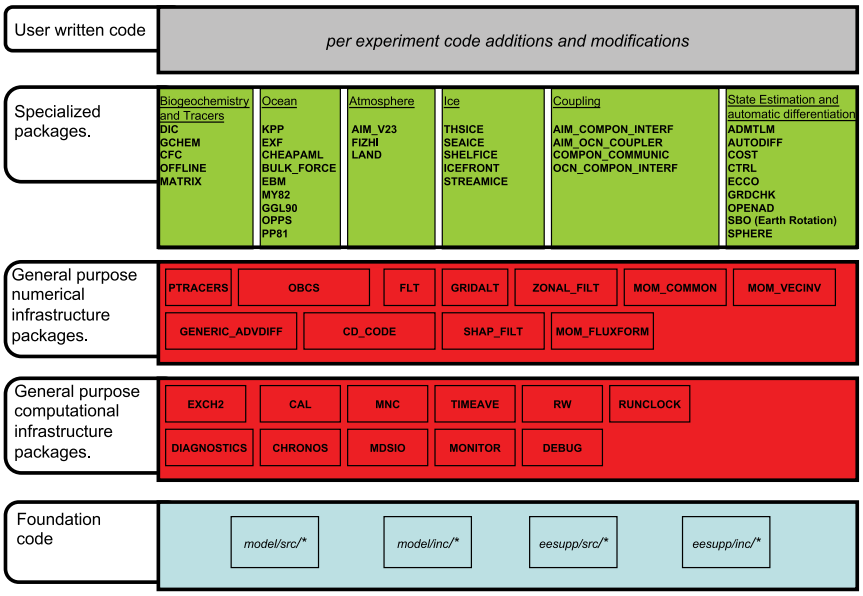

Figure 8.1: Hierarchy of code layers that are assembled to make up an MITgcm simulation. Conceptually (and in terms of code organization) MITgcm consists of several layers. At the base is a layer of core software that provides a basic numerical and computational foundation for MITgcm simulations. This layer is shown marked *Foundation Code* at the bottom of the figure and corresponds to code in the italicised subdirectories on the figure. This layer is not organized into packages. All code above the foundation layer is organized as packages. Much of the code in MITgcm is contained in packages which serve as a useful way of organizing and layering the different levels of functionality that make up the full MITgcm software distribution. The figure shows the different packages in MITgcm as boxes containing bold face upper case names. Directly above the foundation layer are two layers of general purpose infrastructure software that consist of computational and numerical packages. These general purpose packages can be applied to both online and offline simulations and are used in many different physical simulation types. Above these layers are more specialized packages.

Here we use the following shorthand for various forms of package names, i.e. that appear in package-related filenames, parameters etc.: all upper case `${PKG}`, all lower case `${pkg}`, and mixed case `${Pkg}`. For example, for pkg/gmredi these are `GMREDI`, `gmredi`, and `gmRedi` respectively.

8.1.1.1 Package Inclusion/Exclusion

There are numerous ways that one can specify compile–time package inclusion or exclusion and they are all implemented by the `genmake2` program which was previously described in Section Section 3.5. The options are as follows:

1. Setting the `genamake2` options `-enable PKG` and/or `-disable PKG` specifies inclusion or exclusion. This method is intended as a convenient way to perform a single (perhaps for a quick test) compilation.
2. By creating a text file with the name `packages.conf` in either the local build directory or the `-mods=DIR` directory, one can specify a list of packages (one package per line, with '#' as the comment character) to be included. Since the `packages.conf` file can be saved, this is the preferred method for setting and recording (for future reference) the package configuration.
3. For convenience, a list of "standard" package groups is contained in the `pkg/pkg_groups` file. By selecting one of the package group names in the `packages.conf` file, one automatically obtains all packages in that group.
4. By default (that is, if a `packages.conf` file is not found), the `genmake2` program will use the package group default "`default_pkg_list`" as defined in `pkg/pkg_groups` file.
5. To help prevent users from creating unusable package groups, the `genmake2` program will parse the contents of the `pkg/pkg_depend` file to determine:
   - whether any two requested packages cannot be simultaneously included (*eg. seaice* and *thsice* are mutually exclusive),
   - whether additional packages must be included in order to satisfy package dependencies (*eg. rw* depends upon functionality within the *mdsio* package), and
   - whether the set of all requested packages is compatible with the dependencies (and producing an error if they aren't).

   Thus, as a result of the dependencies, additional packages may be added to those originally requested.

8.1.1.2 Package Activation

For run–time package control, MITgcm uses flags set through a `data.pkg` file. While some packages (*eg.* `debug`, `mnc`, `exch2`) may have their own usage conventions, most follow a simple flag naming convention of the form:

```
usePackageName=.TRUE.
```

where the `usePackageName` variable can activate or disable the package at runtime. As mentioned previously, packages must be included in order to be activated. Generally, such mistakes will be detected and reported as errors by the code. However, users should still be aware of the dependency.

8.1.1.3 Package Coding Standards

The following sections describe how to modify and/or create new MITgcm packages.

Packages are Not Libraries

To a beginner, the MITgcm packages may resemble libraries as used in myriad software projects. While future versions are likely to implement packages as libraries (perhaps using FORTRAN90/95 syntax) the current packages (FORTRAN77) are **not** based upon any concept of libraries.

File Inclusion Rules

Instead, packages should be viewed only as directories containing "sets of source files" that are built using some simple mechanisms provided by `genmake2`. Conceptually, the build process adds files as they are found and proceeds according to the following rules:

1. `genmake2` locates a "core" or main set of source files (the `-standarddirs` option sets these locations and the default value contains the directories `eesupp` and `model`).
2. `genmake2` then finds additional source files by inspecting the contents of each of the package directories:
    1. As the new files are found, they are added to a list of source files.
    2. If there is a file name "collision" (that is, if one of the files in a package has the same name as one of the files previously encountered) then the file within the newer (more recently visited) package will superseed (or "hide") any previous file(s) with the same name.
    3. Packages are visited (and thus files discovered) *in the order that the packages are enabled* within `genmake2`. Thus, the files in `PackB` may superseed the files in `PackA` if `PackA` is enabled before `PackB`. Thus, package ordering can be significant! For this reason, `genmake2` honors the order in which packages are specified.

These rules were adopted since they provide a relatively simple means for rapidly including (or "hiding") existing files with modified versions.

Conditional Compilation and `PACKAGES_CONFIG.h`

Given that packages are simply groups of files that may be added or removed to form a whole, one may wonder how linking (that is, FORTRAN symbol resolution) is handled. This is the second way that `genmake2` supports the concept of packages. Basically, `genmake2` creates a `Makefile` that, in turn, is able to create a file called `PACKAGES_CONFIG.h` that contains a set of C pre-processor (or "CPP") directives such as:

```
#undef  ALLOW_KPP
#undef  ALLOW_LAND
...
#define ALLOW_GENERIC_ADVDIFF
#define ALLOW_MDSIO
...
```

These CPP symbols are then used throughout the code to conditionally isolate variable definitions, function calls, or any other code that depends upon the presence or absence of any particular package.

An example illustrating the use of these defines is:

```
#ifdef ALLOW_GMREDI
      IF (useGMRedi) CALL GMREDI_CALC_DIFF(
     I        bi,bj,iMin,iMax,jMin,jMax,K,
     I        maskUp,
     O        KappaRT,KappaRS,
```

(continues on next page)

(continued from previous page)

```
     I        myThid)
#endif
```

which is included from the file and shows how both the compile–time ALLOW_GMREDI flag and the run–time useGMRedi are nested.

There are some benefits to using the technique described here. The first is that code snippets or subroutines associated with packages can be placed or called from almost anywhere else within the code. The second benefit is related to memory footprint and performance. Since unused code can be removed, there is no performance penalty due to unnecessary memory allocation, unused function calls, or extra run-time IF (...) conditions. The major problems with this approach are the potentially difficult-to-read and difficult-to-debug code caused by an overuse of CPP statements. So while it can be done, developers should exerecise some discipline and avoid unnecesarily "smearing" their package implementation details across numerous files.

Package Startup or Boot Sequence

Calls to package routines within the core code timestepping loop can vary. However, all packages should follow a required "boot" sequence outlined here:

```
1. S/R PACKAGES_BOOT()
       :
    CALL OPEN_COPY_DATA_FILE( 'data.pkg', 'PACKAGES_BOOT', ... )


2. S/R PACKAGES_READPARMS()
       :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_READPARMS( retCode )
    #endif

3. S/R PACKAGES_INIT_FIXED()
       :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_INIT_FIXED( retCode )
    #endif

4. S/R PACKAGES_CHECK()
       :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_CHECK( retCode )
    #else
      if ( use${Pkg} )
 &       CALL PACKAGES_CHECK_ERROR('${PKG}')
    #endif

5. S/R PACKAGES_INIT_VARIABLES()
       :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_INIT_VARIA( )
    #endif
```

(continues on next page)

(continued from previous page)

```
6. S/R DO_THE_MODEL_IO

   #ifdef ALLOW_${PKG}
     if ( use${Pkg} )
&       CALL ${PKG}_OUTPUT( )
   #endif

7. S/R PACKAGES_WRITE_PICKUP()

   #ifdef ALLOW_${PKG}
     if ( use${Pkg} )
&       CALL ${PKG}_WRITE_PICKUP( )
   #endif
```

Adding a package to PARAMS.h and packages_boot()

An MITgcm package directory contains all the code needed for that package apart from one variable for each package. This variable is the `use${Pkg}` flag. This flag, which is of type logical, **must** be declared in the shared header file `PARAMS.h` in the `PARM_PACKAGES` block. This convention is used to support a single runtime control file `data.pkg` which is read by the startup routine `packages_boot()` and that sets a flag controlling the runtime use of a package. This routine needs to be able to read the flags for packages that were not built at compile time. Therefore when adding a new package, in addition to creating the per-package directory in the `pkg/` subdirectory a developer should add a `use${Pkg}` flag to `PARAMS.h` and a `use${Pkg}` entry to the `packages_boot() PACKAGES` namelist. The only other package specific code that should appear outside the individual package directory are calls to the specific package API.

8.2 Packages Related to Hydrodynamical Kernel

8.2.1 Generic Advection/Diffusion

The generic_advdiff package contains high-level subroutines to solve the advection-diffusion equation of any tracer, either active (potential temperature, salinity or water vapor) or passive (see *pkg/ptracer*). (see also Section 2.16 and Section 2.17).

8.2.1.1 Introduction

Package "generic_advdiff" provides a common set of routines for calculating advective/diffusive fluxes for tracers (cell centered quantities on a C-grid).

Many different advection schemes are available: the standard centered second order, centered fourth order and upwind biased third order schemes are known as linear methods and require some stable time-stepping method such as Adams-Bashforth. Alternatives such as flux-limited schemes are stable in the forward sense and are best combined with the multi-dimensional method provided in gad_advection.

8.2.1.2 Key subroutines, parameters and files

There are two high-level routines:

- GAD_CALC_RHS calculates all fluxes at time level "n" and is used for the standard linear schemes. This must be used in conjuction with Adams–Bashforth time stepping. Diffusive and parameterized fluxes are always calculated here.
- GAD_ADVECTION calculates just the advective fluxes using the non-linear schemes and can not be used in conjuction with Adams–Bashforth time stepping.

CPP Flag Name	Default	Description
COSINEMETH_III	#define	sets the implementation form of $\cos\varphi$ scaling of bi-harmonic terms for tracer diffusivity (note, in pkg/generic_advdiff routines the definition set here overrides whether this is defined in model/inc/CPP_OPTIONS.h, where the setting affects viscous term calculations)
ISOTROPIC_COS_SCALING	#undef	selects isotropic scaling of harmonic and bi-harmonic terms when using the $\cos\varphi$ scaling (note, in pkg/generic_advdiff routines the definition set here overrides whether this is defined in model/inc/CPP_OPTIONS.h, where the setting affects viscous term calculations)
DISABLE_MULTIDIM_ADVECTION	#undef	disables compilation of multi-dim. advection code
GAD_MULTIDIM_COMPRESSIBLE	#undef	use compressible flow method for multi-dim advection instead of older, less accurate method; note option has no effect on SOM advection which always uses compressible flow method
GAD_ALLOW_TS_SOM_ADV	#undef	enable the use of 2nd-order moment advection scheme (Prather 1986 [Pra86]) for temp. and salinity
GAD_SMOLARKIEWICZ_HACK	#undef	enables hack to get rid of negatives caused by Redi, see Smolarkiewicz (1989) [Smo89] (for ptracers, except temp and salinity)

8.2.1.3 GAD Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
ADVr_TH | 15 |WM      LR      |degC.m^3/s      |Vertical   Advective Flux of Pot.
 ↪Temperature
ADVx_TH | 15 |UU   087MR      |degC.m^3/s      |Zonal      Advective Flux of Pot.
 ↪Temperature
ADVy_TH | 15 |VV   086MR      |degC.m^3/s      |Meridional Advective Flux of Pot.
 ↪Temperature
DFrE_TH | 15 |WM      LR      |degC.m^3/s      |Vertical Diffusive Flux of Pot.
 ↪Temperature (Explicit part)
DIFx_TH | 15 |UU   090MR      |degC.m^3/s      |Zonal      Diffusive Flux of Pot.
 ↪Temperature
DIFy_TH | 15 |VV   089MR      |degC.m^3/s      |Meridional Diffusive Flux of Pot.
 ↪Temperature
DFrI_TH | 15 |WM      LR      |degC.m^3/s      |Vertical Diffusive Flux of Pot.
 ↪Temperature (Implicit part)
ADVr_SLT| 15 |WM      LR      |psu.m^3/s       |Vertical   Advective Flux of Salinity
ADVx_SLT| 15 |UU   094MR      |psu.m^3/s       |Zonal      Advective Flux of Salinity
ADVy_SLT| 15 |VV   093MR      |psu.m^3/s       |Meridional Advective Flux of Salinity
DFrE_SLT| 15 |WM      LR      |psu.m^3/s       |Vertical Diffusive Flux of Salinity
 ↪ (Explicit part)
```

(continues on next page)

(continued from previous page)

```
DIFx_SLT| 15 |UU   097MR      |psu.m^3/s       |Zonal      Diffusive Flux of Salinity
DIFy_SLT| 15 |VV   096MR      |psu.m^3/s       |Meridional Diffusive Flux of Salinity
DFrI_SLT| 15 |WM      LR     |psu.m^3/s       |Vertical Diffusive Flux of Salinity  ␣
↪ (Implicit part)
```

8.2.1.4 Experiments and tutorials that use GAD

- Baroclinic gyre experiment, in tutorial_baroclinic_gyre verification directory.
- Tracer Sensitivity tutorial, in tutorial_tracer_adjsens verification directory.

8.2.2 Momentum Packages

CPP Flag Name	Default	Description
ALLOW_SMAG_3D	#undef	allow isotropic 3D Smagorinsky viscosity (MOM_COMMON_OPTIONS.h)
ALLOW_3D_VISCAH	#undef	allow full 3D specification of horizontal Laplacian viscosity (MOM_COMMON_OPTIONS.h)
ALLOW_3D_VISCA4	#undef	allow full 3D specification of horizontal biharmonic viscosity (MOM_COMMON_OPTIONS.h)
MOM_BOUNDARY_CONSERVE	#undef	conserve u, v momentum next to a step (vertical plane) or a coastline edge (horizontal plane) (MOM_FLUXFORM_OPTIONS.h)

8.2.3 Shapiro Filter

(in directory: pkg/shap_filt/)

8.2.3.1 Key subroutines, parameters and files

Implementation of filter is described in Section 2.18.

8.2.3.2 Experiments and tutorials that use shap filter

- Held Suarez tutorial, in verification/tutorial_held_suarez_cs.
- Other Held Suarez verification experiments (hs94.128x64x5, hs94.1x64x5, hs94.cs-32x32x5)
- AIM verification experiments (aim.5l_cs, aim.5l_Equatorial_Channel, aim.5l_LatLon)
- Fizhi verification experiments (fizhi-cs-32x32x40, fizhi-cs-aqualev20, fizhi-gridalt-hs)

8.2.4 FFT Filtering Code

(in directory: pkg/zonal_filt/)

8.2.4.1 Key subroutines, parameters and files

8.2.4.2 Experiments and tutorials that use zonal filter

- Held Suarez verification experiment (hs94.128x64x5)
- AIM verification experiment (aim.5l_LatLon)

8.2.5 exch2: Extended Cubed Sphere Topology

(in directory: pkg/exch2/)

8.2.5.1 Introduction

The `exch2` package extends the original cubed sphere topology configuration to allow more flexible domain decomposition and parallelization. Cube faces (also called subdomains) may be divided into any number of tiles that divide evenly into the grid point dimensions of the subdomain. Furthermore, the tiles can run on separate processors individually or in groups, which provides for manual compile-time load balancing across a relatively arbitrary number of processors.

The exchange parameters are declared in W2_EXCH_TOPOLOGY.h and assigned in w2_e2setup.F. The validity of the cube topology depends on the `SIZE.h` file as detailed below. The default files provided in the release configure a cubed sphere topology of six tiles, one per subdomain, each with 32×32 grid points, with all tiles running on a single processor. Both files are generated by Matlab scripts in utils/exch2/matlab-topology-generator; see Section 8.2.5.3 for details on creating alternate topologies. Pregenerated examples of these files with alternate topologies are provided under utils/exch2/code-mods along with the appropriate `SIZE.h` file for single-processor execution.

8.2.5.2 Invoking exch2

To use exch2 with the cubed sphere, the following conditions must be met:

- The exch2 package is included when `genmake2` is run. The easiest way to do this is to add the line `exch2` to the `packages.conf` file – see Section *Building the model* for general details.
- An example of `W2_EXCH2_TOPOLOGY.h` and `w2_e2setup.F` must reside in a directory containing files symbolically linked by the `genmake2` script. The safest place to put these is the directory indicated in the `-mods=DIR` command line modifier (typically `../code`), or the build directory. The default versions of these files reside in pkg/exch2 and are linked automatically if no other versions exist elsewhere in the build path, but they should be left untouched to avoid breaking configurations other than the one you intend to modify.
- Files containing grid parameters, named `tile00$n$.mitgrid` where n=(1:6) (one per subdomain), must be in the working directory when the MITgcm executable is run. These files are provided in the example experiments for cubed sphere configurations with 32×32 cube sides – please contact MITgcm support if you want to generate files for other configurations.
- As always when compiling MITgcm, the file `SIZE.h` must be placed where `genmake2` will find it. In particular for exch2, the domain decomposition specified in `SIZE.h` must correspond with the particular configuration's topology specified in `W2_EXCH2_TOPOLOGY.h` and `w2_e2setup.F`. Domain decomposition issues particular to exch2 are addressed in Section *Generating Topology Files for exch2* and *exch2, SIZE.h, and Multiprocessing* a more general background on the subject relevant to MITgcm is presented in Section *Using the WRAPPER*.

At the time of this writing the following examples use exch2 and may be used for guidance:

- verification/adjust_nlfs.cs-32x32x1

- verification/adjustment.cs-32x32x1
- verification/aim.5l_cs
- verification/global_ocean.cs32x15
- verification/hs94.cs-32x32x5

8.2.5.3 Generating Topology Files for exch2

Alternate cubed sphere topologies may be created using the Matlab scripts in utils/exch2/matlab-topology-generator. Running the m-file driver.m from the Matlab prompt (there are no parameters to pass) generates exch2 topology files `W2_EXCH2_TOPOLOGY.h` and `w2_e2setup.F` in the working directory and displays a figure of the topology via Matlab – Figure 8.4, Figure 8.3, and Figure 8.2 are examples of the generated diagrams. The other m-files in the directory are subroutines called from driver.m and should not be run ''bare" except for development purposes.

The parameters that determine the dimensions and topology of the generated configuration are `nr`, `nb`, `ng`, `tnx` and `tny`, and all are assigned early in the script.

The first three determine the height and width of the subdomains and hence the size of the overall domain. Each one determines the number of grid points, and therefore the resolution, along the subdomain sides in a ''great circle" around each the three spatial axes of the cube. At the time of this writing MITgcm requires these three parameters to be equal, but they provide for future releases to accomodate different resolutions around the axes to allow subdomains with differing resolutions.

The parameters `tnx` and `tny` determine the width and height of the tiles into which the subdomains are decomposed, and must evenly divide the integer assigned to `nr`, `nb` and `ng`. The result is a rectangular tiling of the subdomain. Figure 8.2 shows one possible topology for a twenty-four-tile cube, and Figure 8.4 shows one for six tiles.

Tiles can be selected from the topology to be omitted from being allocated memory and processors. This tuning is useful in ocean modeling for omitting tiles that fall entirely on land. The tiles omitted are specified in the file blanklist.txt by their tile number in the topology, separated by a newline.

8.2.5.4 exch2, SIZE.h, and Multiprocessing

Once the topology configuration files are created, each Fortran `PARAMETER` in `SIZE.h` must be configured to match. Section 6.3 povides a general description of domain decomposition within MITgcm and its relation to `SIZE.h`. The current section specifies constraints that the exch2 package imposes and describes how to enable parallel execution with MPI.

As in the general case, the parameters sNx and sNy define the size of the individual tiles, and so must be assigned the same respective values as `tnx` and `tny` in driver.m.

The halo width parameters OLx and OLy have no special bearing on exch2 and may be assigned as in the general case. The same holds for Nr, the number of vertical levels in the model.

The parameters nSx, nSy, nPx, and nPy relate to the number of tiles and how they are distributed on processors. When using exch2, the tiles are stored in the `x` dimension, and so nSy =1 in all cases. Since the tiles as configured by exch2 cannot be split up accross processors without regenerating the topology, nPy = 1 as well.

The number of tiles MITgcm allocates and how they are distributed between processors depends on nPx and nSx. nSx is the number of tiles per processor and nPx is the number of processors. The total number of tiles in the topology minus those listed in `blanklist.txt` must equal `nSx*nPx`. Note that in order to obtain maximum usage from a given number of processors in some cases, this restriction might entail sharing a processor with a tile that would otherwise be excluded because it is topographically outside of the domain and therefore in `blanklist.txt`. For example, suppose you have five processors and a domain decomposition of thirty-six tiles that allows you to exclude

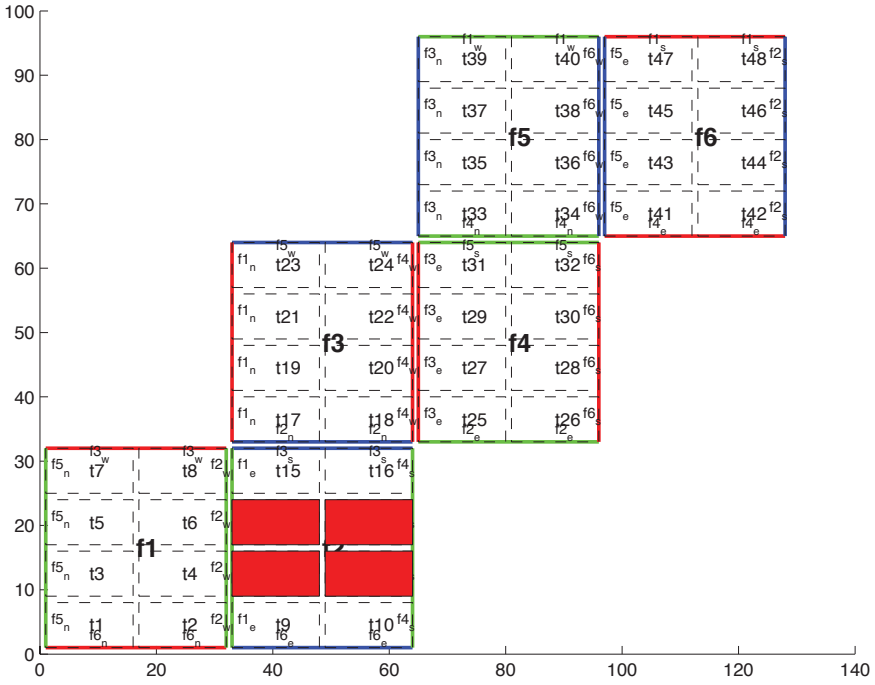

Figure 8.2: Plot of a cubed sphere topology with a 32 × 192 domain divided into six 32 × 32 subdomains, each of which is divided into eight tiles of width `tnx=16` and height `tny=8` for a total of forty-eight tiles. The colored borders of the subdomains represent the parameters `nr` (red), `ng` (green), and `nb` (blue). This tiling is used in the example verification/adjustment.cs-32x32x1/ with the option (blanklist.txt) to remove the land-only 4 tiles (11,12,13,14) which are filled in red on the plot.

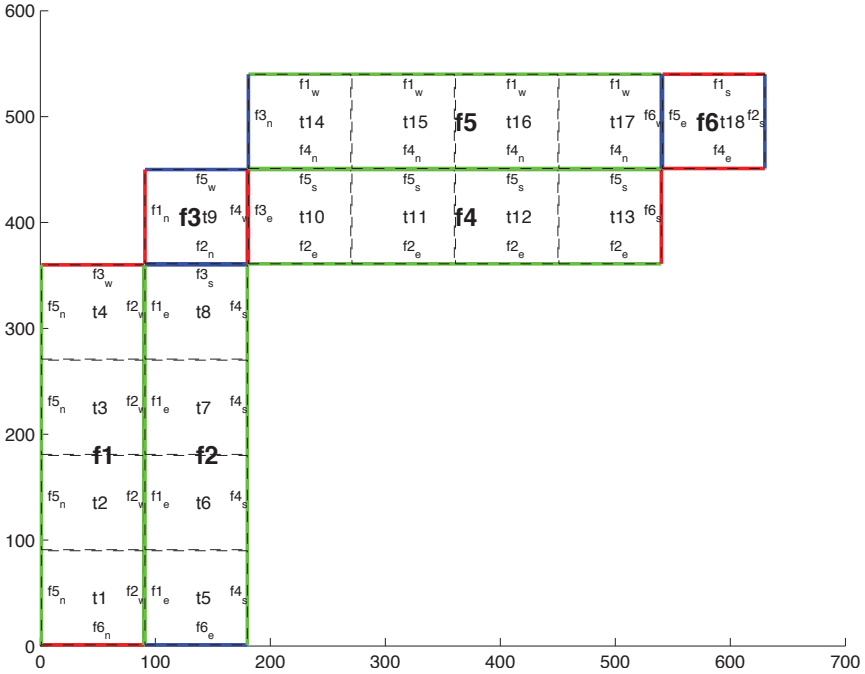

Figure 8.3: Plot of a non-square cubed sphere topology with 6 subdomains of different size (nr=90,ng=360,nb=90), divided into one to four tiles each (`tnx=90, tny=90`), resulting in a total of 18 tiles.

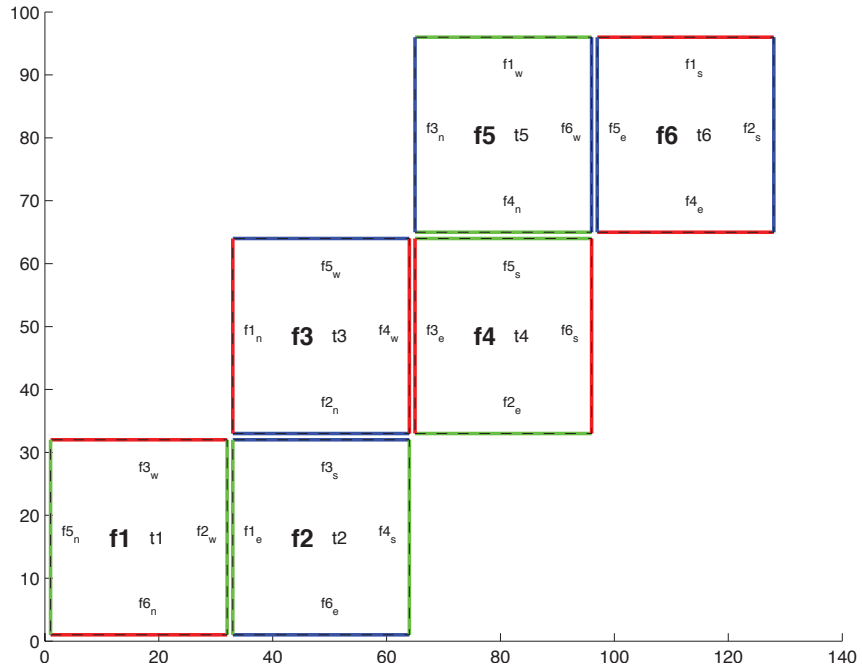

Figure 8.4: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains with one tile each (`tnx=32, tny=32`). This is the default configuration.

seven tiles. To evenly distribute the remaining twenty-nine tiles among five processors, you would have to run one ‘’dummy” tile to make an even six tiles per processor. Such dummy tiles are *not* listed in `blanklist.txt`.

The following is an example of `SIZE.h` for the six-tile configuration illustrated in Figure 8.4 running on one processor:

```
PARAMETER (
&            sNx =  32,
&            sNy =  32,
&            OLx =   2,
&            OLy =   2,
&            nSx =   6,
&            nSy =   1,
&            nPx =   1,
&            nPy =   1,
&            Nx   = sNx*nSx*nPx,
&            Ny   = sNy*nSy*nPy,
&            Nr   =   5)
```

The following is an example for the forty-eight-tile topology in Figure 8.2 running on six processors:

```
PARAMETER (
&            sNx =  16,
&            sNy =   8,
&            OLx =   2,
&            OLy =   2,
&            nSx =   8,
&            nSy =   1,
&            nPx =   6,
```

(continues on next page)

(continued from previous page)

```
&           nPy =   1,
&           Nx  = sNx*nSx*nPx,
&           Ny  = sNy*nSy*nPy,
&           Nr  =   5)
```

8.2.5.5 Key Variables

The descriptions of the variables are divided up into scalars, one-dimensional arrays indexed to the tile number, and two and three-dimensional arrays indexed to tile number and neighboring tile. This division reflects the functionality of these variables: The scalars are common to every part of the topology, the tile-indexed arrays to individual tiles, and the arrays indexed by tile and neighbor to relationships between tiles and their neighbors.

Scalars:

The number of tiles in a particular topology is set with the parameter exch2_nTiles, and the maximum number of neighbors of any tiles by W2_maxNeighbours. These parameters are used for defining the size of the various one and two dimensional arrays that store tile parameters indexed to the tile number and are assigned in the files generated by driver.m.

The scalar parameters `exch2_domain_nxt` and `exch2_domain_nyt` express the number of tiles in the `x` and `y` global indices. For example, the default setup of six tiles (Figure 8.4) has `exch2_domain_nxt=6` and `exch2_domain_nyt=1`. A topology of forty-eight tiles, eight per subdomain (as in Figure 8.2), will have `exch2_domain_nxt=12` and `exch2_domain_nyt=4`. Note that these parameters express the tile layout in order to allow global data files that are tile-layout-neutral. They have no bearing on the internal storage of the arrays. The tiles are stored internally in a range from `bi = (1:exch2_nTiles)` in the `x` axis, and the `y` axis variable `bj` is assumed to equal 1 throughout the package.

Arrays indexed to tile number:

The following arrays are of length `exch2_nTiles` and are indexed to the tile number, which is indicated in the diagrams with the notation `tn`. The indices are omitted in the descriptions.

The arrays exch2_tnx and exch2_tny express the `x` and `y` dimensions of each tile. At present for each tile `exch2_tnx`=``sNx`` and `exch2_tny` = `sNy`, as assigned in `SIZE.h` and described in Section 8.2.5.4. Future releases of MITgcm may allow varying tile sizes.

The arrays exch2_tbasex and exch2_tbasey determine the tiles' Cartesian origin within a subdomain and locate the edges of different tiles relative to each other. As an example, in the default six-tile topology (Figure 8.4) each index in these arrays is set to 0 since a tile occupies its entire subdomain. The twenty-four-tile case discussed above will have values of 0 or 16, depending on the quadrant of the tile within the subdomain. The elements of the arrays exch2_txglobalo and exch2_txglobalo are similar to exch2_tbasex and exch2_tbasey, but locate the tile edges within the global address space, similar to that used by global output and input files.

The array exch2_myFace contains the number of the subdomain of each tile, in a range `(1:6)` in the case of the standard cube topology and indicated by **fn** in Figure 8.4 and Figure 8.2. exch2_nNeighbours contains a count of the neighboring tiles each tile has, and sets the bounds for looping over neighboring tiles. exch2_tProc holds the process rank of each tile, and is used in interprocess communication.

The arrays exch2_isWedge, exch2_isEedge, exch2_isSedge, and exch2_isNedge are set to 1 if the indexed tile lies on the edge of its subdomain, 0 if not. The values are used within the topology generator to determine the orientation of neighboring tiles, and to indicate whether a tile lies on the corner of a subdomain. The latter case requires special exchange and numerical handling for the singularities at the eight corners of the cube.

Arrays Indexed to Tile Number and Neighbor:

The following arrays have vectors of length W2_maxNeighbours and exch2_nTiles and describe the orientations between the the tiles.

The array `exch2_neighbourId(a,T)` holds the tile number `Tn` for each of the tile number `T`'s neighboring tiles `a`. The neighbor tiles are indexed `1:exch2_nNeighbours(T)` in the order right to left on the north then south edges, and then top to bottom on the east then west edges.

The `exch2_opposingSend_record(a,T)` array holds the index `b` of the element in `exch2_neighbourId(b,Tn)` that holds the tile number `T`, given `Tn=exch2_neighborId(a,T)`. In other words,

```
exch2_neighbourId( exch2_opposingSend_record(a,T),
                   exch2_neighbourId(a,T) ) = T
```

This provides a back-reference from the neighbor tiles.

The arrays exch2_pi and exch2_pj specify the transformations of indices in exchanges between the neighboring tiles. These transformations are necessary in exchanges between subdomains because a horizontal dimension in one subdomain may map to other horizonal dimension in an adjacent subdomain, and may also have its indexing reversed. This swapping arises from the ''folding" of two-dimensional arrays into a three-dimensional cube.

The dimensions of `exch2_pi(t,N,T)` and `exch2_pj(t,N,T)` are the neighbor ID `N` and the tile number `T` as explained above, plus a vector of length 2 containing transformation factors `t`. The first element of the transformation vector holds the factor to multiply the index in the same dimension, and the second element holds the the same for the orthogonal dimension. To clarify, `exch2_pi(1,N,T)` holds the mapping of the `x` axis index of tile `T` to the `x` axis of tile `T`'s neighbor `N`, and `exch2_pi(2,N,T)` holds the mapping of `T`'s `x` index to the neighbor `N`'s `y` index.

One of the two elements of `exch2_pi` or `exch2_pj` for a given tile `T` and neighbor `N` will be 0, reflecting the fact that the two axes are orthogonal. The other element will be 1 or -1, depending on whether the axes are indexed in the same or opposite directions. For example, the transform vector of the arrays for all tile neighbors on the same subdomain will be `(1,0)`, since all tiles on the same subdomain are oriented identically. An axis that corresponds to the orthogonal dimension with the same index direction in a particular tile-neighbor orientation will have `(0,1)`. Those with the opposite index direction will have `(0,-1)` in order to reverse the ordering.

The arrays exch2_oi, exch2_oj, exch2_oi_f, and exch2_oj_f are indexed to tile number and neighbor and specify the relative offset within the subdomain of the array index of a variable going from a neighboring tile `N` to a local tile `T`. Consider `T=1` in the six-tile topology (Figure 8.4), where

```
exch2_oi(1,1)=33
exch2_oi(2,1)=0
exch2_oi(3,1)=32
exch2_oi(4,1)=-32
```

The simplest case is `exch2_oi(2,1)`, the southern neighbor, which is `Tn=6`. The axes of `T` and `Tn` have the same orientation and their `x` axes have the same origin, and so an exchange between the two requires no changes to the `x` index. For the western neighbor (`Tn=5`), `code_oi(3,1)=32` since the `x=0` vector on `T` corresponds to the `y=32` vector on `Tn`. The eastern edge of `T` shows the reverse case (`exch2_oi(4,1)=-32`), where `x=32` on `T` exchanges with `x=0` on `Tn=2`.

The most interesting case, where `exch2_oi(1,1)=33` and `Tn=3`, involves a reversal of indices. As in every case, the offset `exch2_oi` is added to the original `x` index of `T` multiplied by the transformation factor `exch2_pi(t,N,T)`. Here `exch2_pi(1,1,1)=0` since the `x` axis of `T` is orthogonal to the `x` axis of `Tn`. `exch2_pi(2,1,1)=-1` since the `x` axis of `T` corresponds to the `y` axis of `Tn`, but the index is reversed. The result is that the index of the northern edge of `T`, which runs `(1:32)`, is transformed to `(-1:-32)`. `exch2_oi(1,1)` is then added to this range to get back `(32:1)` – the index of the `y` axis of `Tn` relative to `T`. This transformation may seem overly convoluted for the six-tile case, but it is necessary to provide a general solution for various topologies.

Finally, exch2_itlo_c, exch2_ithi_c, exch2_jtlo_c and exch2_jthi_c hold the location and index bounds of the edge segment of the neighbor tile N's subdomain that gets exchanged with the local tile T. To take the example of tile T=2 in the forty-eight-tile topology (Figure 8.2):

```
exch2_itlo_c(4,2)=17
exch2_ithi_c(4,2)=17
exch2_jtlo_c(4,2)=0
exch2_jthi_c(4,2)=33
```

Here N=4, indicating the western neighbor, which is Tn=1. Tn resides on the same subdomain as T, so the tiles have the same orientation and the same x and y axes. The x axis is orthogonal to the western edge and the tile is 16 points wide, so exch2_itlo_c and exch2_ithi_c indicate the column beyond Tn's eastern edge, in that tile's halo region. Since the border of the tiles extends through the entire height of the subdomain, the y axis bounds exch2_jtlo_c to exch2_jthi_c cover the height of (1:32), plus 1 in either direction to cover part of the halo.

For the north edge of the same tile T=2 where N=1 and the neighbor tile is Tn=5:

```
exch2_itlo_c(1,2)=0
exch2_ithi_c(1,2)=0
exch2_jtlo_c(1,2)=0
exch2_jthi_c(1,2)=17
```

T's northern edge is parallel to the x axis, but since Tn's y axis corresponds to T's x axis, T's northern edge exchanges with Tn's western edge. The western edge of the tiles corresponds to the lower bound of the x axis, so exch2_itlo_c and exch2_ithi_c are 0, in the western halo region of Tn. The range of exch2_jtlo_c and exch2_jthi_c correspond to the width of T's northern edge, expanded by one into the halo.

8.2.5.6 Key Routines

Most of the subroutines particular to exch2 handle the exchanges themselves and are of the same format as those described in *Cube sphere communication*. Like the original routines, they are written as templates which the local Makefile converts from RX into RL and RS forms.

The interfaces with the core model subroutines are EXCH_UV_XY_RX, EXCH_UV_XYZ_RX and EXCH_XY_RX. They override the standard exchange routines when genmake2 is run with exch2 option. They in turn call the local exch2 subroutines EXCH2_UV_XY_RX and EXCH2_UV_XYZ_RX for two and three-dimensional vector quantities, and EXCH2_XY_RX and EXCH2_XYZ_RX for two and three-dimensional scalar quantities. These subroutines set the dimensions of the area to be exchanged, call EXCH2_RX1_CUBE for scalars and EXCH2_RX2_CUBE for vectors, and then handle the singularities at the cube corners.

The separate scalar and vector forms of EXCH2_RX1_CUBE and EXCH2_RX2_CUBE reflect that the vector-handling subroutine needs to pass both the u and v components of the physical vectors. This swapping arises from the topological folding discussed above, where the x and y axes get swapped in some cases, and is not an issue with the scalar case. These subroutines call EXCH2_SEND_RX1 and EXCH2_SEND_RX2, which do most of the work using the variables discussed above.

8.2.5.7 Experiments and tutorials that use exch2

- Held Suarez tutorial, in verification/tutorial_held_suarez_cs verification directory.

8.2.6 Gridalt - Alternate Grid Package

8.2.6.1 Introduction

The gridalt package [Mol09] is designed to allow different components of MITgcm to be run using horizontal and/or vertical grids which are different from the main model grid. The gridalt routines handle the definition of the all the various alternative grid(s) and the mappings between them and the MITgcm grid. The implementation of the gridalt package which allows the high end atmospheric physics (fizhi) to be run on a high resolution and quasi terrain-following vertical grid is documented here. The package has also (with some user modifications) been used for other calculations within the GCM.

The rationale for implementing the atmospheric physics on a high resolution vertical grid involves the fact that the MITgcm p^* (or any pressure-type) coordinate cannot maintain the vertical resolution near the surface as the bottom topography rises above sea level. The vertical length scales near the ground are small and can vary on small time scales, and the vertical grid must be adequate to resolve them. Many studies with both regional and global atmospheric models have demonstrated the improvements in the simulations when the vertical resolution near the surface is increased (). Some of the benefit of increased resolution near the surface is realized by employing the higher resolution for the computation of the forcing due to turbulent and convective processes in the atmosphere.

The parameterizations of atmospheric subgrid scale processes are all essentially one-dimensional in nature, and the computation of the terms in the equations of motion due to these processes can be performed for the air column over one grid point at a time. The vertical grid on which these computations take place can therefore be entirely independant of the grid on which the equations of motion are integrated, and the 'tendency' terms can be interpolated to the vertical grid on which the equations of motion are integrated. A modified p^* coordinate, which adjusts to the local terrain and adds additional levels between the lower levels of the existing p^* grid (and perhaps between the levels near the tropopause as well), is implemented. The vertical discretization is different for each grid point, although it consist of the same number of levels. Additional 'sponge' levels aloft are added when needed. The levels of the physics grid are constrained to fit exactly into the existing p^* grid, simplifying the mapping between the two vertical coordinates. This is illustrated as follows:

The algorithm presented here retains the state variables on the high resolution 'physics' grid as well as on the coarser resolution 'dynamics' grid, and ensures that the two estimates of the state 'agree' on the coarse resolution grid. It would have been possible to implement a technique in which the tendencies due to atmospheric physics are computed on the high resolution grid and the state variables are retained at low resolution only. This, however, for the case of the turbulence parameterization, would mean that the turbulent kinetic energy source terms, and all the turbulence terms that are written in terms of gradients of the mean flow, cannot really be computed making use of the fine structure in the vertical.

8.2.6.2 Equations on Both Grids

In addition to computing the physical forcing terms of the momentum, thermodynamic and humidity equations on the modified (higher resolution) grid, the higher resolution structure of the atmosphere (the boundary layer) is retained between physics calculations. This neccessitates a second set of evolution equations for the atmospheric state variables on the modified grid. If the equation for the evolution of U on p^* can be expressed as:

$$\left.\frac{\partial U}{\partial t}\right|_{p^*}^{total} = \left.\frac{\partial U}{\partial t}\right|_{p^*}^{dynamics} + \left.\frac{\partial U}{\partial t}\right|_{p^*}^{physics}$$

where the physics forcing terms on p^* have been mapped from the modified grid, then an additional equation to govern the evolution of U (for example) on the modified grid is written:

$$\left.\frac{\partial U}{\partial t}\right|_{p^{*m}}^{total} = \left.\frac{\partial U}{\partial t}\right|_{p^{*m}}^{dynamics} + \left.\frac{\partial U}{\partial t}\right|_{p^{*m}}^{physics} + \gamma(U|_{p^*} - U|_{p^{*m}})$$

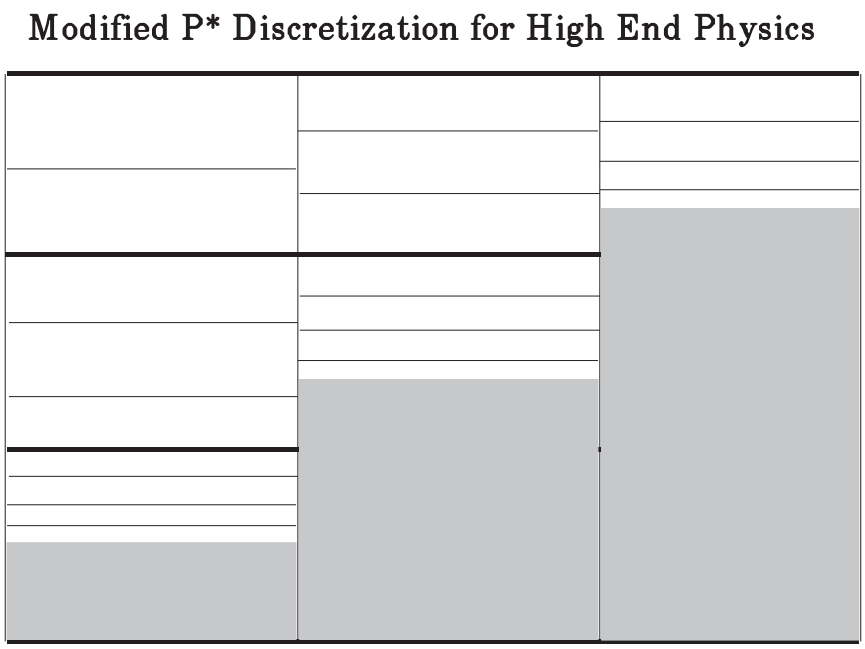

Figure 8.5: Vertical discretization for MITgcm (dark grey lines) and for the atmospheric physics (light grey lines). In this implementation, all MITgcm level interfaces must coincide with atmospheric physics level interfaces.

where p^{*m} refers to the modified higher resolution grid, and the dynamics forcing terms have been mapped from p^* space. The last term on the RHS is a relaxation term, meant to constrain the state variables on the modified vertical grid to 'track' the state variables on the p^* grid on some time scale, governed by γ. In the present implementation, $\gamma = 1$, requiring an immediate agreement between the two 'states'.

8.2.6.3 Time stepping Sequence

If we write T_{phys} as the temperature (or any other state variable) on the high resolution physics grid, and T_{dyn} as the temperature on the coarse vertical resolution dynamics grid, then:

1. Compute the tendency due to physics processes.
2. Advance the physics state: ${T^{n+1}}^{**}{}_{phys}(l) = {T^n}_{phys}(l) + \delta T_{phys}$.
3. Interpolate the physics tendency to the dynamics grid, and advance the dynamics state by physics and dynamics tendencies: ${T^{n+1}}_{dyn}(L) = {T^n}_{dyn}(L) + \delta T_{dyn}(L) + [\delta T_{phys}(l)](L)$.
4. Interpolate the dynamics tendency to the physics grid, and update the physics grid due to dynamics tendencies: ${T^{n+1^*}}_{phys}(l) = {T^{n+1^{**}}}_{phys}(l) + \delta T_{dyn}(L)(l)$.
5. Apply correction term to physics state to account for divergence from dynamics state: ${T^{n+1}}_{phys}(l) = {T^{n+1^*}}_{phys}(l) + \gamma\{T_{dyn}(L) - [T_{phys}(l)](L)\}(l)$. Where $\gamma = 1$ here.

8.2.6.4 Interpolation

In order to minimize the correction terms for the state variables on the alternative, higher resolution grid, the vertical interpolation scheme must be constructed so that a dynamics-to-physics interpolation can be exactly reversed with a physics-to-dynamics mapping. The simple scheme employed to achieve this is:

Coarse to fine: For all physics layers l in dynamics layer L, $T_{phys}(l) = \{T_{dyn}(L)\} = T_{dyn}(L)$.

Fine to coarse: For all physics layers l in dynamics layer L, $T_{dyn}(L) = [T_{phys}(l)] = \int T_{phys} dp$.

Where $\{\}$ is defined as the dynamics-to-physics operator and [] is the physics-to-dynamics operator, T stands for any state variable, and the subscripts $phys$ and dyn stand for variables on the physics and dynamics grids, respectively.

8.2.6.5 Key subroutines, parameters and files

One of the central elements of the gridalt package is the routine which is called from subroutine gridalt_initialise to define the grid to be used for the high end physics calculations. Routine make_phys_grid passes back the parameters which define the grid, ultimately stored in the common block gridalt_mapping.

```
       subroutine make_phys_grid(drF,hfacC,im1,im2,jm1,jm2,Nr,
     . Nsx,Nsy,i1,i2,j1,j2,bi,bj,Nrphys,Lbot,dpphys,numlevphys,nlperdyn)
c**********************************************************************
c Purpose: Define the grid that the will be used to run the high-end
c          atmospheric physics.
c
c Algorithm: Fit additional levels of some (~) known thickness in
c          between existing levels of the grid used for the dynamics
c
c Need:    Information about the dynamics grid vertical spacing
c
c Input:   drF         - delta r (p*) edge-to-edge
c          hfacC       - fraction of grid box above topography
c          im1, im2    - beginning and ending i - dimensions
c          jm1, jm2    - beginning and ending j - dimensions
c          Nr          - number of levels in dynamics grid
c          Nsx,Nsy     - number of processes in x and y direction
c          i1, i2      - beginning and ending i - index to fill
c          j1, j2      - beginning and ending j - index to fill
c          bi, bj      - x-dir and y-dir index of process
c          Nrphys      - number of levels in physics grid
c
c Output:  dpphys      - delta r (p*) edge-to-edge of physics grid
c          numlevphys  - number of levels used in the physics
c          nlperdyn    - physics level number atop each dynamics layer
c
c NOTES: 1) Pressure levs are built up from bottom, using p0, ps and dp:
c              p(i,j,k)=p(i,j,k-1) + dp(k)*ps(i,j)/p0(i,j)
c        2) Output dp's are aligned to fit EXACTLY between existing
c           levels of the dynamics vertical grid
c        3) IMPORTANT! This routine assumes the levels are numbered
c           from the bottom up, ie, level 1 is the surface.
c           IT WILL NOT WORK OTHERWISE!!!
c        4) This routine does NOT work for surface pressures less
c           (ie, above in the atmosphere) than about 350 mb
c**********************************************************************
```

In the case of the grid used to compute the atmospheric physical forcing (*Fizhi: High-end Atmospheric Physics*), the locations of the grid points move in time with the MITgcm p^* coordinate, and subroutine gridalt_update is called during the run to update the locations of the grid points:

```
       subroutine gridalt_update(myThid)
c**********************************************************************
c Purpose: Update the pressure thicknesses of the layers of the
```

(continues on next page)

(continued from previous page)

```
c          alternative vertical grid (used now for atmospheric physics).
c
c Calculate: dpphys    - new delta r (p*) edge-to-edge of physics grid
c                        using dpphys0 (initial value) and rstarfacC
c*********************************************************************
```

The gridalt package also supplies utility routines which perform the mappings from one grid to the other. These routines are called from the code which computes the fields on the alternative (fizhi) grid.

```
      subroutine dyn2phys(qdyn,pedyn,im1,im2,jm1,jm2,lmdyn,Nsx,Nsy,
     . idim1,idim2,jdim1,jdim2,bi,bj,windphy,pephy,Lbot,lmphy,nlperdyn,
     . flg,qphy)
C*********************************************************************
C Purpose:
C   To interpolate an arbitrary quantity from the 'dynamics' eta (pstar)
C               grid to the higher resolution physics grid
C Algorithm:
C   Routine works one layer (edge to edge pressure) at a time.
C   Dynamics -> Physics retains the dynamics layer mean value,
C   weights the field either with the profile of the physics grid
C   wind speed (for U and V fields), or uniformly (T and Q)
C
C Input:
C   qdyn..... [im,jm,lmdyn] Arbitrary Quantity on Input Grid
C   pedyn.... [im,jm,lmdyn+1] Pressures at bottom edges of input levels
C   im1,2 ... Limits for Longitude Dimension of Input
C   jm1,2 ... Limits for Latitude  Dimension of Input
C   lmdyn.... Vertical  Dimension of Input
C   Nsx...... Number of processes in x-direction
C   Nsy...... Number of processes in y-direction
C   idim1,2.. Beginning and ending i-values to calculate
C   jdim1,2.. Beginning and ending j-values to calculate
C   bi....... Index of process number in x-direction
C   bj....... Index of process number in x-direction
C   windphy.. [im,jm,lmphy] Magnitude of the wind on the output levels
C   pephy.... [im,jm,lmphy+1] Pressures at bottom edges of output levels
C   lmphy.... Vertical  Dimension of Output
C   nlperdyn. [im,jm,lmdyn] Highest Physics level in each dynamics level
C   flg...... Flag to indicate field type (0 for T or Q, 1 for U or V)
C
C Output:
C   qphy..... [im,jm,lmphy] Quantity at output grid (physics grid)
C
C Notes:
C   1) This algorithm assumes that the output (physics) grid levels
C      fit exactly into the input (dynamics) grid levels
C*********************************************************************
```

And similarly, gridalt contains subroutine phys2dyn.

8.2.6.6 Gridalt Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------------------
DPPHYS  | 20 |SM      ML      |Pascal          |Pressure Thickness of Layers on Fizhi␣
↪Grid
```

8.2.6.7 Dos and donts

8.2.6.8 Gridalt Reference

8.2.6.9 Experiments and tutorials that use gridalt

- Fizhi experiment, in verification/fizhi-cs-32x32x10 verification directory

8.3 General purpose numerical infrastructure packages

8.3.1 OBCS: Open boundary conditions for regional modeling

Authors: Alistair Adcroft, Patrick Heimbach, Samar Katiwala, Martin Losch

8.3.1.1 Introduction

The OBCS-package is fundamental to regional ocean modeling with the MITgcm, but there are so many details to be considered in regional ocean modeling that this package cannot accommodate all imaginable and possible options. Therefore, for a regional simulation with very particular details it is recommended to familiarize oneself not only with the compile- and runtime-options of this package, but also with the code itself. In many cases it will be necessary to adapt the obcs-code (in particular S/R OBCS_CALC) to the application in question; in these cases the obcs-package (together with the rbcs-package Section 8.3.2) is a very useful infrastructure for implementing special regional models.

8.3.1.2 OBCS configuration and compiling

As with all MITgcm packages OBCS can be turned on or off at compile time

- using the `packages.conf` file by adding `obcs` to it
- or using `genmake2` adding `-enable=obcs` or `-disable=obcs` switches
- *Required packages and CPP options:*
  - Two alternatives are available for prescribing open boundary values, which differ in the way how OB's are treated in time:
    - Simple time-management (e.g., constant in time, or cyclic with fixed frequency) is provided through S/R `obcs_external_fields_load`.
    - More sophisticated 'real-time' (i.e. calendar time) management is available through `obcs_prescribe_read`.
  - The latter case requires packages `cal` and `exf` to be enabled.

(see also Section 3.5).

Parts of the OBCS code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in OBCS_OPTIONS.h. Table 8.3.1.2 summarizes these options.

CPP option	Default	Description
ALLOW_OBCS_NORTH	#define	enable Northern OB
ALLOW_OBCS_SOUTH	#define	enable Southern OB
ALLOW_OBCS_EAST	#define	enable Eastern OB
ALLOW_OBCS_WEST	#define	enable Western OB
ALLOW_OBCS_PRESCRIBE	#define	enable code for prescribing OB's
ALLOW_OBCS_SPONGE	#undef	enable sponge layer code
ALLOW_OBCS_BALANCE	#define	enable code for balancing transports through OB's
ALLOW_ORLANSKI	#define	enable Orlanski radiation conditions at OB's
ALLOW_OBCS_STEVENS	#undef	enable Stevens (1990) boundary conditions at OB's (currently only implemented for eastern and western boundaries and NOT for ptracers)
ALLOW_OBCS_SEAICE_SPONGE	#undef	Include hooks to sponge layer treatment of pkg/seaice variables
ALLOW_OBCS_TIDES	#undef	Add tidal contributions to normal OB flow (At the moment tidal forcing is applied only to "normal" flow)

8.3.1.3 Run-time parameters

Run-time parameters are set in files `data.pkg` `data.obcs` and `data.exf` if 'real-time' prescription is requested (i.e., package *exf* enabled). These parameter files are read in S/R packages_readparms.F obcs_readparms.F and exf_readparms.F respectively. Run-time parameters may be broken into 3 categories:

1. switching on/off the package at runtime
2. OBCS package flags and parameters
3. additional timing flags in `data.exf` if selected.

Enabling the package

The OBCS package is switched on at runtime by setting useOBCS = `.TRUE.` in `data.pkg`.

Package flags and parameters

Table 8.3.1.3 summarizes the runtime flags that are set in `data.obcs` and their default values.

Flag/parameter	default	Description
OB_Jnorth	0	Nx-vector of J-indices (w.r.t. Ny) of Northern OB at each I-position (w.r.t. Nx)
OB_Jsouth	0	Nx-vector of J-indices (w.r.t. Ny) of Southern OB at each I-position (w.r.t. Nx)
OB_Ieast	0	Ny-vector of I-indices (w.r.t. Nx) of Eastern OB at each J-position (w.r.t. Ny)
OB_Iwest	0	Ny-vector of I-indices (w.r.t. Nx) of Western OB at each J-position (w.r.t. Ny)
useOBCSprescribe	FALSE	
useOBCSsponge	FALSE	
useOBCSbalance	FALSE	
OBCS_balanceFacN, OBCS_balanceFacS, OBCS_balanceFacE, OBCS_balanceFacW	1	Factor(s) determining the details of the balancing code
OBCSbalanceSurf	FALSE	include surface mass flux in balance
useOrlanskiNorth, useOrlanskiSouth, useOrlanskiEast, useOrlanskiWest	FALSE	Turn on Orlanski boundary conditions for individual boundary.
useStevensNorth, useStevensSouth, useStevensEast, useStevensWest	FALSE	Turn on Stevens boundary conditions for individual boundary
OB**Xy**File	' '	File name of OB field: **X**: **N**(orth), **S**(outh), **E**(ast), **W**(est) **y**: **t**(emperature), **s**(alinity), **eta** (sea surface height), **u**(-velocity), **v**(-velocity), **w**(-velocity), **a** (seaice area), **h** (sea ice thickness), **sn** (snow thickness), **sl** (sea ice salinity)
Orlanski Parameters	*OBCS_PARM02*	
cvelTimeScale	2000.0	Averaging period for phase speed (seconds)
CMAX	0.45	Maximum allowable phase speed-CFL for AB-II (m/s)
CFIX	0.8	Fixed boundary phase speed (m/s)
useFixedCEast	FALSE	
useFixedCWest	FALSE	
Sponge layer parameters	*OBCS_PARM03*	
spongeThickness	0	sponge layer thickness (in grid points)
Urelaxobcsinner	0.0	relaxation time scale at the innermost sponge layer point of a meridional OB (s)
Vrelaxobcsinner	0.0	relaxation time scale at the innermost sponge layer point of a zonal OB (s)
Urelaxobcsbound	0.0	relaxation time scale at the outermost sponge layer point of a meridional OB (s)
Vrelaxobcsbound	0.0	relaxation time scale at the outermost sponge layer point of a zonal OB (s)
Stevens parameters	*OBCS_PARM04*	
TrelaxStevens SrelaxStevens	0	Relaxation time scale for temperature/salinity (s)
useStevensPhaseVel	TRUE	
useStevensAdvection	TRUE	

8.3.1.4 Defining open boundary positions

There are four open boundaries (OBs): Northern, Southern, Eastern, and Western. All OB locations are specified by their absolute meridional (Northern/Southern) or zonal (Eastern/Western) indices. Thus, for each zonal position $i = 1 \ldots N_x$ a meridional index j specifies the Northern/Southern OB position, and for each meridional position $j = 1 \ldots N_y$ a zonal index i specifies the Eastern/Western OB position. For Northern/Southern OB this defines an N_x-dimensional "row" array `OB_Jnorth(Nx)` / `OB_Jsouth(Nx)` and an N_y-dimenisonal "column" array `OB_Ieast(Ny)` / `OB_Iwest(Ny)`. Positions determined in this way allows Northern/Southern OBs to be at variable j (or y) positions and Eastern/Western OBs at variable i (or x) positions. Here indices refer to tracer points on the C-grid. A zero (0) element in `OB_I...` `OB_J...` means there is no corresponding OB in that column/row. For a Northern/Southern OB, the OB V point is to the South/North. For an Eastern/Western OB, the OB U point is to the West/East. For example

`OB_Jnorth(3)=34` means that:

- `T(3,34)` is a an OB point
- `U(3,34)` is a an OB point
- `V(3,34)` is a an OB point

`OB_Jsouth(3)=1` means that:

- `T(3,1)` is a an OB point
- `U(3,1)` is a an OB point
- `V(3,2)` is a an OB point

`OB_Ieast(10)=69` means that:

- `T(69,10)` is a an OB point
- `U(69,10)` is a an OB point
- `V(69,10)` is a an OB point

`OB_Iwest(10)=1` means that:

- `T(1,10)` is a an OB point
- `U(2,10)` is a an OB point
- `V(1,10)` is a an OB point

For convenience, negative values for `Jnorth`/`Ieast` refer to points relative to the Northern/Eastern edges of the model eg. `OB_Jnorth(3)` $= -1$ means that the point $(3, \texttt{Ny})$ is a northern OB.

Simple examples: For a model grid with $N_x \times N_y = 120 \times 144$ horizontal grid points with four open boundaries along the four edges of the domain the simplest way of specifying the boundary points in is:

```
  OB_Ieast = 144*-1,
# or OB_Ieast = 144*120,
  OB_Iwest = 144*1,
  OB_Jnorth = 120*-1,
# or OB_Jnorth = 120*144,
  OB_Jsouth = 120*1,
```

If only the first 50 grid points of the southern boundary are boundary points:

```
OB_Jsouth(1:50) = 50*1,
```

8.3.1.5 Equations and key routines

OBCS_READPARMS:

Set OB positions through arrays OB_Jnorth(Nx), OB_Jsouth(Nx), OB_Ieast(Ny), OB_Iwest(Ny) and runtime flags (see Table Table 8.3.1.3).

OBCS_CALC:

Top-level routine for filling values to be applied at OB for T, S, U, V, η into corresponding "slice" arrays (x, z) (y, z) for each OB: `OB[N/S/E/W][t/s/u/v]`; e.g. for salinity array at Southern OB, array name is `OBSt`. Values filled are either

- constant vertical T, S profiles as specified in file data (tRef(Nr), sRef(Nr)) with zero velocities U, V
- T, S, U, V values determined via Orlanski radiation conditions (see below)
- prescribed time-constant or time-varying fields (see below).
- use prescribed boundary fields to compute Stevens boundary conditions.

ORLANSKI:

Orlanski radiation conditions [Orl76] examples can be found in example configurations dome (http://www.rsmas.miami.edu/personal/tamay/DOME/dome.html) and plume_on_slope.

OBCS_PRESCRIBE_READ:

When useOBCSprescribe = `.TRUE.` the model tries to read temperature, salinity, u- and v-velocities from files specified in the runtime parameters `OB[N/S/E/W][t/s/u/v]File`. These files are the usual IEEE, big-endian files with dimensions of a section along an open boundary:

- For North/South boundary files the dimensions are $(N_x \times N_r \times$ time levels$)$, for East/West boundary files the dimensions are $(N_y \times N_r \times$ time levels$)$.
- If a non-linear free surface is used (Section 2.10.2), additional files `OB[N/S/E/W]etaFile` for the sea surface height eta with dimension $(N_{x/y} \times$ time levels$)$ may be specified.
- If non-hydrostatic dynamics are used (Section 2.9), additional files `OB[N/S/E/W]wFile` for the vertical velocity w with dimensions $(N_{x/y} \times N_r \times$ time levels$)$ can be specified.
- If `useSEAICE=.TRUE.` then additional files `OB[N/S/E/W][a,h,sl,sn,uice,vice]` for sea ice area, thickness (`HEFF`), seaice salinity, snow and ice velocities $(N_{x/y} \times$ time levels$)$ can be specified.

As in S/R external_fields_load or the `exf`-package, the code reads two time levels for each variable, e.g., OBNu0 and OBNu1, and interpolates linearly between these time levels to obtain the value `OBNu` at the current model time (step). When the `exf`-package is used, the time levels are controlled for each boundary separately in the same way as the `exf`-fields in `data.exf`, namelist `EXF_NML_OBCS`. The runtime flags follow the above naming conventions, e.g., for the western boundary the corresponding flags are `OBCWstartdate1/2` and `OBCWperiod`. Sea-ice boundary values are controlled separately with `siobWstartdate1/2` and `siobWperiod`. When the `exf`-package is not used the time levels are controlled by the runtime flags `externForcingPeriod` and `externForcingCycle` in `data`; see `verification/exp4` for an example.

OBCS_CALC_STEVENS:

The boundary conditions following [Ste90] require the vertically averaged normal velocity (originally specified as a stream function along the open boundary) $\bar{u}_{ob}$ and the tracer fields χ_{ob} (note: passive tracers are currently not implemented and the code stops when package *ptracers* is used together with this option). Currently the code vertically averages the normal velocity as specified in `OB[E,W]u` or `OB[N,S]v`. From these prescribed values the code computes the boundary values for the next timestep $n+1$ as follows (as an example, we use the notation for an eastern or western boundary):

- $u^{n+1}(y,z) = \bar{u}_{ob}(y) + (u')^n(y,z)$ where $(u')^n$ is the deviation from the vertically averaged velocity at timestep n on the boundary. $(u')^n$ is computed in the previous time step n from the intermediate velocity u^* prior to the correction step (see Section 2.2 eq. (2.12)). (This velocity is not available at the beginning of the next time step $n+1$, when S/R `OBCS_CALC/OBCS_CALC_STEVENS` are called, therefore it needs to be saved in S/R `DYNAMICS` by calling S/R `OBCS\_SAVE\_UV\_N` and also stored in a separate restart files `pickup_stevens[N/S/E/W].${iteration}.data`)
- If u^{n+1} is directed into the model domain, the boudary value for tracer χ is restored to the prescribed values:

$$\chi^{n+1} = \chi^n + \frac{\Delta t}{\tau_\chi}(\chi_{ob} - \chi^n)$$

  where τ_χ is the relaxation time scale `T/SrelaxStevens`. The new χ^{n+1} is then subject to the advection by u^{n+1}.
- If u^{n+1} is directed out of the model domain, the tracer χ^{n+1} on the boundary at timestep $n+1$ is estimated from advection out of the domain with $u^{n+1}+c$, where c is a phase velocity estimated as $\frac{1}{2}\frac{\partial\chi}{\partial t}/\frac{\partial\chi}{\partial x}$. The numerical scheme is (as an example for an eastern boundary):

$$\chi^{n+1}_{i_b,j,k} = \chi^n_{i_b,j,k} + \Delta t(u^{n+1}+c)_{i_b,j,k}\frac{\chi^n_{i_b,j,k} - \chi^n_{i_b-1,j,k}}{\Delta x^C_{i_b j}} \text{ if } u^{n+1}_{i_b jk} > 0$$

  where i_b is the boundary index. For test purposes, the phase velocity contribution or the entire advection can be turned off by setting the corresponding parameters `useStevensPhaseVel` and `useStevensAdvection` to `.FALSE.`.

See [Ste90] for details. With this boundary condition specifying the exact net transport across the open boundary is simple, so that balancing the flow with (S/R `OBCS_BALANCE_FLOW` see next paragraph) is usually not necessary.

Special cases where the current implementation is not complete:

- When you use the non-linear free surface option (parameter `nonlinFreeSurf` > 1), the current implementation just assumes that the gradient normal to the open boundary is zero ($\frac{\partial\eta}{\partial n} = 0$). Although this is inconsistent with geostrophic dynamics and the possibility to specify a non-zero tangent velocity together with Stevens BCs for normal velocities, it seems to work. Recommendation: Always specify zero tangential velocities with Stevens BCs.
- There is no code for passive tracers, just a commented template in S/R `obcs_calc_stevens`. This means that passive tracers can be specified independently and are fluxed with the velocities that the Stevens BCs compute, but without the restoring term.
- There are no specific Stevens BCs for sea ice, e.g., *pkg/seaice*. The model uses the default boundary conditions for the sea ice packages.

OBCS_BALANCE_FLOW:

When turned on (`ALLOW_OBCS_BALANCE` defined in `OBCS_OPTIONS.h` and `useOBCSbalance=.true.` in `data.obcs/OBCS_PARM01`), this routine balances the net flow across the open boundaries. By default the net flow across the boundaries is computed and all normal velocities on boundaries are adjusted to obtain zero net inflow.

This behavior can be controlled with the runtime flags `OBCS_balanceFacN/S/E/W`. The values of these flags determine how the net inflow is redistributed as small correction velocities between the individual sections. A value -1 balances an individual boundary, values > 0 determine the relative size of the correction. For example, the values

```
OBCS_balanceFacE = 1.,      OBCS_balanceFacW = -1.,      OBCS_balanceFacN = 2.,
OBCS_balanceFacS = 0.,
```

make the model

- correct Western `OBWu` by substracting a uniform velocity to ensure zero net transport through the Western open boundary;
- correct Eastern and Northern normal flow, with the Northern velocity correction two times larger than the Eastern correction, but *not* the Southern normal flow, to ensure that the total inflow through East, Northern, and Southern open boundary is balanced.

The old method of balancing the net flow for all sections individually can be recovered by setting all flags to -1. Then the normal velocities across each of the four boundaries are modified separately, so that the net volume transport across *each* boundary is zero. For example, for the western boundary at $i = i_b$, the modified velocity is:

$$u(y,z) - \int_{\text{western boundary}} udydz \approx OBNu(jk) - \sum_{jk} OBNu(jk) h_w(i_b jk) \Delta y_G(i_b j) \Delta z(k).$$

This also ensures a net total inflow of zero through all boundaries, but this combination of flags is *not* useful if you want to simulate, for example, a sector of the Southern Ocean with a strong ACC entering through the western and leaving through the eastern boundary, because the value of ''-1" for these flags will make sure that the strong inflow is removed. Clearly, gobal balancing with `OBCS_balanceFacE/W/N/S` ≥ 0 is the preferred method.

With runtime parameter `OBCSbalanceSurf=.TRUE.`, the surface mass flux contribution, say, from surface freshwater flux `EmPmR` is included in the balancing scheme.

OBCS_APPLY_*:

OBCS_SPONGE:

The sponge layer code (turned on with `ALLOW_OBCS_SPONGE` and `useOBCSsponge`) adds a relaxation term to the right-hand-side of the momentum and tracer equations. The variables are relaxed towards the boundary values with a relaxation time scale that increases linearly with distance from the boundary

$$G_\chi^{(\text{sponge})} = -\frac{\chi - [(L-\delta L)\chi_{BC} + \delta L \chi]/L}{[(L-\delta L)\tau_b + \delta L \tau_i]/L} = -\frac{\chi - [(1-l)\chi_{BC} + l\chi]}{[(1-l)\tau_b + l\tau_i]}$$

where χ is the model variable (U/V/T/S) in the interior, χ_{BC} the boundary value, L the thickness of the sponge layer (runtime parameter `spongeThickness` in number of grid points), $\delta L \in [0, L]$ ($\frac{\delta L}{L} = l \in [0, 1]$) the distance from the boundary (also in grid points), and τ_b (runtime parameters `Urelaxobcsbound` and `Vrelaxobcsbound`) and τ_i (runtime parameters `Urelaxobcsinner` and `Vrelaxobcsinner`) the relaxation time scales on the boundary and at the interior termination of the sponge layer. The parameters `Urelaxobcsbound/inner` set the relaxation time scales for the Eastern and Western boundaries, `Vrelaxobcsbound/inner` for the Northern and Southern boundaries.

OB's with nonlinear free surface

OB's with sea ice

8.3.1.6 Flow chart

```
C     !CALLING SEQUENCE:
c ...
```

8.3.1.7 OBCS diagnostics

Diagnostics output is available via the diagnostics package (see Section 9). Available output fields are summarized below:

```
------------------------------------------------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------
```

8.3.1.8 Experiments and tutorials that use obcs

In the directory verification the following experiments use `obcs`:

- exp4: box with 4 open boundaries, simulating flow over a Gaussian bump based on also tests Stevens-boundary conditions;
- dome: based on the project "Dynamics of Overflow Mixing and Entrainment" (http://www.rsmas.miami.edu/personal/tamay/DOME/dome.html) uses Orlanski-BCs;
- internal_wave: uses a heavily modified S/R `OBCS_CALC`
- seaice_obcs: simple example who to use the sea-ice related code based on lab_sea;
- tutorial_plume_on_slope: uses Orlanski-BCs.

8.3.2 RBCS Package

8.3.2.1 Introduction

A package which provides the flexibility to relax fields (temperature, salinity, ptracers, horizontal velocities) in any 3-D location: so could be used as a sponge layer, or as a "source" anywhere in the domain.

For a field (T) at every grid point the tendency is modified so that:

$$\frac{dT}{dt} = \frac{dT}{dt} - \frac{M_{rbc}}{\tau_T}(T - T_{rbc})$$

where M_{rbc} is a 3-D mask (no time dependence) with values between 0 and 1. Where M_{rbc} is 1, relaxing timescale is $1/\tau_T$. Where it is 0 there is no relaxing. The value relaxed to is a 3-D (potentially varying in time) field given by T_{rbc}.

A seperate mask can be used for T,S and ptracers and each of these can be relaxed or not and can have its own timescale τ_T. These are set in data.rbcs (see below).

8.3.2.2 Key subroutines and parameters

The only compile-time parameter you are likely to have to change is in RBCS_SIZE.h, the number of masks, PARAMETER(maskLEN = 3), see below.

Table 8.3.2.2 summarizes the runtime flags that are set in `data.rbcs`, and their default values.

Flag/Parameter	Group	Default	Description
rbcsForcingPeriod	PARM01	0.0	Time interval between forcing fields (in seconds), zero means constant-in-time forcing.
rbcsForcingCycle	PARM01	0.0	Repeat cycle of forcing fields (in seconds), zero means non-cyclic forcing.
rbcsForcingOffset	PARM01	0.0	Time offset of forcing fields (in seconds, default 0); this is relative to time averages starting at $t = 0$, i.e., the first forcing record/file is placed at (rbcsForcingOffset + rbcsForcingPeriod)/2 ; see below for examples.
rbcsSingleTimeFiles	PARM01	FALSE	If `.TRUE.`, forcing fields are given 1 file per rbcsForcingPeriod.
deltaTrbcs	PARM01	deltaTclock	Time step used to compute the iteration numbers for rbcsSingleTimeFiles = `.TRUE.`.
rbcsVanishingTime	PARM01	0.0	If rbcsVanishingTime > 0, the relaxation strength reduces linearly to vanish at myTime == rbcsVanishingTime.
rbcsIter0	PARM01	0	Shift in iteration numbers used to label files if rbcsSingleTimeFiles = `.TRUE.` (see below for examples).
useRBCtemp, useRBCsalt, useRBCuVel, useRCvVel	PARM01	FALSE	Whether to use RBCS for T/S/U/V.
tauRelaxT, tauRelaxT, tauRelaxT, tauRelaxT	PARM01	0.0	Timescales in seconds of relaxing in T/S/U/V (τ_T in equation above). Where mask is 1, relax rate will be 1/tauRelaxT. Must be set if the corresponding `useRBCxxx` is `TRUE`.
relaxMaskFile (irbc)	PARM01	' '	Filename of 3-D file with mask (M_{rbc} in equation above). Need a file for each irbc (1=temperature, 2=salinity, 3=ptracer1, 4=ptracer2, etc). If maskLEN is les than the number of tracers, then `relaxMaskFile(maskLEN)` is used for all remaining tracers.
relaxMaskUFile, relaxMaskVFile	PARM01	' '	Filename of 3-D file with mask for U/V.
relaxTFile, relaxSFile, relaxUFile, relaxVFile	PARM01	' '	Name of file where the field that need to be relaxed to (T_{rbc} in equation above) is stored. The file must contain 3-D records to match the model domain. If rbcsSingleTimeFiles = `.FALSE.`, it must have one record for each forcing period. Otherwise there must be a separate file for each period and a 10-digit iteration number is appended to the file name (see Table [*Timing of RBCS relaxation fields*] and examples below).
useRBCptracers	PARM02	FALSE	**DEPRECATED** Use one useRBCpTrNum per tracer instead.
useRBCpTrNum (iTrc)	PARM02	FALSE	Whether to use RBCS for the corresponding passive tracer.
tauRelaxPTR (iTrc)	PARM02	0.0	Relaxing timescale for the corresponding ptracer.
relaxPtracerFile (iTrc)	PARM02	' '	File with relax fields for the corresponding ptracer.

8.3.2.3 Timing of relaxation forcing fields

For constant-in-time relaxation, set rbcsForcingPeriod =0. For time-varying relaxation, Table Table 8.1 illustrates the relation between model time and forcing fields (either records in one big file or, for rbcsSingleTimeFiles = `.TRUE.`

, individual files labeled with an iteration number). With rbcsSingleTimeFiles = `.TRUE.` , this is the same as in the offline package, except that the forcing offset is in seconds.

Table 8.1: Timing of RBCS relaxation fields

	rbcsSingleTimeFiles = T		F
	$c = 0$	$c \neq 0$	$c \neq 0$
model time	**file number**	**file number**	**record**
$t_0 - p/2$	i_0	$i_0 + c/\Delta t_{\text{rbcs}}$	c/p
$t_0 + p/2$	$i_0 + p/\Delta t_{\text{rbcs}}$	$i_0 + p/\Delta t_{\text{rbcs}}$	1
$t_0 + p + p/2$	$i_0 + 2p/\Delta t_{\text{rbcs}}$	$i_0 + 2p/\Delta t_{\text{rbcs}}$	2
...	...	...	...
$t_0 + c - p/2$	...	$i_0 + c/\Delta t_{\text{rbcs}}$	c/p
...	...	...	...

where

p = rbcsForcingPeriod

c = rbcsForcingCycle

t_0 = rbcsForcingOffset

i_0 = rbcsIter0

Δt_{rbcs} = deltaTrbcs

8.3.2.4 Example 1: forcing with time averages starting at $t = 0$

Cyclic data in a single file

Set rbcsSingleTimeFiles = `.FALSE.` and rbcsForcingOffset = 0, and the model will start by interpolating the last and first records of rbcs data, placed at $-p/2$ and $p/2$, resp., as appropriate for fields averaged over the time intervals $[-p, 0]$ and $[0, p]$.

Non-cyclic data, multiple files

Set rbcsForcingCycle = 0 and rbcsSingleTimeFiles = `.TRUE.` . With rbcsForcingOffset = 0, rbcsIter0 = 0 and deltaTrbcs = rbcsForcingPeriod, the model would then start by interpolating data from files `relax\*File.0000000000.data` and `relax\*File.0000000001.data`, ... , again placed at $-p/2$ and $p/2$.

8.3.2.5 Example 2: forcing with snapshots starting at $t = 0$

Cyclic data in a single file

Set rbcsSingleTimeFiles = `.FALSE.` and rbcsForcingOffset =$-p/2$, and the model will start forcing with the first record at $t = 0$.

Non-cyclic data, multiple files

Set rbcsForcingCycle = 0 and rbcsSingleTimeFiles = `.TRUE.`. In this case, it is more natural to set rbcsForcingOffset =$+p/2$. With rbcsIter0 = 0 and deltaTrbcs = rbcsForcingPeriod, the model would then start with data from files

relax*File.0000000000.data at $t = 0$. It would then proceed to interpolate between this file and files relax*File.0000000001.data at $t =$ rbcsForcingPeriod.

8.3.2.6 Do's and Don'ts

8.3.2.7 Reference Material

8.3.2.8 Experiments and tutorials that use rbcs

In the directory, the following experiments use rbcs:

- exp4 : box with 4 open boundaries, simulating flow over a Gaussian bump based on [AHM97]

8.3.3 PTRACERS Package

8.3.3.1 Introduction

This is a ''passive" tracer package. Passive here means that the tracers don't affect the density of the water (as opposed to temperature and salinity) so no not actively affect the physics of the ocean. Tracers are initialized, advected, diffused and various outputs are taken care of in this package. For methods to add additional sources and sinks of tracers use the *gchem Package*.

Can use up tp 3843 tracers. But can not use the *diagnostics package* with more than about 90 tracers. Use utils/matlab/ioLb2num.m and num2ioLb.m to find correspondence between tracer number and tracer designation in the code for more than 99 tracers (since tracers only have two digit designations).

8.3.3.2 Equations

8.3.3.3 Key subroutines and parameters

The only code you should have to modify is: PTRACERS_SIZE.h where you need to set in the number of tracers to be used in the experiment: PTRACERS_num.

Run time parameters set in data.ptracers:

- PTRACERS_Iter0 which is the integer timestep when the tracer experiment is initialized. If nIter0 = PTRACERS_Iter0 then the tracers are initialized to zero or from initial files. If nIter0 > PTRACERS_Iter0 then tracers (and previous timestep tendency terms) are read in from a the ptracers pickup file. Note that tracers of zeros will be carried around if nIter0 < PTRACERS_Iter0.
- PTRACERS_numInUse: number of tracers to be used in the run (needs to be <= PTRACERS_num set in PTRACERS_SIZE.h)
- PTRACERS_dumpFreq: defaults to dumpFreq (set in data)
- PTRACERS_taveFreq: defaults to taveFreq (set in data)
- PTRACERS_monitorFreq: defaults to monitorFreq (set in data)
- PTRACERS_timeave_mnc: needs useMNC, timeave_mnc, default to false
- PTRACERS_snapshot_mnc: needs useMNC , snapshot_mnc, default to false
- PTRACERS_monitor_mnc: needs useMNC, monitor_mnc, default to false
- PTRACERS_pickup_write_mnc: needs useMNC, pickup_write_mnc, default to false
- PTRACERS_pickup_read_mnc: needs useMNC, pickup_read_mnc, default to false

- PTRACERS_useRecords: defaults to false. If true, will write all tracers in a single file, otherwise each tracer in a seperate file.

The following can be set for each tracer (tracer number iTrc):

- PTRACERS_advScheme (iTrc) will default to saltAdvScheme (set in data). For other options see Table *MITgcm Advection Schemes*.
- PTRACERS_ImplVertAdv (iTrc): implicit vertical advection flag, defaults to false.
- PTRACERS_diffKh (iTrc): horizontal Laplacian Diffusivity, defaults to diffKhS (set in data).
- PTRACERS_diffK4 (iTrc): Biharmonic Diffusivity, defaults to diffK4S (set in data).
- PTRACERS_diffKr (iTrc): vertical diffusion, defaults to un-set.
- PTRACERS_diffKrNr (k,iTrc): level specific vertical diffusion, defaults to diffKrNrS. Will be set to PTRACERS_diffKr if this is set.
- PTRACERS_ref (k,iTrc): reference tracer value for each level k, defaults to 0. Currently only used for dilution/concentration of tracers at surface if PTRACERS_EvPrRn (iTrc) is set and convertFW2Salt (set in data) is set to something other than -1 (note default is convertFW2Salt = 35).
- PTRACERS_EvPrRn (iTrc): tracer concentration in freshwater. Needed for calculation of dilution/concentration in surface layer due to freshwater addition/evaporation. Defaults to un-set in which case no dilution/concentration occurs.
- PTRACERS_useGMRedi (iTrc): apply GM or not. Defaults to useGMREdi.
- PTRACERS_useKPP (iTrc): apply KPP or not. Defaults to useKPP.
- PTRACERS_initialFile (iTrc): file with initial tracer concentration. Will be used if `PTRACERS_Iter0 = nIter0`. Default is no name, in which case tracer is initialised as zero. If `PTRACERS_Iter0 < nIter0`, then tracer concentration will come from `pickup_ptracer`.
- PTRACERS_names (iTrc): tracer name. Needed for netcdf. Defaults to nothing.
- PTRACERS_long_names (iTrc): optional name in long form of tracer.
- PTRACERS_units (iTrc): optional units of tracer.

8.3.3.4 PTRACERS Diagnostics

Note that these will only work for 90 or less tracers (some problems with the numbering/designation over this number)

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
TRAC01  | 15 |SM P    MR      |mol C/m         |Mass-Weighted Dissolved Inorganic␣
↪Carbon
UTRAC01 | 15 |UU   171MR      |mol C/m.m/s     |Zonal Mass-Weighted Transp of␣
↪Dissolved Inorganic Carbon
VTRAC01 | 15 |VV   170MR      |mol C/m.m/s     |Merid Mass-Weighted Transp of␣
↪Dissolved Inorganic Carbon
WTRAC01 | 15 |WM      MR      |mol C/m.m/s     |Vert  Mass-Weighted Transp of␣
↪Dissolved Inorganic Carbon
ADVrTr01| 15 |WM      LR      |mol C/m.m^3/s   |Vertical   Advective Flux of␣
↪Dissolved Inorganic Carbon
ADVxTr01| 15 |UU   175MR      |mol C/m.m^3/s   |Zonal      Advective Flux of␣
↪Dissolved Inorganic Carbon
ADVyTr01| 15 |VV   174MR      |mol C/m.m^3/s   |Meridional Advective Flux of␣
↪Dissolved Inorganic Carbon
```

(continues on next page)

(continued from previous page)

```
DFrETr01| 15 |WM      LR      |mol C/m.m^3/s   |Vertical Diffusive Flux of Dissolved␣
↪Inorganic Carbon (Explicit part)
DIFxTr01| 15 |UU   178MR      |mol C/m.m^3/s   |Zonal      Diffusive Flux of␣
↪Dissolved Inorganic Carbon
DIFyTr01| 15 |VV   177MR      |mol C/m.m^3/s   |Meridional Diffusive Flux of␣
↪Dissolved Inorganic Carbon
DFrITr01| 15 |WM      LR      |mol C/m.m^3/s   |Vertical Diffusive Flux of Dissolved␣
↪Inorganic Carbon (Implicit part)
TRAC02  | 15 |SM P    MR      |mol eq/         |Mass-Weighted Alkalinity
UTRAC02 | 15 |UU   182MR      |mol eq/.m/s     |Zonal Mass-Weighted Transp of␣
↪Alkalinity
VTRAC02 | 15 |VV   181MR      |mol eq/.m/s     |Merid Mass-Weighted Transp of␣
↪Alkalinity
WTRAC02 | 15 |WM      MR      |mol eq/.m/s     |Vert  Mass-Weighted Transp of␣
↪Alkalinity
ADVrTr02| 15 |WM      LR      |mol eq/.m^3/s   |Vertical   Advective Flux of␣
↪Alkalinity
ADVxTr02| 15 |UU   186MR      |mol eq/.m^3/s   |Zonal      Advective Flux of␣
↪Alkalinity
ADVyTr02| 15 |VV   185MR      |mol eq/.m^3/s   |Meridional Advective Flux of␣
↪Alkalinity
DFrETr02| 15 |WM      LR      |mol eq/.m^3/s   |Vertical Diffusive Flux of Alkalinity␣
↪(Explicit part)
DIFxTr02| 15 |UU   189MR      |mol eq/.m^3/s   |Zonal      Diffusive Flux of␣
↪Alkalinity
DIFyTr02| 15 |VV   188MR      |mol eq/.m^3/s   |Meridional Diffusive Flux of␣
↪Alkalinity
DFrITr02| 15 |WM      LR      |mol eq/.m^3/s   |Vertical Diffusive Flux of Alkalinity␣
↪(Implicit part)
TRAC03  | 15 |SM P    MR      |mol P/m         |Mass-Weighted Phosphate
UTRAC03 | 15 |UU   193MR      |mol P/m.m/s     |Zonal Mass-Weighted Transp of␣
↪Phosphate
VTRAC03 | 15 |VV   192MR      |mol P/m.m/s     |Merid Mass-Weighted Transp of␣
↪Phosphate
WTRAC03 | 15 |WM      MR      |mol P/m.m/s     |Vert  Mass-Weighted Transp of␣
↪Phosphate
ADVrTr03| 15 |WM      LR      |mol P/m.m^3/s   |Vertical   Advective Flux of Phosphate
ADVxTr03| 15 |UU   197MR      |mol P/m.m^3/s   |Zonal      Advective Flux of Phosphate
ADVyTr03| 15 |VV   196MR      |mol P/m.m^3/s   |Meridional Advective Flux of Phosphate
DFrETr03| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Phosphate␣
↪(Explicit part)
DIFxTr03| 15 |UU   200MR      |mol P/m.m^3/s   |Zonal      Diffusive Flux of Phosphate
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
DIFyTr03| 15 |VV   199MR      |mol P/m.m^3/s   |Meridional Diffusive Flux of Phosphate
DFrITr03| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Phosphate␣
↪(Implicit part)
TRAC04  | 15 |SM P    MR      |mol P/m         |Mass-Weighted Dissolved Organic␣
↪Phosphorus
UTRAC04 | 15 |UU   204MR      |mol P/m.m/s     |Zonal Mass-Weighted Transp of␣
↪Dissolved Organic Phosphorus
VTRAC04 | 15 |VV   203MR      |mol P/m.m/s     |Merid Mass-Weighted Transp of␣
↪Dissolved Organic Phosphorus
WTRAC04 | 15 |WM      MR      |mol P/m.m/s     |Vert  Mass-Weighted Transp of␣
↪Dissolved Organic Phosphorus
ADVrTr04| 15 |WM      LR      |mol P/m.m^3/s   |Vertical   Advective Flux of␣
↪Dissolved Organic Phosphorus
```

(continues on next page)

(continued from previous page)

```
ADVxTr04| 15 |UU   208MR      |mol P/m.m^3/s   |Zonal      Advective Flux of␣
↪Dissolved Organic Phosphorus
ADVyTr04| 15 |VV   207MR      |mol P/m.m^3/s   |Meridional Advective Flux of␣
↪Dissolved Organic Phosphorus
DFrETr04| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Dissolved␣
↪Organic Phosphorus (Explicit part)
DIFxTr04| 15 |UU   211MR      |mol P/m.m^3/s   |Zonal      Diffusive Flux of␣
↪Dissolved Organic Phosphorus
DIFyTr04| 15 |VV   210MR      |mol P/m.m^3/s   |Meridional Diffusive Flux of␣
↪Dissolved Organic Phosphorus
DFrITr04| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Dissolved␣
↪Organic Phosphorus (Implicit part)
TRAC05  | 15 |SM P    MR      |mol O/m         |Mass-Weighted Dissolved Oxygen
UTRAC05 | 15 |UU   215MR      |mol O/m.m/s     |Zonal Mass-Weighted Transp of␣
↪Dissolved Oxygen
VTRAC05 | 15 |VV   214MR      |mol O/m.m/s     |Merid Mass-Weighted Transp of␣
↪Dissolved Oxygen
WTRAC05 | 15 |WM      MR      |mol O/m.m/s     |Vert  Mass-Weighted Transp of␣
↪Dissolved Oxygen
ADVrTr05| 15 |WM      LR      |mol O/m.m^3/s   |Vertical   Advective Flux of␣
↪Dissolved Oxygen
ADVxTr05| 15 |UU   219MR      |mol O/m.m^3/s   |Zonal      Advective Flux of␣
↪Dissolved Oxygen
ADVyTr05| 15 |VV   218MR      |mol O/m.m^3/s   |Meridional Advective Flux of␣
↪Dissolved Oxygen
DFrETr05| 15 |WM      LR      |mol O/m.m^3/s   |Vertical Diffusive Flux of Dissolved␣
↪Oxygen (Explicit part)
DIFxTr05| 15 |UU   222MR      |mol O/m.m^3/s   |Zonal      Diffusive Flux of␣
↪Dissolved Oxygen
DIFyTr05| 15 |VV   221MR      |mol O/m.m^3/s   |Meridional Diffusive Flux of␣
↪Dissolved Oxygen
DFrITr05| 15 |WM      LR      |mol O/m.m^3/s   |Vertical Diffusive Flux of Dissolved␣
↪Oxygen (Implicit part)
```

8.3.3.5 Do's and Don'ts

8.3.3.6 Reference Material

8.4 Ocean Packages

8.4.1 GMREDI: Gent-McWilliams/Redi SGS Eddy Parameterization

There are two parts to the Redi/GM parameterization of geostrophic eddies. The first, the Redi scheme [Red82], aims to mix tracer properties along isentropes (neutral surfaces) by means of a diffusion operator oriented along the local isentropic surface. The second part, GM [GM90][GWMM95] , adiabatically re-arranges tracers through an advective flux where the advecting flow is a function of slope of the isentropic surfaces.

The first GCM implementation of the Redi scheme was by [Cox87] in the GFDL ocean circulation model. The original approach failed to distinguish between isopycnals and surfaces of locally referenced potential density (now called neutral surfaces) which are proper isentropes for the ocean. As will be discussed later, it also appears that the Cox implementation is susceptible to a computational mode. Due to this mode, the Cox scheme requires a background lateral diffusion to be present to conserve the integrity of the model fields.

The GM parameterization was then added to the GFDL code in the form of a non-divergent bolus velocity. The method

defines two stream-functions expressed in terms of the isoneutral slopes subject to the boundary condition of zero value on upper and lower boundaries. The horizontal bolus velocities are then the vertical derivative of these functions. Here in lies a problem highlighted by [GGP+98]: the bolus velocities involve multiple derivatives on the potential density field, which can consequently give rise to noise. Griffies et al. point out that the GM bolus fluxes can be identically written as a skew flux which involves fewer differential operators. Further, combining the skew flux formulation and Redi scheme, substantial cancellations take place to the point that the horizontal fluxes are unmodified from the lateral diffusion parameterization.

8.4.1.1 Redi scheme: Isopycnal diffusion

The Redi scheme diffuses tracers along isopycnals and introduces a term in the tendency (rhs) of such a tracer (here τ) of the form:

$$\nabla \cdot \kappa_\rho \mathbf{K_{Redi}} \nabla \tau$$

where κ_ρ is the along isopycnal diffusivity and $\mathbf{K_{Redi}}$ is a rank 2 tensor that projects the gradient of τ onto the isopycnal surface. The unapproximated projection tensor is:

$$\mathbf{K_{Redi}} = \frac{\mathbf{1}}{\mathbf{1} + |\mathbf{S}|^\mathbf{2}} \begin{pmatrix} 1+S_y^2 & -S_x S_y & S_x \\ -S_x S_y & 1+S_x^2 & S_y \\ S_x & S_y & |S|^2 \end{pmatrix}$$

Here, $S_x = -\partial_x \sigma / \partial_z \sigma$ and $S_y = -\partial_y \sigma / \partial_z \sigma$ are the components of the isoneutral slope.

The first point to note is that a typical slope in the ocean interior is small, say of the order 10^{-4}. A maximum slope might be of order 10^{-2} and only exceeds such in unstratified regions where the slope is ill defined. It is therefore justifiable, and customary, to make the small slope approximation, $|S| << 1$. The Redi projection tensor then becomes:

$$\mathbf{K_{Redi}} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ S_x & S_y & |S|^2 \end{pmatrix}$$

8.4.1.2 GM parameterization

The GM parameterization aims to represent the "advective" or "transport" effect of geostrophic eddies by means of a "bolus" velocity, $\mathbf{u}^\star$. The divergence of this advective flux is added to the tracer tendency equation (on the rhs):

$$-\nabla \cdot \tau \mathbf{u}^\star$$

The bolus velocity $\mathbf{u}^\star$ is defined as the rotational of a streamfunction $\mathbf{F}^\star$=$(F_x^\star, F_y^\star, 0)$:

$$\mathbf{u}^\star = \nabla \times \mathbf{F}^\star = \begin{pmatrix} -\partial_z F_y^\star \\ +\partial_z F_x^\star \\ \partial_x F_y^\star - \partial_y F_x^\star \end{pmatrix},$$

and thus is automatically non-divergent. In the GM parameterization, the streamfunction is specified in terms of the isoneutral slopes S_x and S_y:

$$\begin{aligned} F_x^\star &= -\kappa_{GM} S_y \\ F_y^\star &= \kappa_{GM} S_x \end{aligned}$$

with boundary conditions $F_x^\star = F_y^\star = 0$ on upper and lower boundaries. In the end, the bolus transport in the GM parameterization is given by:

$$\mathbf{u}^\star = \begin{pmatrix} u^\star \\ v^\star \\ w^\star \end{pmatrix} = \begin{pmatrix} -\partial_z(\kappa_{GM} S_x) \\ -\partial_z(\kappa_{GM} S_y) \\ \partial_x(\kappa_{GM} S_x) + \partial_y(\kappa_{GM} S_y) \end{pmatrix}$$

This is the form of the GM parameterization as applied by Donabasaglu, 1997, in MOM versions 1 and 2.

Note that in the MITgcm, the variables containing the GM bolus streamfunction are:

$$\begin{pmatrix} GM\_PsiX \\ GM\_PsiY \end{pmatrix} = \begin{pmatrix} \kappa_{GM} S_x \\ \kappa_{GM} S_y \end{pmatrix} = \begin{pmatrix} F_y^\star \\ -F_x^\star \end{pmatrix}.$$

8.4.1.3 Griffies Skew Flux

[Gri98] notes that the discretisation of bolus velocities involves multiple layers of differencing and interpolation that potentially lead to noisy fields and computational modes. He pointed out that the bolus flux can be re-written in terms of a non-divergent flux and a skew-flux:

$$\begin{aligned} \mathbf{u}^\star\tau = & \begin{pmatrix} -\partial_z(\kappa_{GM} S_x)\tau \\ -\partial_z(\kappa_{GM} S_y)\tau \\ (\partial_x \kappa_{GM} S_x + \partial_y \kappa_{GM} S_y)\tau \end{pmatrix} \\ = & \begin{pmatrix} -\partial_z(\kappa_{GM} S_x \tau) \\ -\partial_z(\kappa_{GM} S_y \tau) \\ \partial_x(\kappa_{GM} S_x \tau) + \partial_y(\kappa_{GM} S_y \tau) \end{pmatrix} + \begin{pmatrix} \kappa_{GM} S_x \partial_z \tau \\ \kappa_{GM} S_y \partial_z \tau \\ -\kappa_{GM} S_x \partial_x \tau - \kappa_{GM} S_y \partial_y \tau \end{pmatrix} \end{aligned}$$

The first vector is non-divergent and thus has no effect on the tracer field and can be dropped. The remaining flux can be written:

$$\mathbf{u}^\star\tau = -\kappa_{\mathbf{GM}} \mathbf{K}_{\mathbf{GM}} \nabla\tau$$

where

$$\mathbf{K}_{\mathbf{GM}} = \begin{pmatrix} 0 & 0 & -S_x \\ 0 & 0 & -S_y \\ S_x & S_y & 0 \end{pmatrix}$$

is an anti-symmetric tensor.

This formulation of the GM parameterization involves fewer derivatives than the original and also involves only terms that already appear in the Redi mixing scheme. Indeed, a somewhat fortunate cancellation becomes apparent when we use the GM parameterization in conjunction with the Redi isoneutral mixing scheme:

$$\kappa_\rho \mathbf{K}_{\mathbf{Redi}} \nabla\tau - \mathbf{u}^\star\tau = (\kappa_\rho \mathbf{K}_{\mathbf{Redi}} + \kappa_{\mathbf{GM}} \mathbf{K}_{\mathbf{GM}}) \nabla\tau$$

In the instance that $\kappa_{GM} = \kappa_\rho$ then

$$\kappa_\rho \mathbf{K}_{\mathbf{Redi}} + \kappa_{\mathbf{GM}} \mathbf{K}_{\mathbf{GM}} = \kappa_\rho \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2S_x & 2S_y & |S|^2 \end{pmatrix}$$

which differs from the variable Laplacian diffusion tensor by only two non-zero elements in the z-row.

Subroutine

S/R GMREDI_CALC_TENSOR (*pkg/gmredi/gmredi_calc_tensor.F*)

σ_x: **SlopeX** (argument on entry)

σ_y: **SlopeY** (argument on entry)

σ_z: **SlopeY** (argument)

S_x: **SlopeX** (argument on exit)

S_y: **SlopeY** (argument on exit)

8.4.1.4 Variable κ_{GM}

[VMHS97] suggest making the eddy coefficient, κ_{GM}, a function of the Eady growth rate, $|f|/\sqrt{Ri}$. The formula involves a non-dimensional constant, α, and a length-scale L:

$$\kappa_{GM} = \alpha L^2 \overline{\frac{|f|}{\sqrt{Ri}}}^z$$

where the Eady growth rate has been depth averaged (indicated by the over-line). A local Richardson number is defined $Ri = N^2/(\partial u/\partial z)^2$ which, when combined with thermal wind gives:

$$\frac{1}{Ri} = \frac{(\frac{\partial u}{\partial z})^2}{N^2} = \frac{(\frac{g}{f\rho_o}|\nabla\sigma|)^2}{N^2} = \frac{M^4}{|f|^2 N^2}$$

where M^2 is defined $M^2 = \frac{g}{\rho_o}|\nabla\sigma|$. Substituting into the formula for κ_{GM} gives:

$$\kappa_{GM} = \alpha L^2 \overline{\frac{M^2}{N}}^z = \alpha L^2 \overline{\frac{M^2}{N^2}N}^z = \alpha L^2 \overline{|S|N}^z$$

8.4.1.5 Tapering and stability

Experience with the GFDL model showed that the GM scheme has to be matched to the convective parameterization. This was originally expressed in connection with the introduction of the KPP boundary layer scheme [LMD94] but in fact, as subsequent experience with the MIT model has found, is necessary for any convective parameterization.

Subroutine

S/R GMREDI_SLOPE_LIMIT (*pkg/gmredi/gmredi_slope_limit.F*)

σ_x, s_x: **SlopeX** (argument)

σ_y, s_y: **SlopeY** (argument)

σ_z: **dSigmadRReal** (argument)

z^*_σ: **dRdSigmaLtd** (argument)

8.4.1.6 Slope clipping

Deep convection sites and the mixed layer are indicated by homogenized, unstable or nearly unstable stratification. The slopes in such regions can be either infinite, very large with a sign reversal or simply very large. From a numerical point of view, large slopes lead to large variations in the tensor elements (implying large bolus flow) and can be numerically unstable. This was first recognized by [Cox87] who implemented "slope clipping" in the isopycnal mixing tensor. Here, the slope magnitude is simply restricted by an upper limit:

$$\begin{aligned}
|\nabla\sigma| &= \sqrt{\sigma_x^2 + \sigma_y^2} \\
S_{lim} &= -\frac{|\nabla\sigma|}{S_{max}} \quad \text{where } S_{max} \text{ is a parameter} \\
\sigma_z^\star &= \min(\sigma_z, S_{lim}) \\
[s_x, s_y] &= -\frac{[\sigma_x, \sigma_y]}{\sigma_z^\star}
\end{aligned}$$

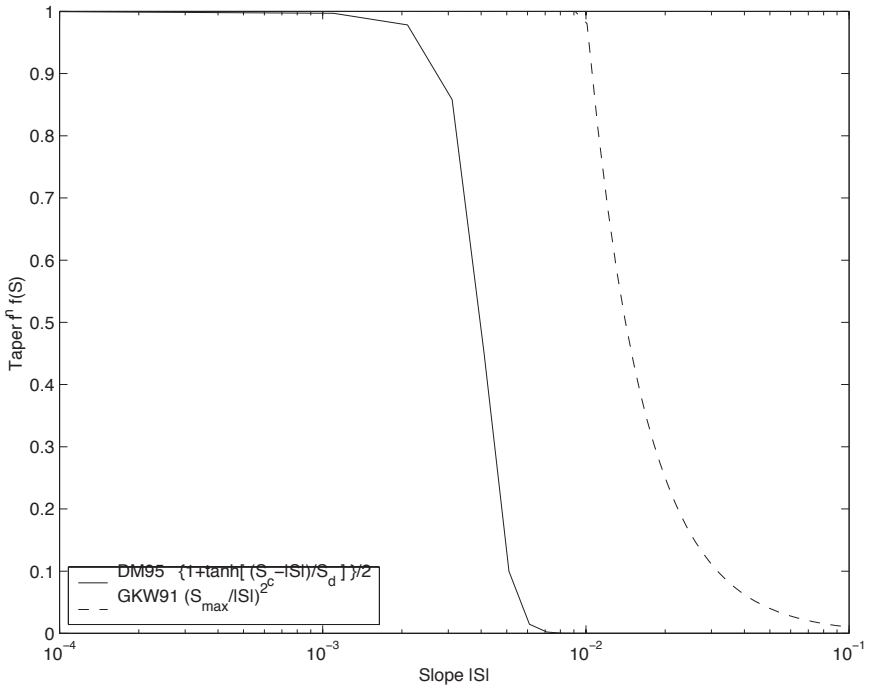

Figure 8.6: Taper functions used in GKW91 and DM95.

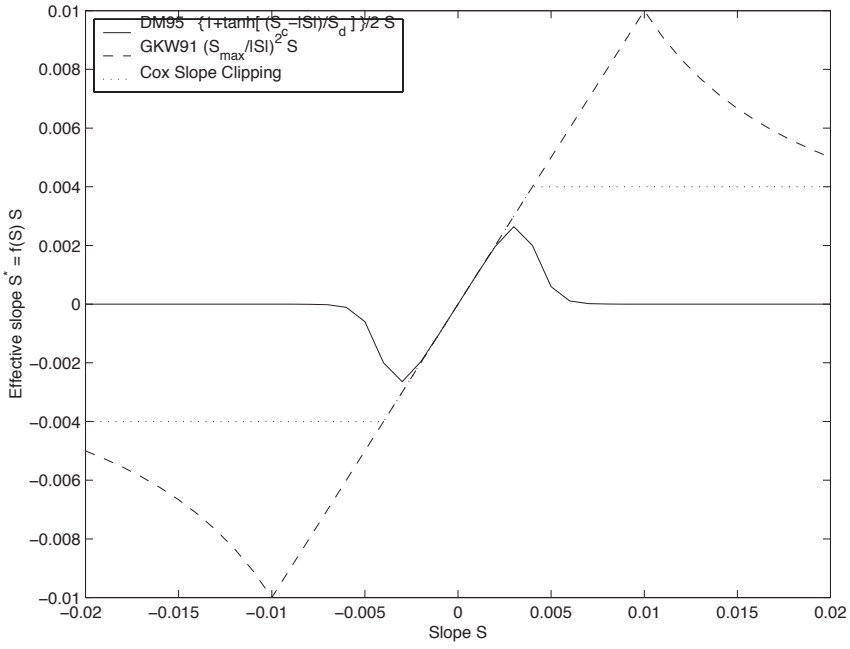

Figure 8.7: Effective slope as a function of 'true' slope using Cox slope clipping, GKW91 limiting and DM95 limiting.

Notice that this algorithm assumes stable stratification through the "min" function. In the case where the fluid is well stratified ($\sigma_z < S_{lim}$) then the slopes evaluate to:

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma_z}$$

while in the limited regions ($\sigma_z > S_{lim}$) the slopes become:

$$[s_x, s_y] = \frac{[\sigma_x, \sigma_y]}{|\nabla\sigma|/S_{max}}$$

so that the slope magnitude is limited $\sqrt{s_x^2 + s_y^2} = S_{max}$.

The slope clipping scheme is activated in the model by setting **GM_taper_scheme = 'clipping'** in *data.gmredi*.

Even using slope clipping, it is normally the case that the vertical diffusion term (with coefficient $\kappa_\rho \mathbf{K}_{33} = \kappa_\rho S_{max}^2$) is large and must be time-stepped using an implicit procedure (see section on discretisation and code later). Fig. [fig-mixedlayer] shows the mixed layer depth resulting from a) using the GM scheme with clipping and b) no GM scheme (horizontal diffusion). The classic result of dramatically reduced mixed layers is evident. Indeed, the deep convection sites to just one or two points each and are much shallower than we might prefer. This, it turns out, is due to the over zealous re-stratification due to the bolus transport parameterization. Limiting the slopes also breaks the adiabatic nature of the GM/Redi parameterization, re-introducing diabatic fluxes in regions where the limiting is in effect.

8.4.1.7 Tapering: Gerdes, Koberle and Willebrand, Clim. Dyn. 1991

The tapering scheme used in [GKW91] addressed two issues with the clipping method: the introduction of large vertical fluxes in addition to convective adjustment fluxes is avoided by tapering the GM/Redi slopes back to zero in low-stratification regions; the adjustment of slopes is replaced by a tapering of the entire GM/Redi tensor. This means the direction of fluxes is unaffected as the amplitude is scaled.

The scheme inserts a tapering function, $f_1(S)$, in front of the GM/Redi tensor:

$$f_1(S) = \min\left[1, \left(\frac{S_{max}}{|S|}\right)^2\right]$$

where S_{max} is the maximum slope you want allowed. Where the slopes, $|S| < S_{max}$ then $f_1(S) = 1$ and the tensor is un-tapered but where $|S| \geq S_{max}$ then $f_1(S)$ scales down the tensor so that the effective vertical diffusivity term $\kappa f_1(S)|S|^2 = \kappa S_{max}^2$.

The GKW91 tapering scheme is activated in the model by setting **GM_taper_scheme = 'gkw91'** in *data.gmredi*.

8.4.1.8 Tapering: Danabasoglu and McWilliams, J. Clim. 1995

The tapering scheme used by followed a similar procedure but used a different tapering function, $f_1(S)$:

$$f_1(S) = \frac{1}{2}\left(1 + \tanh\left[\frac{S_c - |S|}{S_d}\right]\right)$$

where $S_c = 0.004$ is a cut-off slope and $S_d = 0.001$ is a scale over which the slopes are smoothly tapered. Functionally, the operates in the same way as the GKW91 scheme but has a substantially lower cut-off, turning off the GM/Redi SGS parameterization for weaker slopes.

The DM95 tapering scheme is activated in the model by setting **GM_taper_scheme = 'dm95'** in *data.gmredi*.

8.4.1.9 Tapering: Large, Danabasoglu and Doney, JPO 1997

The tapering used in [LDDM97] is based on the DM95 tapering scheme, but also tapers the scheme with an additional function of height, $f_2(z)$, so that the GM/Redi SGS fluxes are reduced near the surface:

$$f_2(z) = \frac{1}{2}\left(1 + \sin(\pi \frac{z}{D} - \frac{\pi}{2})\right)$$

where $D = L_\rho |S|$ is a depth-scale and $L_\rho = c/f$ with $c = 2$ m s:math:^{-1}. This tapering with height was introduced to fix some spurious interaction with the mixed-layer KPP parameterization.

The LDD97 tapering scheme is activated in the model by setting **GM_taper_scheme = 'ldd97'** in *data.gmredi*.

8.4.1.10 Package Reference

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
GM_VisbK|  1 |SM P    M1      |m^2/s           |Mixing coefficient from Visbeck etal␣
↪parameterization
GM_Kux  | 15 |UU P 177MR      |m^2/s           |K_11 element (U.point, X.dir) of GM-
↪Redi tensor
GM_Kvy  | 15 |VV P 176MR      |m^2/s           |K_22 element (V.point, Y.dir) of GM-
↪Redi tensor
GM_Kuz  | 15 |UU   179MR      |m^2/s           |K_13 element (U.point, Z.dir) of GM-
↪Redi tensor
GM_Kvz  | 15 |VV   178MR      |m^2/s           |K_23 element (V.point, Z.dir) of GM-
↪Redi tensor
GM_Kwx  | 15 |UM   181LR      |m^2/s           |K_31 element (W.point, X.dir) of GM-
↪Redi tensor
GM_Kwy  | 15 |VM   180LR      |m^2/s           |K_32 element (W.point, Y.dir) of GM-
↪Redi tensor
GM_Kwz  | 15 |WM P    LR      |m^2/s           |K_33 element (W.point, Z.dir) of GM-
↪Redi tensor
GM_PsiX | 15 |UU   184LR      |m^2/s           |GM Bolus transport stream-function :␣
↪X component
GM_PsiY | 15 |VV   183LR      |m^2/s           |GM Bolus transport stream-function :␣
↪Y component
GM_KuzTz| 15 |UU   186MR      |degC.m^3/s      |Redi Off-diagonal Tempetature flux: X␣
↪component
GM_KvzTz| 15 |VV   185MR      |degC.m^3/s      |Redi Off-diagonal Tempetature flux: Y␣
↪component
```

8.4.1.11 Experiments and tutorials that use gmredi

- Global Ocean tutorial, in tutorial_global_oce_latlon verification directory, described in section [sec:eg-global]
- Front Relax experiment, in front_relax verification directory.
- Ideal 2D Ocean experiment, in ideal_2D_oce verification directory.

8.4.2 KPP: Nonlocal K-Profile Parameterization for Vertical Mixing

Authors: Dimitris Menemenlis and Patrick Heimbach

8.4.2.1 Introduction

The nonlocal K-Profile Parameterization (KPP) scheme of [LMD94] unifies the treatment of a variety of unresolved processes involved in vertical mixing. To consider it as one mixing scheme is, in the view of the authors, somewhat misleading since it consists of several entities to deal with distinct mixing processes in the ocean's surface boundary layer, and the interior:

1. mixing in the interior is goverened by shear instability (modeled as function of the local gradient Richardson number), internal wave activity (assumed constant), and double-diffusion (not implemented here).
2. a boundary layer depth h or `hbl` is determined at each grid point, based on a critical value of turbulent processes parameterized by a bulk Richardson number;
3. mixing is strongly enhanced in the boundary layer under the stabilizing or destabilizing influence of surface forcing (buoyancy and momentum) enabling boundary layer properties to penetrate well into the thermocline; mixing is represented through a polynomial profile whose coefficients are determined subject to several constraints;
4. the boundary-layer profile is made to agree with similarity theory of turbulence and is matched, in the asymptotic sense (function and derivative agree at the boundary), to the interior thus fixing the polynomial coefficients; matching allows for some fraction of the boundary layer mixing to affect the interior, and vice versa;
5. a "non-local" term $\hat{\gamma}$ or `ghat` which is independent of the vertical property gradient further enhances mixing where the water column is unstable

The scheme has been extensively compared to observations (see e.g. [LDDM97]) and is now common in many ocean models.

The current code originates in the NCAR NCOM 1-D code and was kindly provided by Bill Large and Jan Morzel. It has been adapted first to the MITgcm vector code and subsequently to the current parallel code. Adjustment were mainly in conjunction with WRAPPER requirements (domain decomposition and threading capability), to enable automatic differentiation of tangent linear and adjoint code via TAMC.

The following sections will describe the KPP package configuration and compiling ([sec:pkg:kpp:comp]), the settings and choices of runtime parameters ([sec:pkg:kpp:runtime]), more detailed description of equations to which these parameters relate ([sec:pkg:kpp:equations]), and key subroutines where they are used ([sec:pkg:kpp:flowchart]), and diagnostics output of KPP-derived diffusivities, viscosities and boundary-layer/mixed-layer depths ([sec:pkg:kpp:diagnostics]).

8.4.2.2 KPP configuration and compiling

As with all MITgcm packages, KPP can be turned on or off at compile time

- using the `packages.conf` file by adding `kpp` to it,
- or using `genmake2` adding `-enable=kpp` or `-disable=kpp` switches
- *Required packages and CPP options:* No additional packages are required, but the MITgcm kernel flag enabling the penetration of shortwave radiation below the surface layer needs to be set in `CPP_OPTIONS.h` as follows: `#define SHORTWAVE_HEATING`

(see Section [sec:buildingCode]).

Parts of the KPP code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in `KPP_OPTIONS.h`. Table Table 8.2 summarizes them.

Table 8.2: CPP flags for KPP

CPP option	Description
_KPP_RL	
FRUGAL_KPP	
KPP_SMOOTH_SHSQ	
KPP_SMOOTH_DVSQ	
KPP_SMOOTH_DENS	
KPP_SMOOTH_VISC	
KPP_SMOOTH_DIFF	
KPP_ESTIMATE_UREF	
INCLUDE_DIAGNOSTICS_INTERFACE_CODE	
KPP_GHAT	
EXCLUDE_KPP_SHEAR_MIX	

8.4.2.3 Run-time parameters

Run-time parameters are set in files `data.pkg` and `data.kpp` which are read in `kpp_readparms.F`. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) required MITgcm flags, (iii) package flags and parameters.

Enabling the package

The KPP package is switched on at runtime by setting `useKPP = .TRUE.` in `data.pkg`.

Required MITgcm flags

The following flags/parameters of the MITgcm dynamical kernel need to be set in conjunction with KPP:

`implicitViscosity = .TRUE.`	enable implicit vertical viscosity
`implicitDiffusion = .TRUE.`	enable implicit vertical diffusion

Package flags and parameters

Table 8.3 summarizes the runtime flags that are set in `data.pkg`, and their default values.

Table 8.3: Runtime flags for KPP

Flag/parameter	default	Description
I/O related parameters		
kpp_freq	`deltaTClock`	Recomputation frequency for KPP fields
kpp_dumpFreq	`dumpFreq`	Dump frequency of KPP field snapshots
kpp_taveFreq	`taveFreq`	Averaging and dump frequency of KPP fields
KPPmixingMaps	`.FALSE.`	include KPP diagnostic maps in STDOUT

Continued on next page

Table 8.3 – continued from previous page

Flag/parameter	default	Description
KPPwriteState	.FALSE.	write KPP state to file
KPP_ghatUseTotalDiffus	.FALSE.	if .T. compute non-local term using
		total vertical diffusivity
		if .F. use KPP vertical diffusivity
General KPP parameters		
minKPPhbl	delRc(1)	Minimum boundary layer depth
epsilon	0.1	nondimensional extent of the surface layer
vonk	0.4	von Karman constant
dB_dz	5.2E-5 s^{-2}	maximum dB/dz in mixed layer hMix
concs	98.96	
concv	1.8	
Boundary layer parameters (S/R bldepth)		
Ricr	0.3	critical bulk Richardson number
cekman	0.7	coefficient for Ekman depth
cmonob	1.0	coefficient for Monin-Obukhov depth
concv	1.8	ratio of interior to entrainment depth buoyancy frequency
hbf	1.0	fraction of depth to which absorbed solar radiation contributes to surface buoyancy forcing
Vtc		non-dim. coeff. for velocity scale of turbulant velocity shear (= function of concv,concs,epsilon,vonk,Ricr)
Boundary layer mixing parameters (S/R blmix)		
cstar	10.	proportionality coefficient for nonlocal transport
cg		non-dimensional coefficient for counter-gradient term (= function of cstar,vonk,concs,epsilon)
Interior mixing parameters (S/R Ri_iwmix)		
Riinfty	0.7	gradient Richardson number limit for shear instability
BVDQcon	-0.2E-4 s^{-2}	Brunt-Väisalä squared
difm0	0.005 m^2 s^{-1}	viscosity max. due to shear instability
difs0	0.005 m^2/s	tracer diffusivity max. due to shear instability
dift0	0.005 m^2/s	heat diffusivity max. due to shear instability
difmcon	0.1	viscosity due to convective instability
difscon	0.1	tracer diffusivity due to convective instability
diftcon	0.1	heat diffusivity due to convective instability

Continued on next page

Table 8.3 – continued from previous page

Flag/parameter	default	Description
Rrho0	not used	limit for double diffusive density ratio
dsfmax	not used	maximum diffusivity in case of salt fingering

8.4.2.4 Equations and key routines

We restrict ourselves to writing out only the essential equations that relate to main processes and parameters mentioned above. We closely follow the notation of [LMD94].

KPP_CALC:

Top-level routine.

KPP_MIX:

Intermediate-level routine

BLMIX: Mixing in the boundary layer

The vertical fluxes $\overline{wx}$ of momentum and tracer properties X is composed of a gradient-flux term (proportional to the vertical property divergence $\partial_z X$), and a "nonlocal" term γ_x that enhances the gradient-flux mixing coefficient K_x

$$\overline{wx}(d) \;=\; -K_x \left(\frac{\partial X}{\partial z} \;-\; \gamma_x \right)$$

- *Boundary layer mixing profile* It is expressed as the product of the boundary layer depth h, a depth-dependent turbulent velocity scale $w_x(\sigma)$ and a non-dimensional shape function $G(\sigma)$

  $$K_x(\sigma) \;=\; h\, w_x(\sigma)\, G(\sigma)$$

  with dimensionless vertical coordinate $\sigma = d/h$. For details of :math:‘ w_x(sigma)‘ and $G(\sigma)$ we refer to .

- *Nonlocal mixing term* The nonlocal transport term γ is nonzero only for tracers in unstable (convective) forcing conditions. Thus, depending on the stability parameter $\zeta = d/L$ (with depth d, Monin-Obukhov length scale L) it has the following form:

  $$\gamma_x \;=\; 0 \qquad\qquad \zeta \;\geq\; 0$$

  $$\left.\begin{aligned} \gamma_m \;&=\; 0 \\ \gamma_s \;&=\; C_s \tfrac{\overline{ws_0}}{w_s(\sigma)h} \\ \gamma_\theta \;&=\; C_s \tfrac{\overline{w\theta_0}+\overline{w\theta_R}}{w_s(\sigma)h} \end{aligned}\right\} \quad \zeta \;<\; 0$$

In practice, the routine peforms the following tasks:

1. compute velocity scales at hbl
2. find the interior viscosities and derivatives at hbl

3. compute turbulent velocity scales on the interfaces
4. compute the dimensionless shape functions at the interfaces
5. compute boundary layer diffusivities at the interfaces
6. compute nonlocal transport term
7. find diffusivities at kbl-1 grid level

RI_IWMIX: Mixing in the interior

Compute interior viscosity and diffusivity coefficients due to

- shear instability (dependent on a local gradient Richardson number),
- to background internal wave activity, and
- to static instability (local Richardson number < 0).

TO BE CONTINUED.

BLDEPTH: Boundary layer depth calculation:

The oceanic planetary boundary layer depth, `hbl`, is determined as the shallowest depth where the bulk Richardson number is equal to the critical value, `Ricr`.

Bulk Richardson numbers are evaluated by computing velocity and buoyancy differences between values at zgrid(kl) < 0 and surface reference values. In this configuration, the reference values are equal to the values in the surface layer. When using a very fine vertical grid, these values should be computed as the vertical average of velocity and buoyancy from the surface down to epsilon*zgrid(kl).

When the bulk Richardson number at k exceeds Ricr, hbl is linearly interpolated between grid levels zgrid(k) and zgrid(k-1).

The water column and the surface forcing are diagnosed for stable/ustable forcing conditions, and where hbl is relative to grid points (caseA), so that conditional branches can be avoided in later subroutines.

TO BE CONTINUED.

KPP_CALC_DIFF_T/_S, KPP_CALC_VISC:

Add contribution to net diffusivity/viscosity from KPP diffusivity/viscosity.

TO BE CONTINUED.

KPP_TRANSPORT_T/_S/_PTR:

Add non local KPP transport term (ghat) to diffusive temperature/salinity/passive tracer flux. The nonlocal transport term is nonzero only for scalars in unstable (convective) forcing conditions.

TO BE CONTINUED.

Implicit time integration

TO BE CONTINUED.

Penetration of shortwave radiation

TO BE CONTINUED.

8.4.2.5 Flow chart

```
C     !CALLING SEQUENCE:
c ...
c  kpp_calc (TOP LEVEL ROUTINE)
c  |
c  |-- statekpp: o compute all EOS/density-related arrays
c  |             o uses S/R FIND_ALPHA, FIND_BETA, FIND_RHO
c  |
c  |-- kppmix
c  |   |--- ri_iwmix (compute interior mixing coefficients due to constant
c  |   |              internal wave activity, static instability,
c  |   |              and local shear instability).
c  |   |
c  |   |--- bldepth (diagnose boundary layer depth)
c  |   |
c  |   |--- blmix (compute boundary layer diffusivities)
c  |   |
c  |   |--- enhance (enhance diffusivity at interface kbl - 1)
c  |   o
c  |
c  |-- swfrac
c  o
```

8.4.2.6 KPP diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized here:

```
------------------------------------------------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------
 KPPviscA| 23 |SM  |m^2/s           |KPP vertical eddy viscosity coefficient
 KPPdiffS| 23 |SM  |m^2/s           |Vertical diffusion coefficient for salt & tracers
 KPPdiffT| 23 |SM  |m^2/s           |Vertical diffusion coefficient for heat
 KPPghat | 23 |SM  |s/m^2           |Nonlocal transport coefficient
 KPPhbl  |  1 |SM  |m               |KPP boundary layer depth, bulk Ri criterion
 KPPmld  |  1 |SM  |m               |Mixed layer depth, dT=.8degC density criterion
 KPPfrac |  1 |SM  |                |Short-wave flux fraction penetrating mixing layer
```

8.4.2.7 Reference experiments

lab_sea:

natl_box:

8.4.2.8 References

8.4.2.9 Experiments and tutorials that use kpp

- Labrador Sea experiment, in lab_sea verification directory

8.4.3 GGL90: a TKE vertical mixing scheme

(in directory: *pkg/ggl90/*)

8.4.3.1 Key subroutines, parameters and files

see [GGregorisL90]

8.4.3.2 Experiments and tutorials that use GGL90

- Vertical mixing verification experiment (`vermix/input.ggl90`)

8.4.4 OPPS: Ocean Penetrative Plume Scheme

(in directory: *pkg/opps/*)

8.4.4.1 Key subroutines, parameters and files

See [PR97]

8.4.4.2 Experiments and tutorials that use OPPS

- Vertical mixing verification experiment (*vermix/input.opps*)

8.4.5 KL10: Vertical Mixing Due to Breaking Internal Waves

(in directory: *pkg/kl10/*)

Authors: Jody M. Klymak

8.4.5.1 Introduction

The [KL10] parameterization for breaking internal waves is meant to represent mixing in the ocean "interior" due to convective instability. Many mixing schemes in the presence of unstable stratification simply turn on an arbitrarily large diffusivity and viscosity in the overturning region. This assumes the fluid completely mixes, which is probably not a terrible assumption, but it also makes estimating the turbulence dissipation rate in the overturning region meaningless.

The KL10 scheme overcomes this limitation by estimating the viscosity and diffusivity from a combination of the Ozmidov relation and the Osborn relation, assuming a turbulent Prandtl number of one. The Ozmidov relation says that outer scale of turbulence in an overturn will scale with the strength of the turbulence ϵ, and the stratification N, as

$$L_O^2 \approx \epsilon N^{-3}. \tag{8.1}$$

The Osborn relation relates the strength of the dissipation to the vertical diffusivity as

$$K_v = \Gamma \epsilon N^{-2},$$

where $\Gamma \approx 0.2$ is the mixing ratio of buoyancy flux to thermal dissipation due to the turbulence. Combining the two gives us

$$K_v \approx \Gamma L_O^2 N.$$

The ocean turbulence community often approximates the Ozmidov scale by the root-mean-square of the Thorpe displacement, δ_z, in an overturn [Tho77]. The Thorpe displacement is the distance one would have to move a water parcel for the water column to be stable, and is readily measured in a measured profile by sorting the profile and tracking how far each parcel moves during the sorting procedure. This method gives an imperfect estimate of the turbulence, but it has been found to agree on average over a large range of overturns [WG94][SG94][Mou96].

The algorithm coded here is a slight simplification of the usual Thorpe method for estimating turbulence in overturning regions. Usually, overturns are identified and N is averaged over the overturn. Here, instead we estimate

$$K_v(z) \approx \Gamma \delta_z^2 \, N_s(z).$$

where $N_s(z)$ is the local sorted stratification. This saves complexity in the code and adds a slight inaccuracy, but we don't believe is biased.

We assume a turbulent Prandtl number of 1, so $A_v = K_v$.

We also calculate and output a turbulent dissipation from this scheme. We do not simply evaluate the overturns for ϵ using ([eq:pkg:kl10:Lo]). Instead we compute the vertical shear terms that the viscosity is acting on:

$$\epsilon_v = A_v \left(\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2 \right).$$

There are straightforward caveats to this approach, covered in [KL10].

- If your resolution is too low to resolve the breaking internal waves, you won't have any turbulence.
- If the model resolution is too high, the estimates of ϵ_v will start to be exaggerated, particularly if the run in non-hydrostatic. That is because there will be significant shear at small scales that represents the turbulence being parameterized in the scheme. At very high resolutions direct numerical simulation or more sophisticated large-eddy schemes should be used.
- We find that grid cells of approximately 10 to 1 aspect ratio are a good rule of thumb for achieving good results are usual oceanic scales. For a site like the Hawaiian Ridge, and Luzon Strait, this means 10-m vertical resolusion and approximately 100-m horizontal. The 10-m resolution can be relaxed if the stratification drops, and we often WKB-stretch the grid spacing with depth.
- The dissipation estimate is useful for pinpoiting the location of turbulence, but again, is grid size dependent to some extent, and should be treated with a grain of salt. It will also not include any numerical dissipation such as you may find with higher order advection schemes.

8.4.5.2 KL10 configuration and compiling

As with all MITgcm packages, KL10 can be turned on or off at compile time

- using the `packages.conf` file by adding `kl10` to it,
- or using `genmake2` adding `-enable=kl10` or `-disable=kl10` switches
- *Required packages and CPP options:* No additional packages are required.

(see Section [sec:buildingCode]).

KL10 has no compile-time options (`KL10_OPTIONS.h` is empty).

8.4.5.3 Run-time parameters

Run-time parameters are set in files `data.pkg` and `data.kl10` which are read in `kl10_readparms.F`. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) required MITgcm flags, (iii) package flags and parameters.

Enabling the package

The KL10 package is switched on at runtime by setting `useKL10 = .TRUE.` in `data.pkg`.

Required MITgcm flags

The following flags/parameters of the MITgcm dynamical kernel need to be set in conjunction with KL10:

`implicitViscosity = .TRUE.`	enable implicit vertical viscosity
`implicitDiffusion = .TRUE.`	enable implicit vertical diffusion

Package flags and parameters

Table 8.4 summarizes the runtime flags that are set in `data.kl10`, and their default values.

Table 8.4: KL10 runtime parameters.

Flag/parameter	default	Description
KLviscMax	300 m^2 s^{-1}	Maximum viscosity the scheme will ever give (useful for stability)
KLdumpFreq	`dumpFreq`	Dump frequency of KL10 field snapshots
KLtaveFreq	`taveFreq`	Averaging and dump frequency of KL10 fields
KLwriteState	`.FALSE.`	write KL10 state to file

8.4.5.4 Equations and key routines

KL10_CALC:

Top-level routine. Calculates viscosity and diffusivity on the grid cell centers. Note that the runtime parameters `viscAz` and `diffKzT` act as minimum viscosity and diffusivities. So if there are no overturns (or they are weak) then these will be returned.

KL10_CALC_VISC:

Calculates viscosity on the W and S grid faces for U and V respectively.

KL10_CALC_DIFF:

Calculates the added diffusion from KL10.

8.4.5.5 KL10 diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized here:

```
------------------------------------------------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------
 KLviscAr| Nr |SM  |m^2/s           |KL10 vertical eddy viscosity coefficient
 KLdiffKr| Nr |SM  |m^2/s           |Vertical diffusion coefficient for salt,␣
↪temperature, & tracers
 KLeps   | Nr |SM  |m^3/s^3         |Turbulence dissipation estimate.
```

8.4.5.6 References

Klymak and Legg, 2010, *Oc. Modell.*.

8.4.5.7 Experiments and tutorials that use KL10

- Modified Internal Wave experiment, in internal_wave verification directory

8.4.6 BULK_FORCE: Bulk Formula Package

author: Stephanie Dutkiewicz

Instead of forcing the model with heat and fresh water flux data, this package calculates these fluxes using the changing sea surface temperature. We need to read in some atmospheric data: **air temperature, air humidity, down shortwave radiation, down longwave radiation, precipitation, wind speed**. The current setup also reads in **wind stress**, but this can be changed so that the stresses are calculated from the wind speed.

The current setup requires that there is the thermodynamic-seaice package (*pkg/thsice*, also refered below as seaice) is also used. It would be useful though to have it also setup to run with some very simple parametrization of the sea ice.

The heat and fresh water fluxes are calculated in *bulkf_forcing.F* called from *forward_step.F*. These fluxes are used over open water, fluxes over seaice are recalculated in the sea-ice package. Before the call to *bulkf_forcing.F* we call *bulkf_fields_load.F* to find the current atmospheric conditions. The only other changes to the model code come from the initializing and writing diagnostics of these fluxes.

8.4.6.1 subroutine BULKF_FIELDS_LOAD

Here we find the atmospheric data needed for the bulk formula calculations. These are read in at periodic intervals and values are interpolated to the current time. The data file names come from **data.blk**. The values that can be read in are: air temperature, air humidity, precipitation, down solar radiation, down long wave radiation, zonal and meridional wind speeds, total wind speed, net heat flux, net freshwater forcing, cloud cover, snow fall, zonal and meridional wind stresses, and SST and SSS used for relaxation terms. Not all these files are necessary or used. For instance cloud cover and snow fall are not used in the current bulk formula calculation. If total wind speed is not supplied, wind speed is calculate from the zonal and meridional components. If wind stresses are not read in, then the stresses are calculated from the wind speed. Net heat flux and net freshwater can be read in and used over open ocean instead of the bulk formula calculations (but over seaice the bulkf formula is always used). This is "hardwired" into *bulkf_forcing* and the "ch" in the variable names suggests that this is "cheating". SST and SSS need to be read in if there is any relaxation used.

8.4.6.2 subroutine BULKF_FORCING

In *bulkf_forcing.F*, we calculate heat and fresh water fluxes (and wind stress, if necessary) for each grid cell. First we determine if the grid cell is open water or seaice and this information is carried by **iceornot**. There is a provision here for a different designation if there is snow cover (but currently this does not make any difference). We then call *bulkf_formula_lanl.F* which provides values for: up long wave radiation, latent and sensible heat fluxes, the derivative of these three with respect to surface temperature, wind stress, evaporation. Net long wave radiation is calculated from the combination of the down long wave read in and the up long wave calculated.

We then find the albedo of the surface - with a call to *sfc_albedo* if there is sea-ice (see the seaice package for information on the subroutine). If the grid cell is open ocean the albedo is set as 0.1. Note that this is a parameter that can be used to tune the results. The net short wave radiation is then the down shortwave radiation minus the amount reflected.

If the wind stress needed to be calculated in *bulkf_formula_lanl.F*, it was calculated to grid cell center points, so in *bulkf_forcing.F* we regrid to **u** and **v** points. We let the model know if it has read in stresses or calculated stresses by the switch **readwindstress** which is can be set in data.blk, and defaults to **.TRUE.**.

We then calculate **Qnet** and **EmPmR** that will be used as the fluxes over the open ocean. There is a provision for using runoff. If we are "cheating" and using observed fluxes over the open ocean, then there is a provision here to use read in **Qnet** and **EmPmR**.

The final call is to calculate averages of the terms found in this subroutine.

8.4.6.3 subroutine BULKF_FORMULA_LANL

This is the main program of the package where the heat fluxes and freshwater fluxes over ice and open water are calculated. Note that this subroutine is also called from the seaice package during the iterations to find the ice surface temperature.

Latent heat (L) used in this subroutine depends on the state of the surface: vaporization for open water, fusion and vaporization for ice surfaces. Air temperature is converted from Celsius to Kelvin. If there is no wind speed (u_s) given, then the wind speed is calculated from the zonal and meridional components.

We calculate the virtual temperature:

$$T_o = T_{air}(1 + \gamma q_{air})$$

where T_{air} is the air temperature at h_T, q_{air} is humidity at h_q and γ is a constant.

The saturated vapor pressure is calculate (QQ ref):

$$q_{sat} = \frac{a}{p_o} e^{L(b - \frac{c}{T_{srf}})}$$

where a, b, c are constants, T_{srf} is surface temperature and p_o is the surface pressure.

The two values crucial for the bulk formula calculations are the difference between air at sea surface and sea surface temperature:

$$\Delta T = T_{air} - T_{srf} + \alpha h_T$$

where α is adiabatic lapse rate and h_T is the height where the air temperature was taken; and the difference between the air humidity and the saturated humidity

$$\Delta q = q_{air} - q_{sat}.$$

We then calculate the turbulent exchange coefficients following Bryan et al (1996) and the numerical scheme of Hunke and Lipscombe (1998). We estimate initial values for the exchange coefficients, c_u, c_T and c_q as

$$\frac{\kappa}{ln(z_{ref}/z_{rou})}$$

where κ is the Von Karman constant, z_{ref} is a reference height and z_{rou} is a roughness length scale which could be a function of type of surface, but is here set as a constant. Turbulent scales are:

$$\begin{aligned} u^* = &\quad c_u u_s \\ T^* = &\quad c_T \Delta T \\ q^* = &\quad c_q \Delta q \end{aligned}$$

We find the "integrated flux profile" for momentum and stability if there are stable QQ conditions ($\Upsilon > 0$) :

$$\psi_m = \psi_s = -5\Upsilon$$

and for unstable QQ conditions ($\Upsilon < 0$):

$$\begin{aligned} \psi_m = &\quad 2ln(0.5(1+\chi)) + ln(0.5(1+\chi^2)) - 2\tan^{-1}\chi + \pi/2 \\ \psi_s = &\quad 2ln(0.5(1+\chi^2)) \end{aligned}$$

where

$$\Upsilon = \frac{\kappa g z_{ref}}{u^{*2}}\left(\frac{T^*}{T_o} + \frac{q^*}{1/\gamma + q_a}\right)$$

and $\chi = (1 - 16\Upsilon)^{1/2}$.

The coefficients are updated through 5 iterations as:

$$\begin{aligned} c_u = &\quad \frac{\hat{c_u}}{1 + \hat{c_u}(\lambda - \psi_m)/\kappa} \\ c_T = &\quad \frac{\hat{c_T}}{1 + \hat{c_T}(\lambda - \psi_s)/\kappa} \\ c_q = &\quad c'_T \end{aligned}$$

where $\lambda = ln(h_T/z_{ref})$.

We can then find the bulk formula heat fluxes:

Sensible heat flux:

$$Q_s = \rho_{air} c_{p_{air}} u_s c_u c_T \Delta T$$

Latent heat flux:

$$Q_l = \rho_{air} L u_s c_u c_q \Delta q$$

Up long wave radiation

$$Q_{lw}^{up} = \epsilon \sigma T_{srf}^4$$

where ϵ is emissivity (which can be different for open ocean, ice and snow), σ is Stefan-Boltzman constant.

We calculate the derivatives of the three above functions with respect to surface temperature

$$\begin{aligned} \frac{dQ_s}{d_T} = &\quad \rho_{air} c_{p_{air}} u_s c_u c_T \\ \frac{dQ_l}{d_T} = &\quad \frac{\rho_{air} L^2 u_s c_u c_q c}{T_{srf}^2} \\ \frac{dQ_{|lw}^{up}}{d_T} = &\quad 4\epsilon\sigma t_{srf}^3 \end{aligned}$$

And total derivative $\frac{dQ_o}{dT} = \frac{dQ_s}{dT} + \frac{dQ_l}{dT} + \frac{dQ_{lw}^{up}}{dT}$.

If we do not read in the wind stress, it is calculated here.

8.4.6.4 Initializing subroutines

`bulkf_init.F`: Set bulkf variables to zero.

`bulkf_readparms.F`: Reads **data.blk**

8.4.6.5 Diagnostic subroutines

`bulkf_ave.F`: Keeps track of means of the bulkf variables

`bulkf_diags.F`: Finds averages and writes out diagnostics

8.4.6.6 Common Blocks

`BULKF.h`: BULKF Variables, data file names, and logicals **readwindstress** and **readsurface**

`BULKF_DIAGS.h`: matrices for diagnostics: averages of fields from *bulkf_diags.F*

`BULKF_ICE_CONSTANTS.h`: all the parameters needed by the ice model and in the bulkf formula calculations.

8.4.6.7 Input file DATA.ICE

We read in the file names of atmospheric data used in the bulk formula calculations. Here we can also set the logicals: **readwindstress** if we read in the wind stress rather than calculate it from the wind speed; and **readsurface** to read in the surface temperature and salinity if these will be used as part of a relaxing term.

8.4.6.8 Important Notes

1. heat fluxes have different signs in the ocean and ice models.
2. **StartIceModel** must be changed in **data.ice**: 1 (if starting from no ice), 0 (if using pickup.ic file).

8.4.6.9 References

Bryan F.O., B.G Kauffman, W.G. Large, P.R. Gent, 1996: The NCAR CSM flux coupler. Technical note TN-425+STR, NCAR.

Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual. LACC-98-16v.2. (note: this documentation is no longer available as CICE has progressed to a very different version 3)

8.4.6.10 Experiments and tutorials that use bulk_force

- Global ocean experiment in global_ocean.cs32x15 verification directory, input from input.thsice directory.

8.4.7 EXF: The external forcing package

Authors: Patrick Heimbach and Dimitris Menemenlis

8.4.7.1 Introduction

The external forcing package, in conjunction with the calendar package (cal), enables the handling of real-time (or "model-time") forcing fields of differing temporal forcing patterns. It comprises climatological restoring and relaxation. Bulk formulae are implemented to convert atmospheric fields to surface fluxes. An interpolation routine provides on-the-fly interpolation of forcing fields an arbitrary grid onto the model grid.

CPP options enable or disable different aspects of the package (Section [sec:pkg:exf:config]). Runtime options, flags, filenames and field-related dates/times are set in `data.exf` (Section [sec:pkg:exf:runtime]). A description of key subroutines is given in Section [sec:pkg:exf:subroutines]. Input fields, units and sign conventions are summarized in Section [sec:pkg:exf:fields:sub:*u*nits], and available diagnostics output is listed in Section [sec:pkg:exf:diagnostics].

8.4.7.2 EXF configuration, compiling & running

Compile-time options

As with all MITgcm packages, EXF can be turned on or off at compile time

- using the `packages.conf` file by adding `exf` to it,
- or using `genmake2` adding `-enable=exf` or `-disable=exf` switches
- *required packages and CPP options*: EXF requires the calendar package `cal` to be enabled; no additional CPP options are required.

(see Section [sec:buildingCode]).

Parts of the EXF code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in either `EXF_OPTIONS.h` or in `ECCO_CPPOPTIONS.h`. Table 8.5 summarizes these options.

Table 8.5: EXF CPP options

CPP option	Description
`EXF_VERBOSE`	verbose mode (recommended only for testing)
`ALLOW_ATM_TEMP`	compute heat/freshwater fluxes from atmos. state input
`ALLOW_ATM_WIND`	compute wind stress from wind speed input
`ALLOW_BULKFORMULAE`	is used if `ALLOW_ATM_TEMP` or `ALLOW_ATM_WIND` is enabled
`EXF_READ_EVAP`	read evaporation instead of computing it
`ALLOW_RUNOFF`	read time-constant river/glacier run-off field
`ALLOW_DOWNWARD_RADIATION`	compute net from downward or downward from net radiation
`USE_EXF_INTERPOLATION`	enable on-the-fly bilinear or bicubic interpolation of input fields
used in conjunction with relaxation to prescribed (climatological) fields	
`ALLOW_CLIMSST_RELAXATION`	relaxation to 2-D SST climatology
`ALLOW_CLIMSSS_RELAXATION`	relaxation to 2-D SSS climatology
these are set outside of EXF in `CPP_OPTIONS.h`	
`SHORTWAVE_HEATING`	enable shortwave radiation
`ATMOSPHERIC_LOADING`	enable surface pressure forcing

8.4.7.3 Run-time parameters

Run-time parameters are set in files `data.pkg` and `data.exf` which is read in `exf_readparms.F`. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) general flags and param-

eters, and (iii) attributes for each forcing and climatological field.

Enabling the package

A package is switched on/off at runtime by setting (e.g. for EXF) `useEXF = .TRUE.` in `data.pkg`.

General flags and parameters

Table 8.6: EXF runtime options

Flag/parameter	default	Description
useExfCheckRange	`.TRUE.`	check range of input fields and stop if out of range
useExfYearlyFields	`.FALSE.`	append current year postfix of form `_YYYY` on filename
twoDigitYear	`.FALSE.`	instead of appending `_YYYY` append `YY`
repeatPeriod	0.0	> 0: cycle through all input fields at the same period (in seconds)
		= 0: use period assigned to each field
exf_offset_atemp	0.0	set to 273.16 to convert from deg. Kelvin (assumed input) to Celsius
windstressmax	2.0	max. allowed wind stress N m^{-2}
exf_albedo	0.1	surface albedo used to compute downward vs. net radiative fluxes
climtempfreeze	-1.9	???
ocean_emissivity		longwave ocean-surface emissivity
ice_emissivity		longwave seaice emissivity
snow_emissivity		longwave snow emissivity
exf_iceCd	1.63E-3	drag coefficient over sea-ice
exf_iceCe	1.63E-3	evaporation transfer coeff. over sea-ice
exf_iceCh	1.63E-3	sensible heat transfer coeff. over sea-ice
exf_scal_BulkCdn	1.0	overall scaling of neutral drag coeff.
useStabilityFct_overIce	`.FALSE.`	compute turbulent transfer coeff. over sea-ice
readStressOnAgrid	`.FALSE.`	read wind-streess located on model-grid, A-grid point
readStressOnCgrid	`.FALSE.`	read wind-streess located on model-grid, C-grid point
useRelativeWind	`.FALSE.`	subtract [U/V]VEL or [U/VICE from U/V]WIND before computing [U/V]STRESS
zref	10.0	reference height
hu	10.0	height of mean wind
ht	2.0	height of mean temperature and rel. humidity
umin	0.5	minimum absolute wind speed for computing Cd
atmrho	1.2	mean atmospheric density [kg/m^3]
atmcp	1005.0	mean atmospheric specific heat [J/kg/K]
cdrag_[n]	???	n = 1,2,3; parameters for drag coeff. function
cstanton_[n]	???	n = 1,2; parameters for Stanton number function
cdalton	???	parameter for Dalton number function
flamb	2500000.0	latent heat of evaporation [J/kg]
flami	334000.0	latent heat of melting of pure ice [J/kg]
zolmin	-100.0	minimum stability parameter
cvapor_fac	640380.0	
cvapor_exp	5107.4	
cvapor_fac_ice	11637800.0	
cvapor_fac_ice	5897.8	
humid_fac	0.606	parameter for virtual temperature calculation
gamma_blk	0.010	adiabatic lapse rate
saltsat	0.980	reduction of saturation vapor pressure over salt-water

Continued on next page

Table 8.6 – continued from previous page

Flag/parameter	default	Description
psim_fac	5.0	
exf_monFreq	monitorFreq	output frequency [s]
exf_iprec	32	precision of input fields (32-bit or 64-bit)
exf_yftype	'RL'	precision of arrays ('RL' vs. 'RS')

Field attributes

All EXF fields are listed in Section [sec:pkg:exf:fields:sub:*u*nits]. Each field has a number of attributes which can be customized. They are summarized in Table [tab:pkg:exf:runtime:sub:*a*ttributes]. To obtain an attribute for a specific field, e.g. `uwind` prepend the field name to the listed attribute, e.g. for attribute `period` this yields `uwindperiod`:

$$\begin{array}{cccccc} & \texttt{field} & \& & \texttt{attribute} & \longrightarrow & \texttt{parameter} \\ \text{e.g.} & \text{uwind} & \& & \text{period} & \longrightarrow & \text{uwindperiod} \end{array}$$

Table 8.7: EXF runtime attributes Note there is one exception for the default of `atempconst` = celsius2K = 273.16

attribute	Default	Description
field `file`	' '	filename; if left empty no file will be read; `const` will be used instead
field `const`	0.0	constant that will be used if no file is read
field `startdate1`	0.0	format: `YYYYMMDD`; start year (YYYY), month (MM), day (YY)
		of field to determine record number
field `startdate2`	0.0	format: `HHMMSS`; start hour (HH), minute (MM), second(SS)
		of field to determine record number
field `period`	0.0	interval in seconds between two records
`exf_inscal_`*field*		optional rescaling of input fields to comply with EXF units
`exf_outscal_`*field*		optional rescaling of EXF fields when mapped onto MITgcm fields
used in conjunction with `EXF_USE_INTERPOLATION`		
field `_lon0`	`xgOrigin+delX/2`	starting longitude of input
field `_lon_inc`	`delX`	increment in longitude of input
field `_lat0`	`ygOrigin+delY/2`	starting latitude of input
field `_lat_inc`	`delY`	increment in latitude of input
field `_nlon`	`Nx`	number of grid points in longitude of input
field `_nlat`	`Ny`	number of grid points in longitude of input

Example configuration

The following block is taken from the `data.exf` file of the verification experiment `global_with_exf/`. It defines attributes for the heat flux variable `hflux`:

```
hfluxfile       = 'ncep_qnet.bin',
hfluxstartdate1 = 19920101,
hfluxstartdate2 = 000000,
hfluxperiod     = 2592000.0,
hflux_lon0      = 2
```

(continues on next page)

(continued from previous page)

```
hflux_lon_inc   = 4
hflux_lat0      = -78
hflux_lat_inc   = 39*4
hflux_nlon      = 90
hflux_nlat      = 40
```

EXF will read a file of name 'ncep_qnet.bin'. Its first record represents January 1st, 1992 at 00:00 UTC. Next record is 2592000 seconds (or 30 days) later. Note that the first record read and used by the EXF package corresponds to the value 'startDate1' set in data.cal. Therefore if you want to start the EXF forcing from later in the 'ncep_qnet.bin' file, it suffices to specify startDate1 in data.cal as a date later than 19920101 (for example, startDate1 = 19940101, for starting January 1st, 1994). For this to work, 'ncep_qnet.bin' must have at least 2 years of data because in this configuration EXF will read 2 years into the file to find the 1994 starting value. Interpolation on-the-fly is used (in the present case trivially on the same grid, but included nevertheless for illustration), and input field grid starting coordinates and increments are supplied as well.

8.4.7.4 EXF bulk formulae

T.B.D. (cross-ref. to parameter list table)

8.4.7.5 EXF input fields and units

The following list is taken from the header file `EXF_FIELDS.h`. It comprises all EXF input fields.

Output fields which EXF provides to the MITgcm are fields **fu**, **fv**, **Qnet**, **Qsw**, **EmPmR**, and **pload**. They are defined in `FFIELDS.h`.

```
c--------------------------------------------------------------------
c               |
c     field     :: Description
c               |
c--------------------------------------------------------------------
c     ustress   :: Zonal surface wind stress in N/m^2
c               |  > 0 for increase in uVel, which is west to
c               |      east for cartesian and spherical polar grids
c               |  Typical range: -0.5 < ustress < 0.5
c               |  Southwest C-grid U point
c               |  Input field
c--------------------------------------------------------------------
c     vstress   :: Meridional surface wind stress in N/m^2
c               |  > 0 for increase in vVel, which is south to
c               |      north for cartesian and spherical polar grids
c               |  Typical range: -0.5 < vstress < 0.5
c               |  Southwest C-grid V point
c               |  Input field
c--------------------------------------------------------------------
c     hs        :: sensible heat flux into ocean in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c--------------------------------------------------------------------
c     hl        :: latent   heat flux into ocean in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c--------------------------------------------------------------------
c     hflux     :: Net upward surface heat flux in W/m^2
c               |  (including shortwave)
c               |  hflux = latent + sensible + lwflux + swflux
```

(continues on next page)

(continued from previous page)

```
c               |  > 0 for decrease in theta (ocean cooling)
c               |  Typical range: -250 < hflux < 600
c               |  Southwest C-grid tracer point
c               |  Input field
c--------------------------------------------------------------------
c     sflux     :: Net upward freshwater flux in m/s
c               |  sflux = evap - precip - runoff
c               |  > 0 for increase in salt (ocean salinity)
c               |  Typical range: -1e-7 < sflux < 1e-7
c               |  Southwest C-grid tracer point
c               |  Input field
c--------------------------------------------------------------------
c     swflux    :: Net upward shortwave radiation in W/m^2
c               |  swflux = - ( swdown - ice and snow absorption - reflected )
c               |  > 0 for decrease in theta (ocean cooling)
c               |  Typical range: -350 < swflux < 0
c               |  Southwest C-grid tracer point
c               |  Input field
c--------------------------------------------------------------------
c     uwind     :: Surface (10-m) zonal wind velocity in m/s
c               |  > 0 for increase in uVel, which is west to
c               |      east for cartesian and spherical polar grids
c               |  Typical range: -10 < uwind < 10
c               |  Southwest C-grid U point
c               |  Input or input/output field
c--------------------------------------------------------------------
c     vwind     :: Surface (10-m) meridional wind velocity in m/s
c               |  > 0 for increase in vVel, which is south to
c               |      north for cartesian and spherical polar grids
c               |  Typical range: -10 < vwind < 10
c               |  Southwest C-grid V point
c               |  Input or input/output field
c--------------------------------------------------------------------
c     wspeed    :: Surface (10-m) wind speed in m/s
c               |  >= 0 sqrt(u^2+v^2)
c               |  Typical range: 0 < wspeed < 10
c               |  Input or input/output field
c--------------------------------------------------------------------
c     atemp     :: Surface (2-m) air temperature in deg K
c               |  Typical range: 200 < atemp < 300
c               |  Southwest C-grid tracer point
c               |  Input or input/output field
c--------------------------------------------------------------------
c     aqh       :: Surface (2m) specific humidity in kg/kg
c               |  Typical range: 0 < aqh < 0.02
c               |  Southwest C-grid tracer point
c               |  Input or input/output field
c--------------------------------------------------------------------
c     lwflux    :: Net upward longwave radiation in W/m^2
c               |  lwflux = - ( lwdown - ice and snow absorption - emitted )
c               |  > 0 for decrease in theta (ocean cooling)
c               |  Typical range: -20 < lwflux < 170
c               |  Southwest C-grid tracer point
c               |  Input field
c--------------------------------------------------------------------
c     evap      :: Evaporation in m/s
c               |  > 0 for increase in salt (ocean salinity)
```

(continues on next page)

(continued from previous page)

```
c               |  Typical range: 0 < evap < 2.5e-7
c               |  Southwest C-grid tracer point
c               |  Input, input/output, or output field
c--------------------------------------------------------------------
c     precip    :: Precipitation in m/s
c               |  > 0 for decrease in salt (ocean salinity)
c               |  Typical range: 0 < precip < 5e-7
c               |  Southwest C-grid tracer point
c               |  Input or input/output field
c--------------------------------------------------------------------
c    snowprecip :: snow in m/s
c               |  > 0 for decrease in salt (ocean salinity)
c               |  Typical range: 0 < precip < 5e-7
c               |  Input or input/output field
c--------------------------------------------------------------------
c     runoff    :: River and glacier runoff in m/s
c               |  > 0 for decrease in salt (ocean salinity)
c               |  Typical range: 0 < runoff < ????
c               |  Southwest C-grid tracer point
c               |  Input or input/output field
c               |  !!! WATCH OUT: Default exf_inscal_runoff !!!
c               |  !!! in exf_readparms.F is not 1.0        !!!
c--------------------------------------------------------------------
c     swdown    :: Downward shortwave radiation in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c               |  Typical range: 0 < swdown < 450
c               |  Southwest C-grid tracer point
c               |  Input/output field
c--------------------------------------------------------------------
c     lwdown    :: Downward longwave radiation in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c               |  Typical range: 50 < lwdown < 450
c               |  Southwest C-grid tracer point
c               |  Input/output field
c--------------------------------------------------------------------
c     apressure :: Atmospheric pressure field in N/m^2
c               |  > 0 for ????
c               |  Typical range: ???? < apressure < ????
c               |  Southwest C-grid tracer point
c               |  Input field
c--------------------------------------------------------------------
```

8.4.7.6 Key subroutines

Top-level routine: `exf_getforcing.F`

```
C     !CALLING SEQUENCE:
c ...
c  exf_getforcing (TOP LEVEL ROUTINE)
c  |
c  |-- exf_getclim (get climatological fields used e.g. for relax.)
c  |   |--- exf_set_climsst  (relax. to 2-D SST field)
c  |   |--- exf_set_climsss  (relax. to 2-D SSS field)
c  |   o
c  |
```

(continues on next page)

(continued from previous page)

```
c  |-- exf_getffields <- this one does almost everything
c  |   |   1. reads in fields, either flux or atmos. state,
c  |   |      depending on CPP options (for each variable two fields
c  |   |      consecutive in time are read in and interpolated onto
c  |   |      current time step).
c  |   |   2. If forcing is atmos. state and control is atmos. state,
c  |   |      then the control variable anomalies are read here via ctrl_get_gen
c  |   |      (atemp, aqh, precip, swflux, swdown, uwind, vwind).
c  |   |      If forcing and control are fluxes, then
c  |   |      controls are added later.
c  |   o
c  |
c  |-- exf_radiation
c  |   |    Compute net or downwelling radiative fluxes via
c  |   |    Stefan-Boltzmann law in case only one is known.
c  |   o
c  |-- exf_wind
c  |   |   Computes wind speed and stresses, if required.
c  |   o
c  |
c  |-- exf_bulkformulae
c  |   |   Compute air-sea buoyancy fluxes from
c  |   |   atmospheric state following Large and Pond, JPO, 1981/82
c  |   o
c  |
c  |-- < hflux is sum of sensible, latent, longwave rad. >
c  |-- < sflux is sum of evap. minus precip. minus runoff  >
c  |
c  |-- exf_getsurfacefluxes
c  |   If forcing and control is flux, then the
c  |   control vector anomalies are read here via ctrl_get_gen
c  |   (hflux, sflux, ustress, vstress)
c  |
c  |-- < update tile edges here >
c  |
c  |-- exf_check_range
c  |   |   Check whether read fields are within assumed range
c  |   |   (may capture mismatches in units)
c  |   o
c  |
c  |-- < add shortwave to hflux for diagnostics >
c  |
c  |-- exf_diagnostics_fill
c  |   |   Do EXF-related diagnostics output here.
c  |   o
c  |
c  |-- exf_mapfields
c  |   |   Forcing fields from exf package are mapped onto
c  |   |   mitgcm forcing arrays.
c  |   |   Mapping enables a runtime rescaling of fields
c  |   o
C  o
```

Radiation calculation: `exf_radiation.F`

Wind speed and stress calculation: `exf_wind.F`

Bulk formula: `exf_bulkformulae.F`

Generic I/O: `exf_set_gen.F`

Interpolation: `exf_interp.F`

Header routines

8.4.7.7 EXF diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized below.

```
---------+----+----+----------------+-----------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
---------+----+----+----------------+-----------------
 EXFhs   |  1 | SM | W/m^2          | Sensible heat flux into ocean, >0 increases␣
↪theta
 EXFhl   |  1 | SM | W/m^2          | Latent heat flux into ocean, >0 increases theta
 EXFlwnet|  1 | SM | W/m^2          | Net upward longwave radiation, >0 decreases␣
↪theta
 EXFswnet|  1 | SM | W/m^2          | Net upward shortwave radiation, >0 decreases␣
↪theta
 EXFlwdn |  1 | SM | W/m^2          | Downward longwave radiation, >0 increases theta
 EXFswdn |  1 | SM | W/m^2          | Downward shortwave radiation, >0 increases theta
 EXFqnet |  1 | SM | W/m^2          | Net upward heat flux (turb+rad), >0 decreases␣
↪theta
 EXFtaux |  1 | SU | N/m^2          | zonal surface wind stress, >0 increases uVel
 EXFtauy |  1 | SV | N/m^2          | meridional surface wind stress, >0 increases␣
↪vVel
 EXFuwind|  1 | SM | m/s            | zonal 10-m wind speed, >0 increases uVel
 EXFvwind|  1 | SM | m/s            | meridional 10-m wind speed, >0 increases uVel
 EXFwspee|  1 | SM | m/s            | 10-m wind speed modulus ( >= 0 )
 EXFatemp|  1 | SM | degK           | surface (2-m) air temperature
 EXFaqh  |  1 | SM | kg/kg          | surface (2-m) specific humidity
 EXFevap |  1 | SM | m/s            | evaporation, > 0 increases salinity
 EXFpreci|  1 | SM | m/s            | evaporation, > 0 decreases salinity
 EXFsnow |  1 | SM | m/s            | snow precipitation, > 0 decreases salinity
 EXFempmr|  1 | SM | m/s            | net upward freshwater flux, > 0 increases␣
↪salinity
 EXFpress|  1 | SM | N/m^2          | atmospheric pressure field
```

8.4.7.8 References

8.4.7.9 Experiments and tutorials that use exf

- Global Ocean experiment, in global_with_exf verification directory
- Labrador Sea experiment, in lab_sea verification directory

8.4.8 CAL: The calendar package

Authors: Christian Eckert and Patrick Heimbach

This calendar tool was originally intended to enable the use of absolute dates (Gregorian Calendar dates) in MITgcm. There is, however, a fair number of routines that can be used independently of the main MITgcm executable. After some minor modifications the whole package can be used either as a stand-alone calendar or in connection with any

dynamical model that needs calendar dates. Some straightforward extensions are still pending e.g. the availability of the Julian Calendar, to be able to resolve fractions of a second, and to have a time- step that is longer than one day.

8.4.8.1 Basic assumptions for the calendar tool

It is assumed that the SMALLEST TIME INTERVAL to be resolved is ONE SECOND.

Further assumptions are that there is an INTEGER NUMBER OF MODEL STEPS EACH DAY, and that AT LEAST ONE STEP EACH DAY is made.

Not each individual routine depends on these assumptions; there are only a few places where they enter.

8.4.8.2 Format of calendar dates

In this calendar tool a complete date specification is defined as the following integer array:

```
c            integer date(4)
c
c            ( yyyymmdd, hhmmss, leap_year, dayofweek )
c
c              date(1) = yyyymmdd    <-- Year-Month-Day
c              date(2) =   hhmmss    <-- Hours-Minutes-Seconds
c              date(3) = leap_year   <-- Leap Year/No Leap Year
c              date(4) = dayofweek   <-- Day of the Week
c
c              leap_year is either equal to 1 (normal year)
c                               or equal to 2 (leap year)
c
c              dayofweek has a range of 1 to 7.
```

In case the Gregorian Calendar is used, the first day of the week is Friday, since day of the Gregorian Calendar was Friday, 15 Oct. 1582. As a date array this date would be specified as

```
c                refdate(1) = 15821015
c                refdate(2) =        0
c                refdate(3) =        1
c                refdate(4) =        1
```

8.4.8.3 Calendar dates and time intervals

Subtracting calendar dates yields time intervals. Time intervals have the following format:

```
c          integer datediff(4)
c
c            datediff(1) = # Days
c            datediff(2) = hhmmss
c            datediff(3) =      0
c            datediff(4) =     -1
```

Such time intervals can be added to or can be subtracted from calendar dates. Time intervals can be added to and be subtracted from each other.

8.4.8.4 Using the calendar together with MITgcm

Each routine has as an argument the thread number that it is belonging to, even if this number is not used in the routine itself.

In order to include the calendar tool into the MITgcm setup the MITgcm subroutine "initialise.F" or the routine "initilise_fixed.F", depending on the MITgcm release, has to be modified in the following way:

```
c         #ifdef ALLOW_CALENDAR
c         C--   Initialise the calendar package.
c         #ifdef USE_CAL_NENDITER
c               CALL cal_Init(
c              I               startTime,
c              I               endTime,
c              I               deltaTclock,
c              I               nIter0,
c              I               nEndIter,
c              I               nTimeSteps,
c              I               myThid
c              &             )
c         #else
c               CALL cal_Init(
c              I               startTime,
c              I               endTime,
c              I               deltaTclock,
c              I               nIter0,
c              I               nTimeSteps,
c              I               myThid
c              &             )
c         #endif
c               _BARRIER
c         #endif
```

It is useful to have the CPP flag ALLOW_CALENDAR in order to switch from the usual MITgcm setup to the one that includes the calendar tool. The CPP flag USE_CAL_NENDITER has been introduced in order to enable the use of the calendar for MITgcm releases earlier than checkpoint 25 which do not have the global variable *nEndIter*.

8.4.8.5 The individual calendars

Simple model calendar:

This calendar can be used by defining

```
c                  TheCalendar='model'
```

in the calendar's data file "data.cal".

In this case a year is assumed to have 360 days. The model year is divided into 12 months with 30 days each.

Gregorian Calendar:

This calendar can be used by defining

```
c                  TheCalendar='gregorian'
```

in the calendar's data file "data.cal".

8.4.8.6 Short routine description

```
c      o  cal_Init          - Initialise the calendar. This is the interface
c                             to MITgcm.
c
c      o  cal_Set           - Sets the calendar according to the user
c                             specifications.
c
c      o  cal_GetDate       - Given the model's current timestep or the
c                             model's current time return the corresponding
c                             calendar date.
c
c      o  cal_FullDate      - Complete a date specification (leap year and
c                             day of the week).
c
c      o  cal_IsLeap        - Determine whether a given year is a leap year.
c
c      o  cal_TimePassed    - Determine the time passed between two dates.
c
c      o  cal_AddTime       - Add a time interval either to a time interval
c                             or to a date.
c
c      o  cal_TimeInterval  - Given a time interval return the corresponding
c                             date array.
c
c      o  cal_SubDates      - Determine the time interval between two dates
c                             or between two time intervals.
c
c      o  cal_ConvDate      - Decompose a date array or a time interval
c                             array into its components.
c
c      o  cal_CopyDate      - Copy a date array or a time interval array to
c                             another array.
c
c      o  cal_CompDates     - Compare two calendar dates or time intervals.
c
c      o  cal_ToSeconds     - Given a time interval array return the number
c                             of seconds.
c
c      o  cal_WeekDay       - Return the weekday as a string given the
c                             calendar date.
c
c      o  cal_NumInts       - Return the number of time intervals between two
c                             given dates.
c
c      o  cal_StepsPerDay   - Given an iteration number or the current
c                             integration time return the number of time
c                             steps to integrate in the current calendar day.
c
c      o  cal_DaysPerMonth  - Given an iteration number or the current
c                             integration time return the number of days
c                             to integrate in this calendar month.
c
c      o  cal_MonthsPerYear - Given an iteration number or the current
c                             integration time return the number of months
c                             to integrate in the current calendar year.
c
```

(continues on next page)

(continued from previous page)

```
c      o  cal_StepsForDay   - Given the integration day return the number
c                             of steps to be integrated, the first step,
c                             and the last step in the day specified. The
c                             first and the last step refer to the total
c                             number of steps (1, ... , cal_IntSteps).
c
c      o  cal_DaysForMonth  - Given the integration month return the number
c                             of days to be integrated, the first day,
c                             and the last day in the month specified. The
c                             first and the last day refer to the total
c                             number of steps (1, ... , cal_IntDays).
c
c      o  cal_MonthsForYear - Given the integration year return the number
c                             of months to be integrated, the first month,
c                             and the last month in the year specified. The
c                             first and the last step refer to the total
c                             number of steps (1, ... , cal_IntMonths).
c
c      o  cal_Intsteps      - Return the number of calendar years that are
c                             affected by the current integration.
c
c      o  cal_IntDays       - Return the number of calendar days that are
c                             affected by the current integration.
c
c      o  cal_IntMonths     - Return the number of calendar months that are
c                             affected by the current integration.
c
c      o  cal_IntYears      - Return the number of calendar years that are
c                             affected by the current integration.
c
c      o  cal_nStepDay      - Return the number of time steps that can be
c                             performed during one calendar day.
c
c      o  cal_CheckDate     - Do some simple checks on a date array or on a
c                             time interval array.
c
c      o  cal_PrintError    - Print error messages according to the flags
c                             raised by the calendar routines.
c
c      o  cal_PrintDate     - Print a date array in some format suitable for
c                             MITgcm's protocol output.
c
c      o  cal_TimeStamp     - Given the time and the iteration number return
c                             the date and print all the above numbers.
c
c      o  cal_Summary       - List all the setttings of the calendar tool.
```

8.4.8.7 Experiments and tutorials that use cal

- Global ocean experiment in global_with_exf verification directory.
- Labrador Sea experiment in lab_sea verification directory.

8.5 Atmosphere Packages

8.5.1 Atmospheric Intermediate Physics: AIM

Note: The folowing document below describes the `aim_v23` package that is based on the version v23 of the SPEEDY code ().

8.5.1.1 Key subroutines, parameters and files

8.5.1.2 AIM Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units    -->|<- Tile (max=80c)
------------------------------------------------------------------------
DIABT   |  5 |SM      ML      |K/s              |Pot. Temp.  Tendency (Mass-Weighted)␣
↪from Diabatic Processes
DIABQ   |  5 |SM      ML      |g/kg/s           |Spec.Humid. Tendency (Mass-Weighted)␣
↪from Diabatic Processes
RADSW   |  5 |SM      ML      |K/s              |Temperature Tendency due to Shortwave␣
↪Radiation (TT_RSW)
RADLW   |  5 |SM      ML      |K/s              |Temperature Tendency due to Longwave ␣
↪Radiation (TT_RLW)
DTCONV  |  5 |SM      MR      |K/s              |Temperature Tendency due to␣
↪Convection (TT_CNV)
TURBT   |  5 |SM      ML      |K/s              |Temperature Tendency due to␣
↪Turbulence in PBL (TT_PBL)
DTLS    |  5 |SM      ML      |K/s              |Temperature Tendency due to Large-
↪scale condens. (TT_LSC)
DQCONV  |  5 |SM      MR      |g/kg/s           |Spec. Humidity Tendency due to␣
↪Convection (QT_CNV)
TURBQ   |  5 |SM      ML      |g/kg/s           |Spec. Humidity Tendency due to␣
↪Turbulence in PBL (QT_PBL)
DQLS    |  5 |SM      ML      |g/kg/s           |Spec. Humidity Tendency due to Large-
↪Scale Condens. (QT_LSC)
TSR     |  1 |SM P    U1      |W/m^2            |Top-of-atm. net Shortwave Radiation␣
↪(+=dw)
OLR     |  1 |SM P    U1      |W/m^2            |Outgoing Longwave  Radiation (+=up)
RADSWG  |  1 |SM P    L1      |W/m^2            |Net Shortwave Radiation at the Ground␣
↪(+=dw)
RADLWG  |  1 |SM      L1      |W/m^2            |Net Longwave  Radiation at the Ground␣
↪(+=up)
HFLUX   |  1 |SM      L1      |W/m^2            |Sensible Heat Flux (+=up)
EVAP    |  1 |SM      L1      |g/m^2/s          |Surface Evaporation (g/m2/s)
PRECON  |  1 |SM P    L1      |g/m^2/s          |Convective  Precipitation (g/m2/s)
PRECLS  |  1 |SM      M1      |g/m^2/s          |Large Scale Precipitation (g/m2/s)
CLDFRC  |  1 |SM P    M1      |0-1              |Total Cloud Fraction (0-1)
CLDPRS  |  1 |SM PC167M1      |0-1              |Cloud Top Pressure (normalized)
CLDMAS  |  5 |SM P    LL      |kg/m^2/s         |Cloud-base Mass Flux  (kg/m^2/s)
DRAG    |  5 |SM P    LL      |kg/m^2/s         |Surface Drag Coefficient (kg/m^2/s)
WINDS   |  1 |SM P    L1      |m/s              |Surface Wind Speed  (m/s)
TS      |  1 |SM      L1      |K                |near Surface Air Temperature  (K)
QS      |  1 |SM P    L1      |g/kg             |near Surface Specific Humidity  (g/kg)
ENPREC  |  1 |SM      M1      |W/m^2            |Energy flux associated with precip.␣
↪(snow, rain Temp)
ALBVISDF|  1 |SM P    L1      |0-1              |Surface Albedo (Visible band) (0-1)
```

(continues on next page)

(continued from previous page)

```
DWNLWG   |  1 |SM P    L1      |W/m^2           |Downward Component of Longwave Flux␣
↪at the Ground (+=dw)
SWCLR    |  5 |SM      ML      |K/s             |Clear Sky Temp. Tendency due to␣
↪Shortwave Radiation
LWCLR    |  5 |SM      ML      |K/s             |Clear Sky Temp. Tendency due to␣
↪Longwave  Radiation
TSRCLR   |  1 |SM P    U1      |W/m^2           |Clear Sky Top-of-atm. net Shortwave␣
↪Radiation (+=dw)
OLRCLR   |  1 |SM P    U1      |W/m^2           |Clear Sky Outgoing Longwave ␣
↪Radiation  (+=up)
SWGCLR   |  1 |SM P    L1      |W/m^2           |Clear Sky Net Shortwave Radiation at␣
↪the Ground (+=dw)
LWGCLR   |  1 |SM      L1      |W/m^2           |Clear Sky Net Longwave  Radiation at␣
↪the Ground (+=up)
UFLUX    |  1 |UM   184L1      |N/m^2           |Zonal Wind Surface Stress  (N/m^2)
VFLUX    |  1 |VM   183L1      |N/m^2           |Meridional Wind Surface Stress  (N/m^
↪2)
DTSIMPL  |  1 |SM P    L1      |K               |Surf. Temp Change after 1 implicit␣
↪time step
```

8.5.1.3 Experiments and tutorials that use aim

- Global atmosphere experiment in aim.5l_cs verification directory.

8.5.2 Land package

8.5.2.1 Introduction

This package provides a simple land model based on Rong Zhang [e-mail:roz@gfdl.noaa.gov] 2 layers model (see documentation below).

It is primarily implemented for AIM (_v23) atmospheric physics but could be adapted to work with a different atmospheric physics. Two subroutines (*aim_aim2land.F aim_land2aim.F* in *pkg/aim_v23*) are used as interface with AIM physics.

Number of layers is a parameter (*land_nLev* in *LAND_SIZE.h*) and can be changed.

Note on Land Model date: June 1999 author: Rong Zhang

8.5.2.2 Equations and Key Parameters

This is a simple 2-layer land model. The top layer depth $z1 = 0.1m$, the second layer depth $z2 = 4m$.

Let T_{g1}, T_{g2} be the temperature of each layer, W_1, W_2 be the soil moisture of each layer. The field capacity f_1, f_2 are the maximum water amount in each layer, so W_i is the ratio of available water to field capacity. $f_i = \gamma z_i, \gamma = 0.24$ is the field capapcity per meter soil, so $f_1 = 0.024m$, $f_2 = 0.96m$.

The land temperature is determined by total surface downward heat flux F,

$$z_1 C_1 \frac{dT_{g1}}{dt} = F - \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2}$$

$$z_2 C_2 \frac{dT_{g2}}{dt} = \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2}$$

here C_1, C_2 are the heat capacity of each layer , λlambda =0.42Wm^{-1}K^{-1}.‘

$$C_1 = C_w W_1 \gamma + C_s$$

$$C_2 = C_w W_2 \gamma + C_s$$

C_w, C_s are the heat capacity of water and dry soil respectively. $C_w = 4.2 \times 10^6 Jm^{-3}K^{-1}, C_s = 1.13 \times 10^6 Jm^{-3}K^{-1}$.

The soil moisture is determined by precipitation $P(m/s)$,surface evaporation $E(m/s)$ and runoff $R(m/s)$.

$$\frac{dW_1}{dt} = \frac{P - E - R}{f_1} + \frac{W_2 - W_1}{\tau}$$

$\tau = 2\ days$ is the time constant for diffusion of moisture between layers.

$$\frac{dW_2}{dt} = \frac{f_1}{f_2}\frac{W_1 - W_2}{\tau}$$

In the code, $R = 0$ gives better result, W_1, W_2 are set to be within [0, 1]. If W_1 is greater than 1, then let $\delta W_1 = W_1 - 1, W_1 = 1$ and $W_2 = W_2 + p\delta W_1 \frac{f_1}{f_2}$, i.e. the runoff of top layer is put into second layer. $p = 0.5$ is the fraction of top layer runoff that is put into second layer.

The time step is 1 hour, it takes several years to reach equalibrium offline.

8.5.2.3 Land diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
GrdSurfT|   1 |SM      Lg      |degC            |Surface Temperature over land
GrdTemp |   2 |SM      MG      |degC            |Ground Temperature at each level
GrdEnth |   2 |SM      MG      |J/m3            |Ground Enthalpy at each level
GrdWater|   2 |SM P    MG      |0-1             |Ground Water (vs Field Capacity)␣
↪Fraction at each level
LdSnowH |   1 |SM P    Lg      |m               |Snow Thickness over land
LdSnwAge|   1 |SM P    Lg      |s               |Snow Age over land
RUNOFF  |   1 |SM      L1      |m/s             |Run-Off per surface unit
EnRunOff|   1 |SM      L1      |W/m^2           |Energy flux associated with run-Off
landHFlx|   1 |SM      Lg      |W/m^2           |net surface downward Heat flux over␣
↪land
landPmE |   1 |SM      Lg      |kg/m^2/s        |Precipitation minus Evaporation over␣
↪land
ldEnFxPr|   1 |SM      Lg      |W/m^2           |Energy flux (over land) associated␣
↪with Precip (snow,rain)
```

8.5.2.4 References

Hansen J. et al. Efficient three-dimensional global models for climate studies: models I and II. *Monthly Weather Review*, vol.111, no.4, pp. 609-62, 1983

8.5.2.5 Experiments and tutorials that use land

- Global atmosphere experiment in aim.5l_cs verification directory.

8.5.3 Fizhi: High-end Atmospheric Physics

8.5.3.1 Introduction

The fizhi (high-end atmospheric physics) package includes a collection of state-of-the-art physical parameterizations for atmospheric radiation, cumulus convection, atmospheric boundary layer turbulence, and land surface processes. The collection of atmospheric physics parameterizations were originally used together as part of the GEOS-3 (Goddard Earth Observing System-3) GCM developed at the NASA/Goddard Global Modelling and Assimilation Office (GMAO).

8.5.3.2 Equations

Moist Convective Processes:

Sub-grid and Large-scale Convection

Sub-grid scale cumulus convection is parameterized using the Relaxed Arakawa Schubert (RAS) scheme of [MS92], which is a linearized Arakawa Schubert type scheme. RAS predicts the mass flux from an ensemble of clouds. Each subensemble is identified by its entrainment rate and level of neutral bouyancy which are determined by the grid-scale properties.

The thermodynamic variables that are used in RAS to describe the grid scale vertical profile are the dry static energy, $s = c_pT + gz$, and the moist static energy, $h = c_pT + gz + Lq$. The conceptual model behind RAS depicts each subensemble as a rising plume cloud, entraining mass from the environment during ascent, and detraining all cloud air at the level of neutral buoyancy. RAS assumes that the normalized cloud mass flux, η, normalized by the cloud base mass flux, is a linear function of height, expressed as:

$$\frac{\partial\eta(z)}{\partial z} = \lambda \quad or \quad \frac{\partial\eta(P^\kappa)}{\partial P^\kappa} = -\frac{c_p}{g}\theta\lambda$$

where we have used the hydrostatic equation written in the form:

$$\frac{\partial z}{\partial P^\kappa} = -\frac{c_p}{g}\theta$$

The entrainment parameter, λ, characterizes a particular subensemble based on its detrainment level, and is obtained by assuming that the level of detrainment is the level of neutral buoyancy, ie., the level at which the moist static energy of the cloud, h_c, is equal to the saturation moist static energy of the environment, h^*. Following [MS92], λ may be written as

$$\lambda = \frac{h_B - h_D^*}{\frac{c_p}{g}\int_{P_D}^{P_B}\theta(h_D^* - h)dP^\kappa},$$

where the subscript B refers to cloud base, and the subscript D refers to the detrainment level.

The convective instability is measured in terms of the cloud work function A, defined as the rate of change of cumulus kinetic energy. The cloud work function is related to the buoyancy, or the difference between the moist static energy in the cloud and in the environment:

$$A = \int_{P_D}^{P_B}\frac{\eta}{1+\gamma}\left[\frac{h_c - h^*}{P^\kappa}\right]dP^\kappa$$

where γ is $\frac{L}{c_p}\frac{\partial q^*}{\partial T}$ obtained from the Claussius Clapeyron equation, and the subscript c refers to the value inside the cloud.

To determine the cloud base mass flux, the rate of change of A in time *due to dissipation by the clouds* is assumed to approximately balance the rate of change of A *due to the generation by the large scale*. This is the quasi-equilibrium assumption, and results in an expression for m_B:

$$m_B = \frac{-\frac{dA}{dt}\big|_{ls}}{K}$$

where K is the cloud kernel, defined as the rate of change of the cloud work function per unit cloud base mass flux, and is currently obtained by analytically differentiating the expression for A in time. The rate of change of A due to the generation by the large scale can be written as the difference between the current $A(t + \Delta t)$ and its equillibrated value after the previous convective time step $A(t)$, divided by the time step. $A(t)$ is approximated as some critical A_{crit}, computed by Lord (1982) from $insitu$ observations.

The predicted convective mass fluxes are used to solve grid-scale temperature and moisture budget equations to determine the impact of convection on the large scale fields of temperature (through latent heating and compensating subsidence) and moisture (through precipitation and detrainment):

$$\left.\frac{\partial \theta}{\partial t}\right|_c = \alpha \frac{m_B}{c_p P^\kappa} \eta \frac{\partial s}{\partial p}$$

and

$$\left.\frac{\partial q}{\partial t}\right|_c = \alpha \frac{m_B}{L} \eta \left(\frac{\partial h}{\partial p} - \frac{\partial s}{\partial p}\right)$$

where $\theta = \frac{T}{P^\kappa}$, $P = (p/p_0)$, and α is the relaxation parameter.

As an approximation to a full interaction between the different allowable subensembles, many clouds are simulated frequently, each modifying the large scale environment some fraction α of the total adjustment. The parameterization thereby "relaxes" the large scale environment towards equillibrium.

In addition to the RAS cumulus convection scheme, the fizhi package employs a Kessler-type scheme for the re-evaporation of falling rain [SM88], which correspondingly adjusts the temperature assuming h is conserved. RAS in its current formulation assumes that all cloud water is deposited into the detrainment level as rain. All of the rain is available for re-evaporation, which begins in the level below detrainment. The scheme accounts for some microphysics such as the rainfall intensity, the drop size distribution, as well as the temperature, pressure and relative humidity of the surrounding air. The fraction of the moisture deficit in any model layer into which the rain may re-evaporate is controlled by a free parameter, which allows for a relatively efficient re-evaporation of liquid precipitate and larger rainout for frozen precipitation.

Due to the increased vertical resolution near the surface, the lowest model layers are averaged to provide a 50 mb thick sub-cloud layer for RAS. Each time RAS is invoked (every ten simulated minutes), a number of randomly chosen subensembles are checked for the possibility of convection, from just above cloud base to 10 mb.

Supersaturation or large-scale precipitation is initiated in the fizhi package whenever the relative humidity in any grid-box exceeds a critical value, currently 100 %. The large-scale precipitation re-evaporates during descent to partially saturate lower layers in a process identical to the re-evaporation of convective rain.

Cloud Formation

Convective and large-scale cloud fractons which are used for cloud-radiative interactions are determined diagnostically as part of the cumulus and large-scale parameterizations. Convective cloud fractions produced by RAS are proportional to the detrained liquid water amount given by

$$F_{RAS} = \min\left[\frac{l_{RAS}}{l_c}, 1.0\right]$$

where l_c is an assigned critical value equal to 1.25 g/kg. A memory is associated with convective clouds defined by:

$$F_{RAS}^n = \min\left[F_{RAS} + (1 - \frac{\Delta t_{RAS}}{\tau})F_{RAS}^{n-1}, 1.0\right]$$

where F_{RAS} is the instantanious cloud fraction and F_{RAS}^{n-1} is the cloud fraction from the previous RAS timestep. The memory coefficient is computed using a RAS cloud timescale, τ, equal to 1 hour. RAS cloud fractions are cleared when they fall below 5 %.

Large-scale cloudiness is defined, following Slingo and Ritter (1985), as a function of relative humidity:

$$F_{LS} = \min\left[\left(\frac{RH - RH_c}{1 - RH_c}\right)^2, 1.0\right]$$

where

RH$_c$ & = & 1-s(1-s)(2-+2 s)r s & = & p/p$_{surf}$ r & = & ( ) RH$_{min}$ & = & 0.75 & = & 0.573285 .

These cloud fractions are suppressed, however, in regions where the convective sub-cloud layer is conditionally unstable. The functional form of RH_c is shown in Figure 8.8

The total cloud fraction in a grid box is determined by the larger of the two cloud fractions:

$$F_{CLD} = \max\left[F_{RAS}, F_{LS}\right].$$

Finally, cloud fractions are time-averaged between calls to the radiation packages.

Radiation:

The parameterization of radiative heating in the fizhi package includes effects from both shortwave and longwave processes. Radiative fluxes are calculated at each model edge-level in both up and down directions. The heating rates/cooling rates are then obtained from the vertical divergence of the net radiative fluxes.

The net flux is

$$F = F^{\uparrow} - F^{\downarrow}$$

where F is the net flux, $F^{\uparrow}$ is the upward flux and $F^{\downarrow}$ is the downward flux.

The heating rate due to the divergence of the radiative flux is given by

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial F}{\partial z}$$

or

$$\frac{\partial T}{\partial t} = \frac{g}{c_p \pi}\frac{\partial F}{\partial \sigma}$$

where g is the accelation due to gravity and c_p is the heat capacity of air at constant pressure.

The time tendency for Longwave Radiation is updated every 3 hours. The time tendency for Shortwave Radiation is updated once every three hours assuming a normalized incident solar radiation, and subsequently modified at every model time step by the true incident radiation. The solar constant value used in the package is equal to 1365 W/m^2 and a CO_2 mixing ratio of 330 ppm. For the ozone mixing ratio, monthly mean zonally averaged climatological values specified as a function of latitude and height [RSG87] are linearly interpolated to the current time.

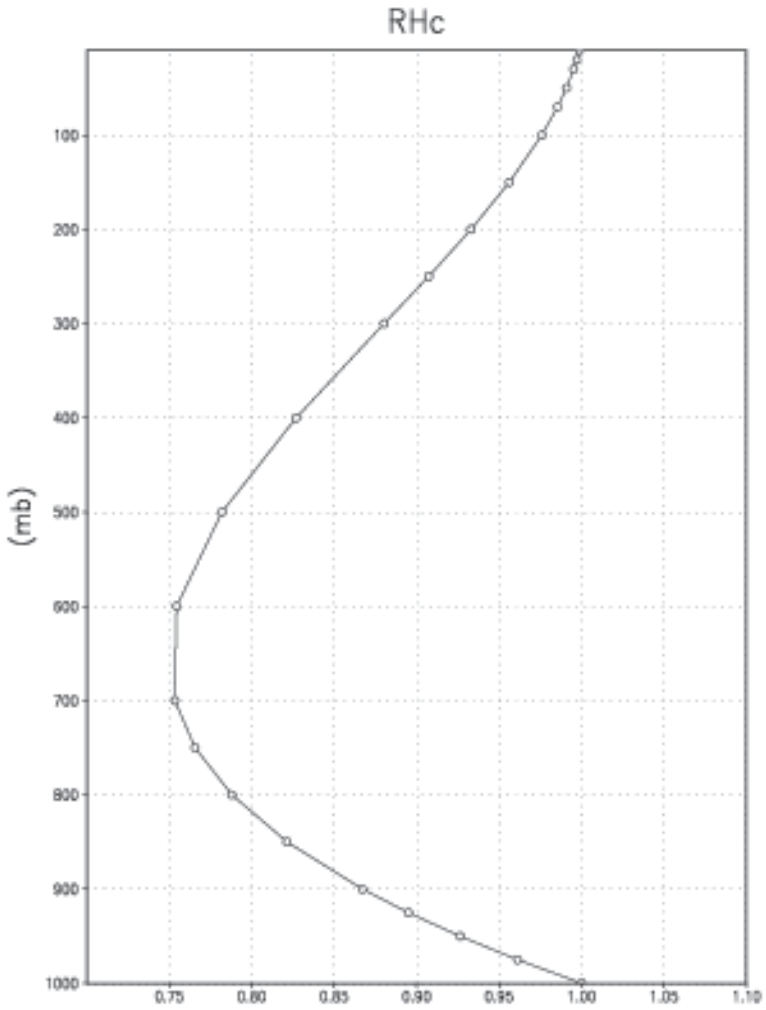

Figure 8.8: Critical Relative Humidity for Clouds.

Shortwave Radiation

The shortwave radiation package used in the package computes solar radiative heating due to the absoption by water vapor, ozone, carbon dioxide, oxygen, clouds, and aerosols and due to the scattering by clouds, aerosols, and gases. The shortwave radiative processes are described by [Cho90][Cho92]. This shortwave package uses the Delta-Eddington approximation to compute the bulk scattering properties of a single layer following King and Harshvardhan (JAS, 1986). The transmittance and reflectance of diffuse radiation follow the procedures of Sagan and Pollock (JGR, 1967) and [LH74].

Highly accurate heating rate calculations are obtained through the use of an optimal grouping strategy of spectral bands. By grouping the UV and visible regions as indicated in Table 8.8, the Rayleigh scattering and the ozone absorption of solar radiation can be accurately computed in the ultraviolet region and the photosynthetically active radiation (PAR) region. The computation of solar flux in the infrared region is performed with a broadband parameterization using the spectrum regions shown in Table 8.9. The solar radiation algorithm used in the fizhi package can be applied not only for climate studies but also for studies on the photolysis in the upper atmosphere and the photosynthesis in the biosphere.

Table 8.8: UV and Visible Spectral Regions used in shortwave radiation package.

UV and Visible Spectral Regions		
Region	Band	Wavelength (micron)
UV-C	1.	.175 - .225
	2.	.225 - .245
		.260 - .280
	3.	.245 - .260
UV-B	4.	.280 - .295
	5.	.295 - .310
	6.	.310 - .320
UV-A	7.	.320 - .400
PAR	8.	.400 - .700

Table 8.9: Infrared Spectral Regions used in shortwave radiation package.

Infrared Spectral Regions		
Band	Wavenumber (cm^{-1})	Wavelength (micron)
1	1000-4400	2.27-10.0
2	4400-8200	1.22-2.27
3	8200-14300	0.70-1.22

Within the shortwave radiation package, both ice and liquid cloud particles are allowed to co-exist in any of the model layers. Two sets of cloud parameters are used, one for ice paticles and the other for liquid particles. Cloud parameters are defined as the cloud optical thickness and the effective cloud particle size. In the fizhi package, the effective radius for water droplets is given as 10 microns, while 65 microns is used for ice particles. The absorption due to aerosols is currently set to zero.

To simplify calculations in a cloudy atmosphere, clouds are grouped into low ($p > 700$ mb), middle (700 mb $\geq p >$ 400 mb), and high ($p < 400$ mb) cloud regions. Within each of the three regions, clouds are assumed maximally overlapped, and the cloud cover of the group is the maximum cloud cover of all the layers in the group. The optical thickness of a given layer is then scaled for both the direct (as a function of the solar zenith angle) and diffuse beam radiation so that the grouped layer reflectance is the same as the original reflectance. The solar flux is computed for each of eight cloud realizations possible within this low/middle/high classification, and appropriately averaged to produce the net solar flux.

Longwave Radiation

The longwave radiation package used in the fizhi package is thoroughly described by . As described in that document, IR fluxes are computed due to absorption by water vapor, carbon dioxide, and ozone. The spectral bands together with their absorbers and parameterization methods, configured for the fizhi package, are shown in Table 8.10.

Table 8.10: IR Spectral Bands, Absorbers, and Parameterization Method (from [chsz:94])

IR Spectral Bands			
Band	Spectral Range (cm^{-1})	Absorber	Method
1	0-340	H_2O line	T
2	340-540	H_2O line	T
3a	540-620	H_2O line	K
3b	620-720	H_2O continuum	S
3b	720-800	CO_2	T
4	800-980	H_2O line	K
		H_2O continuum	S
		H_2O line	K
5	980-1100	H_2O continuum	S
		O_3	T
6	1100-1380	H_2O line	K
		H_2O continuum	S
7	1380-1900	H_2O line	T
8	1900-3000	H_2O line	K
K: k-distribution method with linear pressure scaling			
T: Table look-up with temperature and pressure scaling			
S: One-parameter temperature scaling			

The longwave radiation package accurately computes cooling rates for the middle and lower atmosphere from 0.01 mb to the surface. Errors are < 0.4 C day^{-1} in cooling rates and $< 1\%$ in fluxes. From Chou and Suarez, it is estimated that the total effect of neglecting all minor absorption bands and the effects of minor infrared absorbers such as nitrous oxide (N:math:_2O), methane (CH:math:_4), and the chlorofluorocarbons (CFCs), is an underestimate of ≈ 5 W/m^2 in the downward flux at the surface and an overestimate of ≈ 3 W/m^2 in the upward flux at the top of the atmosphere.

Similar to the procedure used in the shortwave radiation package, clouds are grouped into three regions catagorized as low/middle/high. The net clear line-of-site probability (P) between any two levels, p_1 and p_2 $(p_2 > p_1)$, assuming randomly overlapped cloud groups, is simply the product of the probabilities within each group:

$$P_{net} = P_{low} \times P_{mid} \times P_{hi}.$$

Since all clouds within a group are assumed maximally overlapped, the clear line-of-site probability within a group is given by:

$$P_{group} = 1 - F_{max},$$

where F_{max} is the maximum cloud fraction encountered between p_1 and p_2 within that group. For groups and/or levels outside the range of p_1 and p_2, a clear line-of-site probability equal to 1 is assigned.

Cloud-Radiation Interaction

The cloud fractions and diagnosed cloud liquid water produced by moist processes within the fizhi package are used in the radiation packages to produce cloud-radiative forcing. The cloud optical thickness associated with large-scale cloudiness is made proportional to the diagnosed large-scale liquid water, ℓ, detrained due to super-saturation. Two values are used corresponding to cloud ice particles and water droplets. The range of optical thickness for these clouds is given as

$$0.0002 \leq \tau_{ice}(mb^{-1}) \leq 0.002 \quad \text{for} \quad 0 \leq \ell \leq 2 \quad \text{mg/kg},$$

$$0.02 \leq \tau_{h_2o}(mb^{-1}) \leq 0.2 \quad \text{for} \quad 0 \leq \ell \leq 10 \quad \text{mg/kg}.$$

The partitioning, α, between ice particles and water droplets is achieved through a linear scaling in temperature:

$$0 \leq \alpha \leq 1 \quad \text{for} \quad 233.15 \leq T \leq 253.15.$$

The resulting optical depth associated with large-scale cloudiness is given as

$$\tau_{LS} = \alpha\tau_{h_2o} + (1 - \alpha)\tau_{ice}.$$

The optical thickness associated with sub-grid scale convective clouds produced by RAS is given as

$$\tau_{RAS} = 0.16 \quad mb^{-1}.$$

The total optical depth in a given model layer is computed as a weighted average between the large-scale and sub-grid scale optical depths, normalized by the total cloud fraction in the layer:

$$\tau = \left(\frac{F_{RAS} \ \tau_{RAS} + F_{LS} \ \tau_{LS}}{F_{RAS} + F_{LS}} \right) \Delta p,$$

where F_{RAS} and F_{LS} are the time-averaged cloud fractions associated with RAS and large-scale processes described in Section [sec:fizhi:clouds]. The optical thickness for the longwave radiative feedback is assumed to be 75 $\%$ of these values.

The entire Moist Convective Processes Module is called with a frequency of 10 minutes. The cloud fraction values are time-averaged over the period between Radiation calls (every 3 hours). Therefore, in a time-averaged sense, both convective and large-scale cloudiness can exist in a given grid-box.

Turbulence

Turbulence is parameterized in the fizhi package to account for its contribution to the vertical exchange of heat, moisture, and momentum. The turbulence scheme is invoked every 30 minutes, and employs a backward-implicit iterative time scheme with an internal time step of 5 minutes. The tendencies of atmospheric state variables due to turbulent diffusion are calculated using the diffusion equations:

$$\frac{\partial u}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{u'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z})$$

$$\frac{\partial v}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{v'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z})$$
$$\frac{\partial T}{\partial t} = P^\kappa \frac{\partial \theta}{\partial t}_{turb} = P^\kappa \frac{\partial}{\partial z}(-\overline{w'\theta'}) = P^\kappa \frac{\partial}{\partial z}(K_h \frac{\partial \theta_v}{\partial z})$$
$$\frac{\partial q}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{w'q'}) = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z})$$

Within the atmosphere, the time evolution of second turbulent moments is explicitly modeled by representing the third moments in terms of the first and second moments. This approach is known as a second-order closure modeling. To simplify and streamline the computation of the second moments, the level 2.5 assumption of Mellor and Yamada (1974) and [Yam77] is employed, in which only the turbulent kinetic energy (TKE),

$$\frac{1}{2}q^2 = \overline{u'^2} + \overline{v'^2} + \overline{w'^2},$$

is solved prognostically and the other second moments are solved diagnostically. The prognostic equation for TKE allows the scheme to simulate some of the transient and diffusive effects in the turbulence. The TKE budget equation is solved numerically using an implicit backward computation of the terms linear in q^2 and is written:

$$\frac{d}{dt}(\frac{1}{2}q^2) - \frac{\partial}{\partial z}(\frac{5}{3}\lambda_1 q \frac{\partial}{\partial z}(\frac{1}{2}q^2)) = -\overline{u'w'}\frac{\partial U}{\partial z} - \overline{v'w'}\frac{\partial V}{\partial z} + \frac{g}{\Theta_0}\overline{w'\theta_v{}'} - \frac{q^3}{\Lambda_1}$$

where q is the turbulent velocity, u', v', w' and θ' are the fluctuating parts of the velocity components and potential temperature, U and V are the mean velocity components, ${\Theta_0}^{-1}$ is the coefficient of thermal expansion, and λ_1 and Λ_1 are constant multiples of the master length scale, ℓ, which is designed to be a characteristic measure of the vertical structure of the turbulent layers.

The first term on the left-hand side represents the time rate of change of TKE, and the second term is a representation of the triple correlation, or turbulent transport term. The first three terms on the right-hand side represent the sources of TKE due to shear and bouyancy, and the last term on the right hand side is the dissipation of TKE.

In the level 2.5 approach, the vertical fluxes of the scalars θ_v and q and the wind components u and v are expressed in terms of the diffusion coefficients K_h and K_m, respectively. In the statisically realizable level 2.5 turbulence scheme of [HL88], these diffusion coefficients are expressed as

$$K_h = \begin{cases} q\,\ell\,S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_H(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

and

$$K_m = \begin{cases} q\,\ell\,S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_M(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where the subscript e refers to the value under conditions of local equillibrium (obtained from the Level 2.0 Model), ℓ is the master length scale related to the vertical structure of the atmosphere, and S_M and S_H are functions of G_H and G_M, the dimensionless buoyancy and wind shear parameters, respectively. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number:

$$\mathbf{RI} = \frac{\frac{g}{\theta_v}\frac{\partial \theta_v}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2} = \frac{c_p \frac{\partial \theta_v}{\partial z}\frac{\partial P^\kappa}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2}.$$

Negative values indicate unstable buoyancy and shear, small positive values (< 0.2) indicate dominantly unstable shear, and large positive values indicate dominantly stable stratification.

Turbulent eddy diffusion coefficients of momentum, heat and moisture in the surface layer, which corresponds to the lowest GCM level (see — *missing table* —), are calculated using stability-dependant functions based on Monin-Obukhov theory:

$$K_m(surface) = C_u \times u_* = C_D W_s$$

and

$$K_h(surface) = C_t \times u_* = C_H W_s$$

where $u_* = C_u W_s$ is the surface friction velocity, C_D is termed the surface drag coefficient, C_H the heat transfer coefficient, and W_s is the magnitude of the surface layer wind.

C_u is the dimensionless exchange coefficient for momentum from the surface layer similarity functions:

$$C_u = \frac{u_*}{W_s} = \frac{k}{\psi_m}$$

where k is the Von Karman constant and ψ_m is the surface layer non-dimensional wind shear given by

$$\psi_m = \int_{\zeta_0}^{\zeta} \frac{\phi_m}{\zeta} d\zeta.$$

Here ζ is the non-dimensional stability parameter, and ϕ_m is the similarity function of ζ which expresses the stability dependance of the momentum gradient. The functional form of ϕ_m is specified differently for stable and unstable layers.

C_t is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions:

$$C_t = -\frac{(\overline{w'\theta'})}{u_* \Delta\theta} = -\frac{(\overline{w'q'})}{u_* \Delta q} = \frac{k}{(\psi_h + \psi_g)}$$

where ψ_h is the surface layer non-dimensional temperature gradient given by

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta.$$

Here ϕ_h is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, and is specified differently for stable and unstable layers according to [HS95].

ψ_g is the non-dimensional temperature or moisture gradient in the viscous sublayer, which is the mosstly laminar region between the surface and the tops of the roughness elements, in which temperature and moisture gradients can be quite large. Based on [YK74]:

$$\psi_g = \frac{0.55(Pr^{2/3} - 0.2)}{\nu^{1/2}} (h_0 u_* - h_{0_{ref}} u_{*_{ref}})^{1/2}$$

where Pr is the Prandtl number for air, ν is the molecular viscosity, z_0 is the surface roughness length, and the subscript *ref* refers to a reference value. $h_0 = 30 z_0$ with a maximum value over land of 0.01

The surface roughness length over oceans is is a function of the surface-stress velocity,

$$z_0 = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + \frac{c_5}{u_*}$$

where the constants are chosen to interpolate between the reciprocal relation of [Kon75] for weak winds, and the piecewise linear relation of [LP81] for moderate to large winds. Roughness lengths over land are specified from the climatology of [DS89].

For an unstable surface layer, the stability functions, chosen to interpolate between the condition of small values of β and the convective limit, are the KEYPS function [Pan73] for momentum, and its generalization for heat and moisture:

$$\phi_m{}^4 - 18\zeta\phi_m{}^3 = 1 \qquad ; \qquad \phi_h{}^2 - 18\zeta\phi_h{}^3 = 1 \qquad .$$

The function for heat and moisture assures non-vanishing heat and moisture fluxes as the wind speed approaches zero.

For a stable surface layer, the stability functions are the observationally based functions of [Cla70], slightly modified for the momemtum flux:

$$\phi_m = \frac{1+5\zeta_1}{1+0.00794\zeta_1(1+5\zeta_1)} \qquad ; \qquad \phi_h = \frac{1+5\zeta_1}{1+0.00794\zeta(1+5\zeta_1)}.$$

The moisture flux also depends on a specified evapotranspiration coefficient, set to unity over oceans and dependant on the climatological ground wetness over land.

Once all the diffusion coefficients are calculated, the diffusion equations are solved numerically using an implicit backward operator.

Atmospheric Boundary Layer

The depth of the atmospheric boundary layer (ABL) is diagnosed by the parameterization as the level at which the turbulent kinetic energy is reduced to a tenth of its maximum near surface value. The vertical structure of the ABL is explicitly resolved by the lowest few (3-8) model layers.

Surface Energy Budget

The ground temperature equation is solved as part of the turbulence package using a backward implicit time differencing scheme:

$$C_g \frac{\partial T_g}{\partial t} = R_{sw} - R_{lw} + Q_{ice} - H - LE$$

where R_{sw} is the net surface downward shortwave radiative flux and R_{lw} is the net surface upward longwave radiative flux.

H is the upward sensible heat flux, given by:

$$H = P^{\kappa}\rho c_p C_H W_s(\theta_{surface} - \theta_{NLAY}) \qquad where: \;\; C_H = C_u C_t$$

where ρ = the atmospheric density at the surface, c_p is the specific heat of air at constant pressure, and θ represents the potential temperature of the surface and of the lowest σ-level, respectively.

The upward latent heat flux, LE, is given by

$$LE = \rho\beta L C_H W_s(q_{surface} - q_{NLAY}) \qquad where: \;\; C_H = C_u C_t$$

where β is the fraction of the potential evapotranspiration actually evaporated, L is the latent heat of evaporation, and $q_{surface}$ and q_{NLAY} are the specific humidity of the surface and of the lowest σ-level, respectively.

The heat conduction through sea ice, Q_{ice}, is given by

$$Q_{ice} = \frac{C_{ti}}{H_i}(T_i - T_g)$$

where C_{ti} is the thermal conductivity of ice, H_i is the ice thickness, assumed to be 3 m where sea ice is present, T_i is 273 degrees Kelvin, and T_g is the surface temperature of the ice.

C_g is the total heat capacity of the ground, obtained by solving a heat diffusion equation for the penetration of the diurnal cycle into the ground (), and is given by:

$$C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{(0.386 + 0.536W + 0.15W^2)2 \times 10^{-3}\frac{86400}{2\pi}} \; .$$

Here, the thermal conductivity, λ, is equal to 2×10^{-3} $\frac{ly}{sec}\frac{cm}{K}$, the angular velocity of the earth, ω, is written as 86400 sec/day divided by 2π $radians/day$, and the expression for C_s, the heat capacity per unit volume at the surface, is a function of the ground wetness, W.

Land Surface Processes:

Surface Type

The fizhi package surface Types are designated using the Koster-Suarez [KS91][KS92] Land Surface Model (LSM) mosaic philosophy which allows multiple "tiles", or multiple surface types, in any one grid cell. The Koster-Suarez LSM surface type classifications are shown in Table 8.11. The surface types and the percent of the grid cell occupied by any surface type were derived from the surface classification of [DT94], and information about the location of permanent ice was obtained from the classifications of [DS89]. The surface type map for a 1° grid is shown in Figure 8.9. The determination of the land or sea category of surface type was made from NCAR's 10 minute by 10 minute Navy topography dataset, which includes information about the percentage of water-cover at any point. The data were averaged to the model's grid resolutions, and any grid-box whose averaged water percentage was $\geq 60\%$ was defined as a water point. The Land-Water designation was further modified subjectively to ensure sufficient representation from small but isolated land and water regions.

Table 8.11: Surface Type Designation

Type	Vegetation Designation
1	Broadleaf Evergreen Trees
2	Broadleaf Deciduous Trees
3	Needleleaf Trees
4	Ground Cover
5	Broadleaf Shrubs
6	Dwarf Trees (Tundra)
7	Bare Soil
8	Desert (Bright)
9	Glacier
10	Desert (Dark)
100	Ocean

Surface Roughness

The surface roughness length over oceans is computed iteratively with the wind stress by the surface layer parameterization [HS95]. It employs an interpolation between the functions of [LP81] for high winds and of [Kon75] for weak winds.

Albedo

The surface albedo computation, described in , employs the "two stream" approximation used in Sellers' (1987) Simple Biosphere (SiB) Model which distinguishes between the direct and diffuse albedos in the visible and in the near infrared spectral ranges. The albedos are functions of the observed leaf area index (a description of the relative orientation of the leaves to the sun), the greenness fraction, the vegetation type, and the solar zenith angle. Modifications are made to account for the presence of snow, and its depth relative to the height of the vegetation elements.

Gravity Wave Drag

The fizhi package employs the gravity wave drag scheme of [ZSL95]. This scheme is a modified version of Vernekar et al. (1992), which was based on Alpert et al. (1988) and Helfand et al. (1987). In this version, the gravity wave stress at the surface is based on that derived by Pierrehumbert (1986) and is given by:

$$|\vec{\tau}_{sfc}| = \frac{\rho U^3}{N\ell^*}\left(\frac{F_r^2}{1+F_r^2}\right) ,$$

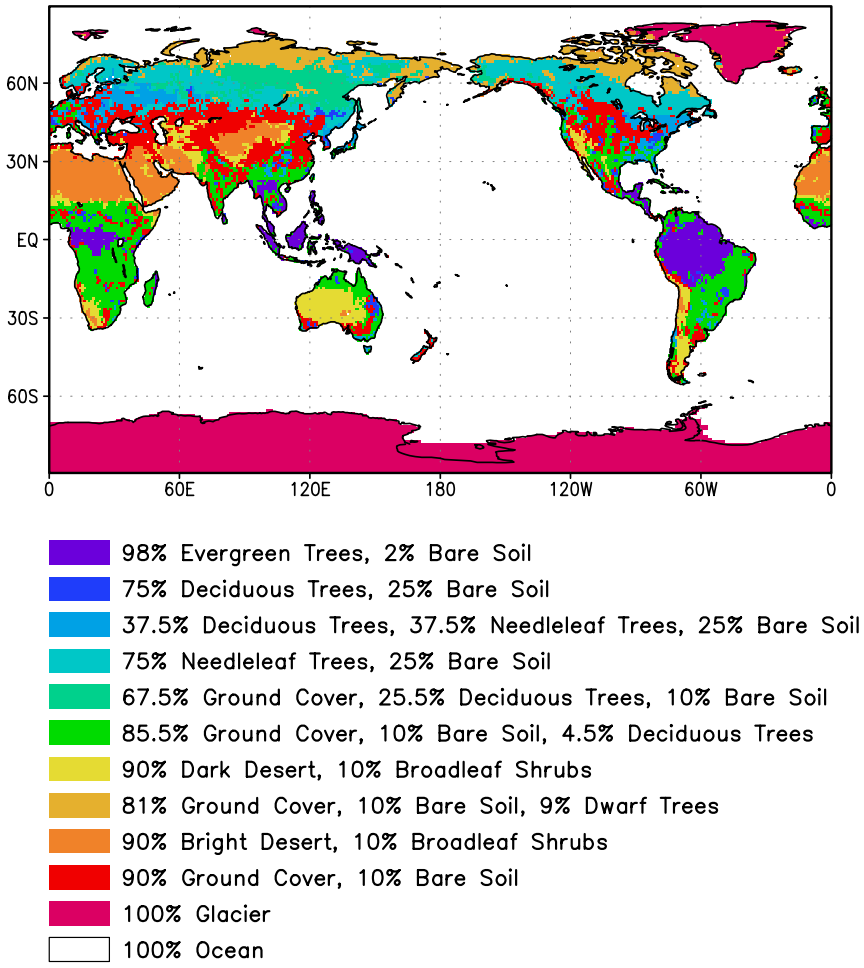

Figure 8.9: Surface type combinations

where $F_r = Nh/U$ is the Froude number, N is the *Brunt - Väisälä* frequency, U is the surface wind speed, h is the standard deviation of the sub-grid scale orography, and ℓ^* is the wavelength of the monochromatic gravity wave in the direction of the low-level wind. A modification introduced by Zhou et al. allows for the momentum flux to escape through the top of the model, although this effect is small for the current 70-level model. The subgrid scale standard deviation is defined by h, and is not allowed to exceed 400 m.

The effects of using this scheme within a GCM are shown in [TS96]. Experiments using the gravity wave drag parameterization yielded significant and beneficial impacts on both the time-mean flow and the transient statistics of the a GCM climatology, and have eliminated most of the worst dynamically driven biases in the a GCM simulation. An examination of the angular momentum budget during climate runs indicates that the resulting gravity wave torque is similar to the data-driven torque produced by a data assimilation which was performed without gravity wave drag. It was shown that the inclusion of gravity wave drag results in large changes in both the mean flow and in eddy fluxes. The result is a more accurate simulation of surface stress (through a reduction in the surface wind strength), of mountain torque (through a redistribution of mean sea-level pressure), and of momentum convergence (through a reduction in the flux of westerly momentum by transient flow eddies).

Boundary Conditions and other Input Data:

Required fields which are not explicitly predicted or diagnosed during model execution must either be prescribed internally or obtained from external data sets. In the fizhi package these fields include: sea surface temperature, sea ice estent, surface geopotential variance, vegetation index, and the radiation-related background levels of: ozone, carbon dioxide, and stratospheric moisture.

Boundary condition data sets are available at the model's resolutions for either climatological or yearly varying conditions. Any frequency of boundary condition data can be used in the fizhi package; however, the current selection of data is summarized in Table 8.12. The time mean values are interpolated during each model timestep to the current time.

Table 8.12: Boundary conditions and other input data used in the fizhi package. Also noted are the current years and frequencies available.

Fizhi Input Datasets		
Sea Ice Extent	monthly	1979-current, climatology
Sea Ice Extent	weekly	1982-current, climatology
Sea Surface Temperature	monthly	1979-current, climatology
Sea Surface Temperature	weekly	1982-current, climatology
Zonally Averaged Upper-Level Moisture	monthly	climatology
Zonally Averaged Ozone Concentration	monthly	climatology

Topography and Topography Variance

Surface geopotential heights are provided from an averaging of the Navy 10 minute by 10 minute dataset supplied by the National Center for Atmospheric Research (NCAR) to the model's grid resolution. The original topography is first rotated to the proper grid-orientation which is being run, and then averages the data to the model resolution.

The standard deviation of the subgrid-scale topography is computed by interpolating the 10 minute data to the model's resolution and re-interpolating back to the 10 minute by 10 minute resolution. The sub-grid scale variance is constructed based on this smoothed dataset.

Upper Level Moisture

The fizhi package uses climatological water vapor data above 100 mb from the Stratospheric Aerosol and Gas Experiment (SAGE) as input into the model's radiation packages. The SAGE data is archived as monthly zonal means at 5° latitudinal resolution. The data is interpolated to the model's grid location and current time, and blended with

the GCM's moisture data. Below 300 mb, the model's moisture data is used. Above 100 mb, the SAGE data is used. Between 100 and 300 mb, a linear interpolation (in pressure) is performed using the data from SAGE and the GCM.

8.5.3.3 Fizhi Diagnostics

Fizhi Diagnostic Menu: [sec:pkg:fizhi:diagnostics]

NAME	UNITS	LEVELS	DESCRIPTION
UFLUX	N m^{-2}	1	Surface U-Wind Stress on the atmosphere
VFLUX	N m^{-2}	1	Surface V-Wind Stress on the atmosphere
HFLUX	W m^{-2}	1	Surface Flux of Sensible Heat
EFLUX	W m^{-2}	1	Surface Flux of Latent Heat
QICE	W m^{-2}	1	Heat Conduction through Sea-Ice
RADLWG	W m^{-2}	1	Net upward LW flux at the ground
RADSWG	W m^{-2}	1	Net downward SW flux at the ground
RI	dimensionless	Nrphys	Richardson Number
CT	dimensionless	1	Surface Drag coefficient for T and Q
CU	dimensionless	1	Surface Drag coefficient for U and V
ET	m^2 s^{-1}	Nrphys	Diffusivity coefficient for T and Q
EU	m^2 s^{-1}	Nrphys	Diffusivity coefficient for U and V
TURBU	m s^{-1} day^{-1}	Nrphys	U-Momentum Changes due to Turbulence
TURBV	m s^{-1} day^{-1}	Nrphys	V-Momentum Changes due to Turbulence
TURBT	deg day^{-1}	Nrphys	Temperature Changes due to Turbulence
TURBQ	g/kg/day	Nrphys	Specific Humidity Changes due to Turbulence
MOISTT	deg day^{-1}	Nrphys	Temperature Changes due to Moist Processes
MOISTQ	g/kg/day	Nrphys	Specific Humidity Changes due to Moist Processes
RADLW	deg day^{-1}	Nrphys	Net Longwave heating rate for each level
RADSW	deg day^{-1}	Nrphys	Net Shortwave heating rate for each level
PREACC	mm/day	1	Total Precipitation
PRECON	mm/day	1	Convective Precipitation
TUFLUX	N m^{-2}	Nrphys	Turbulent Flux of U-Momentum
TVFLUX	N m^{-2}	Nrphys	Turbulent Flux of V-Momentum
TTFLUX	W m^{-2}	Nrphys	Turbulent Flux of Sensible Heat

NAME	UNITS	LEVELS	DESCRIPTION
TQFLUX	W m^{-2}	Nrphys	Turbulent Flux of Latent Heat
CN	dimensionless	1	Neutral Drag Coefficient
WINDS	m s^{-1}	1	Surface Wind Speed
DTSRF	deg	1	Air/Surface virtual temperature difference
TG	deg	1	Ground temperature
TS	deg	1	Surface air temperature (Adiabatic from lowest model layer)
DTG	deg	1	Ground temperature adjustment
QG	g kg^{-1}	1	Ground specific humidity
QS	g kg^{-1}	1	Saturation surface specific humidity
TGRLW	deg	1	Instantaneous ground temperature used as input to the Longwave radiation subroutine
ST4	W m^{-2}	1	Upward Longwave flux at the ground (σT^4)
OLR	W m^{-2}	1	Net upward Longwave flux at the top of the model
OLRCLR	W m^{-2}	1	Net upward clearsky Longwave flux at the top of the model
LWGCLR	W m^{-2}	1	Net upward clearsky Longwave flux at the ground
LWCLR	deg day^{-1}	Nrphys	Net clearsky Longwave heating rate for each level
TLW	deg	Nrphys	Instantaneous temperature used as input to the Longwave radiation subroutine
SHLW	g g^{-1}	Nrphys	Instantaneous specific humidity used as input to the Longwave radiation subroutine
OZLW	g g^{-1}	Nrphys	Instantaneous ozone used as input to the Longwave radiation subroutine
CLMOLW	$0-1$	Nrphys	Maximum overlap cloud fraction used in the Longwave radiation subroutine
CLDTOT	$0-1$	Nrphys	Total cloud fraction used in the Longwave and Shortwave radiation subroutines
LWGDOWN	W m^{-2}	1	Downwelling Longwave radiation at the ground
GWDT	deg day^{-1}	Nrphys	Temperature tendency due to Gravity Wave Drag
RADSWT	W m^{-2}	1	Incident Shortwave radiation at the top of the atmosphere
TAUCLD	per 100 mb	Nrphys	Counted Cloud Optical Depth (non-dimensional) per 100 mb
TAUCLDC	Number	Nrphys	Cloud Optical Depth Counter

NAME	UNITS	LEVELS	Description
CLDLOW	0-1	Nrphys	Low-Level (1000-700 hPa) Cloud Fraction (0-1)
EVAP	mm/day	1	Surface evaporation
DPDT	hPa/day	1	Surface Pressure time-tendency
UAVE	m/sec	Nrphys	Average U-Wind
VAVE	m/sec	Nrphys	Average V-Wind
TAVE	deg	Nrphys	Average Temperature
QAVE	g/kg	Nrphys	Average Specific Humidity
OMEGA	hPa/day	Nrphys	Vertical Velocity
DUDT	m/sec/day	Nrphys	Total U-Wind tendency
DVDT	m/sec/day	Nrphys	Total V-Wind tendency
DTDT	deg/day	Nrphys	Total Temperature tendency
DQDT	g/kg/day	Nrphys	Total Specific Humidity tendency
VORT	10^{-4}/sec	Nrphys	Relative Vorticity
DTLS	deg/day	Nrphys	Temperature tendency due to Stratiform Cloud Formation
DQLS	g/kg/day	Nrphys	Specific Humidity tendency due to Stratiform Cloud Formation
USTAR	m/sec	1	Surface USTAR wind
Z0	m	1	Surface roughness
FRQTRB	0-1	Nrphys-1	Frequency of Turbulence
PBL	mb	1	Planetary Boundary Layer depth
SWCLR	deg/day	Nrphys	Net clearsky Shortwave heating rate for each level
OSR	W m^{-2}	1	Net downward Shortwave flux at the top of the model
OSRCLR	W m^{-2}	1	Net downward clearsky Shortwave flux at the top of the model
CLDMAS	kg / m^2	Nrphys	Convective cloud mass flux
UAVE	m/sec	Nrphys	Time-averaged u-Wind

NAME	UNITS	LEVELS	DESCRIPTION
VAVE	m/sec	Nrphys	Time-averaged v-Wind
TAVE	deg	Nrphys	Time-averaged Temperature‘
QAVE	g/g	Nrphys	Time-averaged Specific Humidity
RFT	deg/day	Nrphys	Temperature tendency due Rayleigh Friction
PS	mb	1	Surface Pressure
QQAVE	$(m/sec)^2$	Nrphys	Time-averaged Turbulent Kinetic Energy
SWGCLR	W m^{-2}	1	Net downward clearsky Shortwave flux at the ground
PAVE	mb	1	Time-averaged Surface Pressure
DIABU	m/sec/day	Nrphys	Total Diabatic forcing on u-Wind
DIABV	m/sec/day	Nrphys	Total Diabatic forcing on v-Wind
DIABT	deg/day	Nrphys	Total Diabatic forcing on Temperature
DIABQ	g/kg/day	Nrphys	Total Diabatic forcing on Specific Humidity
RFU	m/sec/day	Nrphys	U-Wind tendency due to Rayleigh Friction
RFV	m/sec/day	Nrphys	V-Wind tendency due to Rayleigh Friction
GWDU	m/sec/day	Nrphys	U-Wind tendency due to Gravity Wave Drag
GWDU	m/sec/day	Nrphys	V-Wind tendency due to Gravity Wave Drag
GWDUS	N m^{-2}	1	U-Wind Gravity Wave Drag Stress at Surface
GWDVS	N m^{-2}	1	V-Wind Gravity Wave Drag Stress at Surface
GWDUT	N m^{-2}	1	U-Wind Gravity Wave Drag Stress at Top
GWDVT	N m^{-2}	1	V-Wind Gravity Wave Drag Stress at Top
LZRAD	mg/kg	Nrphys	Estimated Cloud Liquid Water used in Radiation

NAME	UNITS	LEVELS	DESCRIPTION
SLP	mb	1	Time-averaged Sea-level Pressure
CLDFRC	0-1	1	Total Cloud Fraction
TPW	gm cm^{-2}	1	Precipitable water
U2M	m/sec	1	U-Wind at 2 meters
V2M	m/sec	1	V-Wind at 2 meters
T2M	deg	1	Temperature at 2 meters
Q2M	g/kg	1	Specific Humidity at 2 meters
U10M	m/sec	1	U-Wind at 10 meters
V10M	m/sec	1	V-Wind at 10 meters
T10M	deg	1	Temperature at 10 meters
Q10M	g/kg	1	Specific Humidity at 10 meters
DTRAIN	kg m^{-2}	Nrphys	Detrainment Cloud Mass Flux
QFILL	g/kg/day	Nrphys	Filling of negative specific humidity
DTCONV	deg/sec	Nr	Temp Change due to Convection
DQCONV	g/kg/sec	Nr	Specific Humidity Change due to Convection
RELHUM	percent	Nr	Relative Humidity
PRECLS	g/m^2/sec	1	Large Scale Precipitation
ENPREC	J/g	1	Energy of Precipitation (snow, rain Temp)

8.5.3.4 Fizhi Diagnostic Description

In this section we list and describe the diagnostic quantities available within the GCM. The diagnostics are listed in the order that they appear in the Diagnostic Menu, Section [sec:pkg:fizhi:diagnostics]. In all cases, each diagnostic as currently archived on the output datasets is time-averaged over its diagnostic output frequency:

$$\mathbf{DIAGNOSTIC} = \frac{1}{TTOT} \sum_{t=1}^{t=TTOT} diag(t)$$

where $TTOT = \frac{\mathbf{NQDIAG}}{\Delta t}$, **NQDIAG** is the output frequency of the diagnostic, and Δt is the timestep over which the diagnostic is updated.

Surface Zonal Wind Stress on the Atmosphere ($Newton/m^2$)

The zonal wind stress is the turbulent flux of zonal momentum from the surface.

$$\mathbf{UFLUX} = -\rho C_D W_s u \qquad where: \quad C_D = C_u^2$$

where ρ = the atmospheric density at the surface, C_D is the surface drag coefficient, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), W_s is the magnitude of the surface layer wind, and u is the zonal wind in the lowest model layer.

Surface Meridional Wind Stress on the Atmosphere ($Newton/m^2$)

The meridional wind stress is the turbulent flux of meridional momentum from the surface.

$$\mathbf{VFLUX} = -\rho C_D W_s v \qquad where: \quad C_D = C_u^2$$

where ρ = the atmospheric density at the surface, C_D is the surface drag coefficient, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), W_s is the magnitude of the surface layer wind, and v is the meridional wind in the lowest model layer.

Surface Flux of Sensible Heat (W m^{-2})

The turbulent flux of sensible heat from the surface to the atmosphere is a function of the gradient of virtual potential temperature and the eddy exchange coefficient:

$$\mathbf{HFLUX} = P^{\kappa}\rho c_p C_H W_s(\theta_{surface} - \theta_{Nrphys}) \qquad where: \quad C_H = C_u C_t$$

where ρ = the atmospheric density at the surface, c_p is the specific heat of air, C_H is the dimensionless surface heat transfer coefficient, W_s is the magnitude of the surface layer wind, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), C_t is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and θ is the potential temperature at the surface and at the bottom model level.

Surface Flux of Latent Heat ($Watts/m^2$)

The turbulent flux of latent heat from the surface to the atmosphere is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

$$\mathbf{EFLUX} = \rho\beta L C_H W_s(q_{surface} - q_{Nrphys}) \qquad where: \quad C_H = C_u C_t$$

where ρ = the atmospheric density at the surface, β is the fraction of the potential evapotranspiration actually evaporated, L is the latent heat of evaporation, C_H is the dimensionless surface heat transfer coefficient, W_s is the magnitude of the surface layer wind, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), C_t is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and $q_{surface}$ and q_{Nrphys} are the specific humidity at the surface and at the bottom model level, respectively.

Heat Conduction Through Sea Ice ($Watts/m^2$)

Over sea ice there is an additional source of energy at the surface due to the heat conduction from the relatively warm ocean through the sea ice. The heat conduction through sea ice represents an additional energy source term for the ground temperature equation.

$$\mathbf{QICE} = \frac{C_{ti}}{H_i}(T_i - T_g)$$

where C_{ti} is the thermal conductivity of ice, H_i is the ice thickness, assumed to be 3 m where sea ice is present, T_i is 273 degrees Kelvin, and T_g is the temperature of the sea ice.

NOTE: QICE is not available through model version 5.3, but is available in subsequent versions.

Net upward Longwave Flux at the surface ($Watts/m^2$)

$$\begin{aligned} \mathbf{RADLWG} &= F^{Net}_{LW,Nrphys+1} \\ &= F^{\uparrow}_{LW,Nrphys+1} - F^{\downarrow}_{LW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F^{\uparrow}_{LW}$ is the upward Longwave flux and $F^{\downarrow}_{LW}$ is the downward Longwave flux.

Net downard shortwave Flux at the surface ($Watts/m^2$)

$$\begin{aligned}\mathbf{RADSWG} = & \quad F^{Net}_{SW,Nrphys+1} \\ = & \quad F^{\downarrow}_{SW,Nrphys+1} - F^{\uparrow}_{SW,Nrphys+1}\end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F^{\downarrow}_{SW}$ is the downward Shortwave flux and $F^{\uparrow}_{SW}$ is the upward Shortwave flux.

Richardson number ($dimensionless$)

The non-dimensional stability indicator is the ratio of the buoyancy to the shear:

$$\mathbf{RI} = \frac{\frac{g}{\theta_v}\frac{\partial \theta_v}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2} = \frac{c_p \frac{\partial \theta_v}{\partial z}\frac{\partial P^\kappa}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2}$$

where we used the hydrostatic equation:

$$\frac{\partial \Phi}{\partial P^\kappa} = c_p \theta_v$$

Negative values indicate unstable buoyancy **AND** shear, small positive values (< 0.4) indicate dominantly unstable shear, and large positive values indicate dominantly stable stratification.

CT - Surface Exchange Coefficient for Temperature and Moisture (dimensionless)

The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

$$\mathbf{CT} = -\frac{(\overline{w'\theta'})}{u_* \Delta\theta} = -\frac{(\overline{w'q'})}{u_* \Delta q} = \frac{k}{(\psi_h + \psi_g)}$$

where ψ_h is the surface layer non-dimensional temperature change and ψ_g is the viscous sublayer non-dimensional temperature or moisture change:

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta \qquad and \qquad \psi_g = \frac{0.55(Pr^{2/3} - 0.2)}{\nu^{1/2}}(h_0 u_* - h_{0_{ref}} u_{*_{ref}})^{1/2}$$

and: $h_0 = 30 z_0$ with a maximum value over land of 0.01

ϕ_h is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to . k is the Von Karman constant, ζ is the non-dimensional stability parameter, Pr is the Prandtl number for air, ν is the molecular viscosity, z_0 is the surface roughness length, u_* is the surface stress velocity (see diagnostic number 67), and the subscript ref refers to a reference value.

CU - Surface Exchange Coefficient for Momentum (dimensionless)

The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

$$\mathbf{CU} = \frac{u_*}{W_s} = \frac{k}{\psi_m}$$

where ψ_m is the surface layer non-dimensional wind shear:

$$\psi_m = \int_{\zeta_0}^{\zeta} \frac{\phi_m}{\zeta} d\zeta$$

ϕ_m is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to . k is the Von Karman constant, ζ is the non-dimensional stability parameter, u_* is the surface stress velocity (see diagnostic number 67), and W_s is the magnitude of the surface layer wind.

ET - Diffusivity Coefficient for Temperature and Moisture (m^2/sec)

In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat or moisture flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient K_h times the negative of the gradient of potential temperature or moisture. In the [HL88] adaptation of this closure, K_h takes the form:

$$\mathbf{ET} = K_h = -\frac{(\overline{w'\theta'_v})}{\frac{\partial \theta_v}{\partial z}} = \begin{cases} q\,\ell\,S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_H(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * turbulent\ \ kinetic\ \ energy}$, q_e is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, ℓ is the master length scale related to the layer depth, S_H is a function of G_H and G_M, the dimensionless buoyancy and wind shear parameters, respectively, or a function of G_{H_e} and G_{M_e}, the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number.

For the detailed equations and derivations of the modified level 2.5 closure scheme, see [HL88].

In the surface layer, **ET** is the exchange coefficient for heat and moisture, in units of m/sec, given by:

$$\mathbf{ET_{Nrphys}} = C_t * u_* = C_H W_s$$

where C_t is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions (see diagnostic number 9), u_* is the surface friction velocity (see diagnostic number 67), C_H is the heat transfer coefficient, and W_s is the magnitude of the surface layer wind.

EU - Diffusivity Coefficient for Momentum (m^2/sec)

In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat momentum flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient K_m times the negative of the gradient of the u-wind. In the [HL88] adaptation of this closure, K_m takes the form:

$$\mathbf{EU} = K_m = -\frac{(\overline{u'w'})}{\frac{\partial U}{\partial z}} = \begin{cases} q\,\ell\,S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_M(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * turbulent\ \ kinetic\ \ energy}$, q_e is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, ℓ is the master length scale related to the layer depth, S_M is a function of G_H and G_M, the dimensionless buoyancy and wind shear parameters, respectively, or a function of G_{H_e} and G_{M_e}, the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number.

For the detailed equations and derivations of the modified level 2.5 closure scheme, see [HL88].

In the surface layer, **EU** is the exchange coefficient for momentum, in units of m/sec, given by:

$$\mathbf{EU_{Nrphys}} = C_u * u_* = C_D W_s$$

where C_u is the dimensionless exchange coefficient for momentum from the surface layer similarity functions (see diagnostic number 10), u_* is the surface friction velocity (see diagnostic number 67), C_D is the surface drag coefficient, and W_s is the magnitude of the surface layer wind.

TURBU - Zonal U-Momentum changes due to Turbulence (m/sec/day)

The tendency of U-Momentum due to turbulence is written:

$$\mathbf{TURBU} = \frac{\partial u}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{u'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of u-momentum in terms of K_m, and the equation has the form of a diffusion equation.

TURBV - Meridional V-Momentum changes due to Turbulence (m/sec/day)

The tendency of V-Momentum due to turbulence is written:

$$\mathbf{TURBV} = \frac{\partial v}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{v'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of v-momentum in terms of K_m, and the equation has the form of a diffusion equation.

TURBT - Temperature changes due to Turbulence (deg/day)

The tendency of temperature due to turbulence is written:

$$\mathbf{TURBT} = \frac{\partial T}{\partial t} = P^\kappa \frac{\partial \theta}{\partial t}_{turb} = P^\kappa \frac{\partial}{\partial z}(-\overline{w'\theta'}) = P^\kappa \frac{\partial}{\partial z}(K_h \frac{\partial \theta_v}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of temperature in terms of K_h, and the equation has the form of a diffusion equation.

TURBQ - Specific Humidity changes due to Turbulence (g/kg/day)

The tendency of specific humidity due to turbulence is written:

$$\mathbf{TURBQ} = \frac{\partial q}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{w'q'}) = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of temperature in terms of K_h, and the equation has the form of a diffusion equation.

MOISTT - Temperature Changes Due to Moist Processes (deg/day)

$$\mathbf{MOISTT} = \left.\frac{\partial T}{\partial t}\right|_c + \left.\frac{\partial T}{\partial t}\right|_{ls}$$

where:

$$\left.\frac{\partial T}{\partial t}\right|_c = R\sum_i \left(\alpha \frac{m_B}{c_p}\Gamma_s\right)_i \quad and \quad \left.\frac{\partial T}{\partial t}\right|_{ls} = \frac{L}{c_p}(q^* - q)$$

$$and$$

$$\Gamma_s = g\eta\frac{\partial s}{\partial p}$$

The subscript c refers to convective processes, while the subscript ls refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as s, the convective cloud base mass flux is given as m_B, and the cloud entrainment is given as η, which are explicitly defined in Section 8.5.3.2, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as α, while R is the rain re-evaporation adjustment.

MOISTQ - Specific Humidity Changes Due to Moist Processes (g/kg/day)

$$\mathbf{MOISTQ} = \left.\frac{\partial q}{\partial t}\right|_c + \left.\frac{\partial q}{\partial t}\right|_{ls}$$

where:

$$\left.\frac{\partial q}{\partial t}\right|_c = R\sum_i \left(\alpha \frac{m_B}{L}(\Gamma_h - \Gamma_s)\right)_i \quad and \quad \left.\frac{\partial q}{\partial t}\right|_{ls} = (q^* - q)$$

and

$$\Gamma_s = g\eta\frac{\partial s}{\partial p} \quad and \quad \Gamma_h = g\eta\frac{\partial h}{\partial p}$$

The subscript c refers to convective processes, while the subscript ls refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as s, the moist static energy is given as h, the convective cloud base mass flux is given as m_B, and the cloud entrainment is given as η, which are explicitly defined in Section 8.5.3.2, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as α, while R is the rain re-evaporation adjustment.

RADLW - Heating Rate due to Longwave Radiation (deg/day)

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, $F_{LW}^{clearsky}$, first. For a given cloud fraction, the clear line-of-sight probability $C(p,p')$ is computed from the current level pressure p to the model top pressure, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see Section [sec:fizhi:radcloud]). The cloudy-sky flux is then obtained as:

$$F_{LW} = C(p,p') \cdot F_{LW}^{clearsky},$$

Finally, the net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z}F_{LW}^{NET},$$

or

$$\mathbf{RADLW} = \frac{g}{c_p \pi}\frac{\partial}{\partial \sigma}F_{LW}^{NET}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, and

$$F_{LW}^{NET} = F_{LW}^{\uparrow} - F_{LW}^{\downarrow}$$

RADSW - Heating Rate due to Shortwave Radiation (deg/day)

The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section [sec:fizhi:radcloud]). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.

The heating rate due to Shortwave Radiation under cloudy skies is defined as:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F(cloudy)_{SW}^{NET} \cdot \mathrm{RADSWT},$$

or

$$\mathbf{RADSW} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F(cloudy)_{SW}^{NET} \cdot \mathrm{RADSWT}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, RADSWT is the true incident shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

$$F(cloudy)_{SW}^{Net} = F(cloudy)_{SW}^{\uparrow} - F(cloudy)_{SW}^{\downarrow}$$

PREACC - Total (Large-scale + Convective) Accumulated Precipition (mm/day)

For a change in specific humidity due to moist processes, Δq_{moist}, the vertical integral or total precipitable amount is given by:

$$\mathbf{PREACC} = \int_{surf}^{top} \rho \Delta q_{moist} dz = -\int_{surf}^{top} \Delta q_{moist} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{moist} dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day.

PRECON - Convective Precipition (mm/day)

For a change in specific humidity due to sub-grid scale cumulus convective processes, Δq_{cum}, the vertical integral or total precipitable amount is given by:

$$\mathbf{PRECON} = \int_{surf}^{top} \rho \Delta q_{cum} dz = -\int_{surf}^{top} \Delta q_{cum} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{cum} dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day.

TUFLUX - Turbulent Flux of U-Momentum (Newton/m^2)

The turbulent flux of u-momentum is calculated for $diagnostic \ \ purposes \ \ only$ from the eddy coefficient for momentum:

$$\mathbf{TUFLUX} = \rho(\overline{u'w'}) = \rho(-K_m \frac{\partial U}{\partial z})$$

where ρ is the air density, and K_m is the eddy coefficient.

TVFLUX - Turbulent Flux of V-Momentum (Newton/m^2)

The turbulent flux of v-momentum is calculated for $diagnostic\ \ purposes\ \ only$ from the eddy coefficient for momentum:

$$\mathbf{TVFLUX} = \rho(\overline{v'w'}) = \rho(-K_m \frac{\partial V}{\partial z})$$

where ρ is the air density, and K_m is the eddy coefficient.

TTFLUX - Turbulent Flux of Sensible Heat (Watts/m^2)

The turbulent flux of sensible heat is calculated for $diagnostic\ \ purposes\ \ only$ from the eddy coefficient for heat and moisture:

$$\mathbf{TTFLUX} = c_p \rho P^\kappa(\overline{w'\theta'}) = c_p \rho P^\kappa(-K_h \frac{\partial \theta_v}{\partial z})$$

where ρ is the air density, and K_h is the eddy coefficient.

TQFLUX - Turbulent Flux of Latent Heat (Watts/m^2)

The turbulent flux of latent heat is calculated for $diagnostic\ \ purposes\ \ only$ from the eddy coefficient for heat and moisture:

$$\mathbf{TQFLUX} = L\rho(\overline{w'q'}) = L\rho(-K_h \frac{\partial q}{\partial z})$$

where ρ is the air density, and K_h is the eddy coefficient.

CN - Neutral Drag Coefficient (dimensionless)

The drag coefficient for momentum obtained by assuming a neutrally stable surface layer:

$$\mathbf{CN} = \frac{k}{\ln(\frac{h}{z_0})}$$

where k is the Von Karman constant, h is the height of the surface layer, and z_0 is the surface roughness.

NOTE: CN is not available through model version 5.3, but is available in subsequent versions.

WINDS - Surface Wind Speed (meter/sec)

The surface wind speed is calculated for the last internal turbulence time step:

$$\mathbf{WINDS} = \sqrt{u^2_{Nrphys} + v^2_{Nrphys}}$$

where the subscript $Nrphys$ refers to the lowest model level.

The air/surface virtual temperature difference measures the stability of the surface layer:

$$\mathbf{DTSRF} = (\theta_{vNrphys+1} - \theta v_{Nrphys}) P^\kappa_{surf}$$

where

$$\theta_{vNrphys+1} = \frac{T_g}{P^{\kappa}_{surf}}(1 + .609 q_{Nrphys+1}) \qquad and \qquad q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

β is the surface potential evapotranspiration coefficient ($\beta = 1$ over oceans), $q^*(T_g, P_s)$ is the saturation specific humidity at the ground temperature and surface pressure, level $Nrphys$ refers to the lowest model level and level $Nrphys + 1$ refers to the surface.

TG - Ground Temperature (deg K)

The ground temperature equation is solved as part of the turbulence package using a backward implicit time differencing scheme:

$$\mathbf{TG}\ is\ obtained\ from:\ C_g \frac{\partial T_g}{\partial t} = R_{sw} - R_{lw} + Q_{ice} - H - LE$$

where R_{sw} is the net surface downward shortwave radiative flux, R_{lw} is the net surface upward longwave radiative flux, Q_{ice} is the heat conduction through sea ice, H is the upward sensible heat flux, LE is the upward latent heat flux, and C_g is the total heat capacity of the ground. C_g is obtained by solving a heat diffusion equation for the penetration of the diurnal cycle into the ground (), and is given by:

$$C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{(0.386 + 0.536W + 0.15W^2) 2x10^{-3} \frac{86400.}{2\pi}}\ .$$

Here, the thermal conductivity, λ, is equal to $2x10^{-3}\ \frac{ly}{sec}\frac{cm}{K}$, the angular velocity of the earth, ω, is written as $86400\ sec/day$ divided by $2\pi\ radians/day$, and the expression for C_s, the heat capacity per unit volume at the surface, is a function of the ground wetness, W.

TS - Surface Temperature (deg K)

The surface temperature estimate is made by assuming that the model's lowest layer is well-mixed, and therefore that θ is constant in that layer. The surface temperature is therefore:

$$\mathbf{TS} = \theta_{Nrphys} P^{\kappa}_{surf}$$

DTG - Surface Temperature Adjustment (deg K)

The change in surface temperature from one turbulence time step to the next, solved using the Ground Temperature Equation (see diagnostic number 30) is calculated:

$$\mathbf{DTG} = {T_g}^n - {T_g}^{n-1}$$

where superscript n refers to the new, updated time level, and the superscript $n - 1$ refers to the value at the previous turbulence time level.

QG - Ground Specific Humidity (g/kg)

The ground specific humidity is obtained by interpolating between the specific humidity at the lowest model level and the specific humidity of a saturated ground. The interpolation is performed using the potential evapotranspiration function:

$$\mathbf{QG} = q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

where β is the surface potential evapotranspiration coefficient ($\beta = 1$ over oceans), and $q^*(T_g, P_s)$ is the saturation specific humidity at the ground temperature and surface pressure.

QS - Saturation Surface Specific Humidity (g/kg)

The surface saturation specific humidity is the saturation specific humidity at the ground temprature and surface pressure:

$$\mathbf{QS} = q^*(T_g, P_s)$$

TGRLW - Instantaneous ground temperature used as input to the Longwave radiation subroutine (deg)

$$\mathbf{TGRLW} = T_g(\lambda, \phi, n)$$

where T_g is the model ground temperature at the current time step n.

ST4 - Upward Longwave flux at the surface (Watts/m^2)

$$\mathbf{ST4} = \sigma T^4$$

where σ is the Stefan-Boltzmann constant and T is the temperature.

OLR - Net upward Longwave flux at $p = p_{top}$ (Watts/m^2)

$$\mathbf{OLR} = F^{NET}_{LW,top}$$

where top indicates the top of the first model layer. In the GCM, p_{top} = 0.0 mb.

OLRCLR - Net upward clearsky Longwave flux at $p = p_{top}$ (Watts/m^2)

$$\mathbf{OLRCLR} = F(clearsky)^{NET}_{LW,top}$$

where top indicates the top of the first model layer. In the GCM, p_{top} = 0.0 mb.

LWGCLR - Net upward clearsky Longwave flux at the surface (Watts/m^2)

$$\begin{aligned} \mathbf{LWGCLR} = & \quad F(clearsky)^{Net}_{LW,Nrphys+1} \\ = & \quad F(clearsky)^{\uparrow}_{LW,Nrphys+1} - F(clearsky)^{\downarrow}_{LW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F(clearsky)^{\uparrow}_{LW}$ is the upward clearsky Longwave flux and the $F(clearsky)^{\downarrow}_{LW}$ is the downward clearsky Longwave flux.

LWCLR - Heating Rate due to Clearsky Longwave Radiation (deg/day)

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, $F_{LW}^{clearsky}$, first. For a given cloud fraction, the clear line-of-sight probability $C(p,p')$ is computed from the current level pressure p to the model top pressure, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see Section [sec:fizhi:radcloud]). The cloudy-sky flux is then obtained as:

$$F_{LW} = C(p,p') \cdot F_{LW}^{clearsky},$$

Thus, **LWCLR** is defined as the net longwave heating rate due to the vertical divergence of the clear-sky longwave radiative flux:

$$\frac{\partial \rho c_p T}{\partial t}_{clearsky} = -\frac{\partial}{\partial z} F(clearsky)_{LW}^{NET},$$

or

$$\mathbf{LWCLR} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F(clearsky)_{LW}^{NET}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, and

$$F(clearsky)_{LW}^{Net} = F(clearsky)_{LW}^{\uparrow} - F(clearsky)_{LW}^{\downarrow}$$

TLW - Instantaneous temperature used as input to the Longwave radiation subroutine (deg)

$$\mathbf{TLW} = T(\lambda, \phi, level, n)$$

where T is the model temperature at the current time step n.

SHLW - Instantaneous specific humidity used as input to the Longwave radiation subroutine (kg/kg)

$$\mathbf{SHLW} = q(\lambda, \phi, level, n)$$

where q is the model specific humidity at the current time step n.

OZLW - Instantaneous ozone used as input to the Longwave radiation subroutine (kg/kg)

$$\mathbf{OZLW} = \mathrm{OZ}(\lambda, \phi, level, n)$$

where OZ is the interpolated ozone data set from the climatological monthly mean zonally averaged ozone data set.

CLMOLW - Maximum Overlap cloud fraction used in LW Radiation (0-1)

CLMOLW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Longwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

$$\mathbf{CLMOLW} = CLMO_{RAS,LW}(\lambda, \phi, level)$$

CLDTOT - Total cloud fraction used in LW and SW Radiation (0-1)

CLDTOT is the time-averaged total cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Longwave and Shortwave Radiation packages. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

$$\mathbf{CLDTOT} = F_{RAS} + F_{LS}$$

where F_{RAS} is the time-averaged cloud fraction due to sub-grid scale convection, and F_{LS} is the time-averaged cloud fraction due to precipitating and non-precipitating large-scale moist processes.

CLMOSW - Maximum Overlap cloud fraction used in SW Radiation (0-1)

CLMOSW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Shortwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

$$\mathbf{CLMOSW} = CLMO_{RAS,SW}(\lambda, \phi, level)$$

CLROSW - Random Overlap cloud fraction used in SW Radiation (0-1)

CLROSW is the time-averaged random overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Shortwave Radiation algorithm. These are convective and large-scale clouds whose radiative characteristics are not assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

$$\mathbf{CLROSW} = CLRO_{RAS,LargeScale,SW}(\lambda, \phi, level)$$

RADSWT - Incident Shortwave radiation at the top of the atmosphere (Watts/m^2)

$$\mathbf{RADSWT} = \frac{S_0}{R_a^2} \cdot cos\phi_z$$

where S_0, is the extra-terrestial solar contant, R_a is the earth-sun distance in Astronomical Units, and $cos\phi_z$ is the cosine of the zenith angle. It should be noted that **RADSWT**, as well as **OSR** and **OSRCLR**, are calculated at the top of the atmosphere (p=0 mb). However, the **OLR** and **OLRCLR** diagnostics are currently calculated at $p = p_{top}$ (0.0 mb for the GCM).

EVAP - Surface Evaporation (mm/day)

The surface evaporation is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

$$\mathbf{EVAP} = \rho \beta K_h (q_{surface} - q_{Nrphys})$$

where ρ = the atmospheric density at the surface, β is the fraction of the potential evapotranspiration actually evaporated ($\beta = 1$ over oceans), K_h is the turbulent eddy exchange coefficient for heat and moisture at the surface in m/sec and $qsurface$ and q_{Nrphys} are the specific humidity at the surface (see diagnostic number 34) and at the bottom model level, respectively.

DUDT - Total Zonal U-Wind Tendency (m/sec/day)

DUDT is the total time-tendency of the Zonal U-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DUDT} = \frac{\partial u}{\partial t}_{Dynamics} + \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

DVDT - Total Zonal V-Wind Tendency (m/sec/day)

DVDT is the total time-tendency of the Meridional V-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DVDT} = \frac{\partial v}{\partial t}_{Dynamics} + \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

DTDT - Total Temperature Tendency (deg/day)

DTDT is the total time-tendency of Temperature due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\begin{aligned}\mathbf{DTDT} = & \frac{\partial T}{\partial t}_{Dynamics} + \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ + & \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis}\end{aligned}$$

DQDT - Total Specific Humidity Tendency (g/kg/day)

DQDT is the total time-tendency of Specific Humidity due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DQDT} = \frac{\partial q}{\partial t}_{Dynamics} + \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

USTAR - Surface-Stress Velocity (m/sec)

The surface stress velocity, or the friction velocity, is the wind speed at the surface layer top impeded by the surface drag:

$$\mathbf{USTAR} = C_u W_s \qquad where: \;\; C_u = \frac{k}{\psi_m}$$

C_u is the non-dimensional surface drag coefficient (see diagnostic number 10), and W_s is the surface wind speed (see diagnostic number 28).

Z0 - Surface Roughness Length (m)

Over the land surface, the surface roughness length is interpolated to the local time from the monthly mean data of . Over the ocean, the roughness length is a function of the surface-stress velocity, u_*.

$$\mathbf{Z0} = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + c_5 u_*$$

where the constants are chosen to interpolate between the reciprocal relation of for weak winds, and the piecewise linear relation of for moderate to large winds.

FRQTRB - Frequency of Turbulence (0-1)

The fraction of time when turbulence is present is defined as the fraction of time when the turbulent kinetic energy exceeds some minimum value, defined here to be $0.005\ m^2/sec^2$. When this criterion is met, a counter is incremented. The fraction over the averaging interval is reported.

PBL - Planetary Boundary Layer Depth (mb)

The depth of the PBL is defined by the turbulence parameterization to be the depth at which the turbulent kinetic energy reduces to ten percent of its surface value.

$$\mathbf{PBL} = P_{PBL} - P_{surface}$$

where P_{PBL} is the pressure in mb at which the turbulent kinetic energy reaches one tenth of its surface value, and P_s is the surface pressure.

SWCLR - Clear sky Heating Rate due to Shortwave Radiation (deg/day)

The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section [sec:fizhi:radcloud]). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.

The heating rate due to Shortwave Radiation under clear skies is defined as:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F(clear)_{SW}^{NET} \cdot \mathrm{RADSWT},$$

or

$$\mathbf{SWCLR} = \frac{g}{c_p}\frac{\partial}{\partial p} F(clear)_{SW}^{NET} \cdot \mathrm{RADSWT}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, RADSWT is the true incident shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

$$F(clear)_{SW}^{Net} = F(clear)_{SW}^{\uparrow} - F(clear)_{SW}^{\downarrow}$$

OSR - Net upward Shortwave flux at the top of the model (Watts/m^2)

$$\mathbf{OSR} = F_{SW,top}^{NET}$$

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW_{top}}$ = 0 mb.

OSRCLR - Net upward clearsky Shortwave flux at the top of the model (Watts/m^2)

$$\mathbf{OSRCLR} = F(clearsky)_{SW,top}^{NET}$$

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW_{top}}$ = 0 mb.

CLDMAS - Convective Cloud Mass Flux (kg/m^2)

The amount of cloud mass moved per RAS timestep from all convective clouds is written:

$$\mathbf{CLDMAS} = \eta m_B$$

where η is the entrainment, normalized by the cloud base mass flux, and m_B is the cloud base mass flux. m_B and η are defined explicitly in Section 8.5.3.2, the description of the convective parameterization.

UAVE - Time-Averaged Zonal U-Wind (m/sec)

The diagnostic **UAVE** is simply the time-averaged Zonal U-Wind over the **NUAVE** output frequency. This is contrasted to the instantaneous Zonal U-Wind which is archived on the Prognostic Output data stream.

$$\mathbf{UAVE} = u(\lambda, \phi, level, t)$$

Note, **UAVE** is computed and stored on the staggered C-grid.

VAVE - Time-Averaged Meridional V-Wind (m/sec)

The diagnostic **VAVE** is simply the time-averaged Meridional V-Wind over the **NVAVE** output frequency. This is contrasted to the instantaneous Meridional V-Wind which is archived on the Prognostic Output data stream.

$$\mathbf{VAVE} = v(\lambda, \phi, level, t)$$

Note, **VAVE** is computed and stored on the staggered C-grid.

TAVE - Time-Averaged Temperature (Kelvin)

The diagnostic **TAVE** is simply the time-averaged Temperature over the **NTAVE** output frequency. This is contrasted to the instantaneous Temperature which is archived on the Prognostic Output data stream.

$$\mathbf{TAVE} = T(\lambda, \phi, level, t)$$

QAVE - Time-Averaged Specific Humidity (g/kg)

The diagnostic **QAVE** is simply the time-averaged Specific Humidity over the **NQAVE** output frequency. This is contrasted to the instantaneous Specific Humidity which is archived on the Prognostic Output data stream.

$$\mathbf{QAVE} = q(\lambda, \phi, level, t)$$

PAVE - Time-Averaged Surface Pressure - PTOP (mb)

The diagnostic **PAVE** is simply the time-averaged Surface Pressure - PTOP over the **NPAVE** output frequency. This is contrasted to the instantaneous Surface Pressure - PTOP which is archived on the Prognostic Output data stream.

$$\begin{aligned} \mathbf{PAVE} = & \quad \pi(\lambda, \phi, level, t) \\ = & \quad p_s(\lambda, \phi, level, t) - p_T \end{aligned}$$

QQAVE - Time-Averaged Turbulent Kinetic Energy (m/sec)^2

The diagnostic **QQAVE** is simply the time-averaged prognostic Turbulent Kinetic Energy produced by the GCM Turbulence parameterization over the **NQQAVE** output frequency. This is contrasted to the instantaneous Turbulent Kinetic Energy which is archived on the Prognostic Output data stream.

$$\mathbf{QQAVE} = qq(\lambda, \phi, level, t)$$

Note, **QQAVE** is computed and stored at the "mass-point" locations on the staggered C-grid.

SWGCLR - Net downward clearsky Shortwave flux at the surface (Watts/m^2)

$$\begin{aligned}\mathbf{SWGCLR} = & \quad F(clearsky)^{Net}_{SW,Nrphys+1} \\ = & \quad F(clearsky)^{\downarrow}_{SW,Nrphys+1} - F(clearsky)^{\uparrow}_{SW,Nrphys+1}\end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F(clearsky)SW^{\downarrow}$ is the downward clearsky Shortwave flux and $F(clearsky)^{\uparrow}_{SW}$ is the upward clearsky Shortwave flux.

DIABU - Total Diabatic Zonal U-Wind Tendency (m/sec/day)

DIABU is the total time-tendency of the Zonal U-Wind due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABU} = \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

DIABV - Total Diabatic Meridional V-Wind Tendency (m/sec/day)

DIABV is the total time-tendency of the Meridional V-Wind due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABV} = \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

DIABT Total Diabatic Temperature Tendency (deg/day)

DIABT is the total time-tendency of Temperature due to Diabatic processes and the Analysis forcing.

$$\begin{aligned}\mathbf{DIABT} = & \quad \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ + & \quad \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis}\end{aligned}$$

If we define the time-tendency of Temperature due to Diabatic processes as

$$\begin{aligned}\frac{\partial T}{\partial t}_{Diabatic} = & \quad \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ + & \quad \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence}\end{aligned}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial T}{\partial t}_{Diabatic} = \frac{p^{\kappa}}{\pi} \frac{\partial \pi\theta}{\partial t}_{Diabatic}$$

where $\theta = T/p^\kappa$. Thus, **DIABT** may be written as

$$\mathbf{DIABT} = \frac{p^\kappa}{\pi}\left(\frac{\partial\pi\theta}{\partial t}_{Diabatic} + \frac{\partial\pi\theta}{\partial t}_{Analysis}\right)$$

DIABQ - Total Diabatic Specific Humidity Tendency (g/kg/day)

DIABQ is the total time-tendency of Specific Humidity due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABQ} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

If we define the time-tendency of Specific Humidity due to Diabatic processes as

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{1}{\pi}\frac{\partial\pi q}{\partial t}_{Diabatic}$$
$$Thus, **DIABQ**maybewrittenas$$

$$\mathbf{DIABQ} = \frac{1}{\pi}\left(\frac{\partial\pi q}{\partial t}_{Diabatic} + \frac{\partial\pi q}{\partial t}_{Analysis}\right)$$

VINTUQ - Vertically Integrated Moisture Flux (m/sec g/kg)

The vertically integrated moisture flux due to the zonal u-wind is obtained by integrating uq over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTUQ} = \frac{\int_{surf}^{top} uq\rho dz}{\int_{surf}^{top}\rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g} = -\frac{1}{g}\delta p$, we have

$$\mathbf{VINTUQ} = \int_0^1 uqdp$$

VINTVQ - Vertically Integrated Moisture Flux (m/sec g/kg)

The vertically integrated moisture flux due to the meridional v-wind is obtained by integrating vq over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTVQ} = \frac{\int_{surf}^{top} vq\rho dz}{\int_{surf}^{top}\rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g} = -\frac{1}{g}\delta p$, we have

$$\mathbf{VINTVQ} = \int_0^1 vqdp$$

VINTUT - Vertically Integrated Heat Flux (m/sec deg)

The vertically integrated heat flux due to the zonal u-wind is obtained by integrating uT over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTUT} = \frac{\int_{surf}^{top} uT\rho dz}{\int_{surf}^{top} \rho dz}$$

Or,

$$\mathbf{VINTUT} = \int_0^1 uTdp$$

VINTVT - Vertically Integrated Heat Flux (m/sec deg)

The vertically integrated heat flux due to the meridional v-wind is obtained by integrating vT over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTVT} = \frac{\int_{surf}^{top} vT\rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g}$, we have

$$\mathbf{VINTVT} = \int_0^1 vTdp$$

CLDFRC - Total 2-Dimensional Cloud Fracton (0-1)

If we define the time-averaged random and maximum overlapped cloudiness as CLRO and CLMO respectively, then the probability of clear sky associated with random overlapped clouds at any level is (1-CLRO) while the probability of clear sky associated with maximum overlapped clouds at any level is (1-CLMO). The total clear sky probability is given by (1-CLRO)*(1-CLMO), thus the total cloud fraction at each level may be obtained by 1-(1-CLRO)*(1-CLMO).

At any given level, we may define the clear line-of-site probability by appropriately accounting for the maximum and random overlap cloudiness. The clear line-of-site probability is defined to be equal to the product of the clear line-of-site probabilities associated with random and maximum overlap cloudiness. The clear line-of-site probability $C(p, p')$ associated with maximum overlap clouds, from the current pressure p to the model top pressure, $p' = p_{top}$, or the model surface pressure, $p' = p_{surf}$, is simply 1.0 minus the largest maximum overlap cloud value along the line-of-site, ie.

$$1 - MAX_p^{p'}\left(CLMO_p\right)$$

Thus, even in the time-averaged sense it is assumed that the maximum overlap clouds are correlated in the vertical. The clear line-of-site probability associated with random overlap clouds is defined to be the product of the clear sky probabilities at each level along the line-of-site, ie.

$$\prod_p^{p'}(1 - CLRO_p)$$

The total cloud fraction at a given level associated with a line- of-site calculation is given by

$$1 - \left(1 - MAX_p^{p'}\left[CLMO_p\right]\right)\prod_p^{p'}(1 - CLRO_p)$$

The 2-dimensional net cloud fraction as seen from the top of the atmosphere is given by

$$\mathbf{CLDFRC} = 1 - \left(1 - MAX_{l=l_1}^{Nrphys}\left[CLMO_l\right]\right) \prod_{l=l_1}^{Nrphys} (1 - CLRO_l)$$

For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

QINT - Total Precipitable Water (gm/cm^2)

The Total Precipitable Water is defined as the vertical integral of the specific humidity, given by:

$$\begin{aligned} \mathbf{QINT} = & \int_{surf}^{top} \rho q dz \\ = & \frac{\pi}{g} \int_0^1 q dp \end{aligned}$$

where we have used the hydrostatic relation $\rho \delta z = -\frac{\delta p}{g}$.

U2M Zonal U-Wind at 2 Meter Depth (m/sec)

The u-wind at the 2-meter depth is determined from the similarity theory:

$$\mathbf{U2M} = \frac{u_*}{k} \psi_{m_{2m}} \frac{u_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}} u_{sl}$$

where $\psi_m(2m)$ is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, $\mathbf{U2M}$ is undefined.

V2M - Meridional V-Wind at 2 Meter Depth (m/sec)

The v-wind at the 2-meter depth is a determined from the similarity theory:

$$\mathbf{V2M} = \frac{u_*}{k} \psi_{m_{2m}} \frac{v_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}} v_{sl}$$

where $\psi_m(2m)$ is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, $\mathbf{V2M}$ is undefined.

T2M - Temperature at 2 Meter Depth (deg K)

The temperature at the 2-meter depth is a determined from the similarity theory:

$$\mathbf{T2M} = P^\kappa \left(\frac{\theta *}{k}(\psi_{h_{2m}} + \psi_g) + \theta_{surf}\right) = P^\kappa \left(\theta_{surf} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(\theta_{sl} - \theta_{surf})\right)$$

where:

$$\theta_* = -\frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, $\mathbf{T2M}$ is undefined.

Q2M - Specific Humidity at 2 Meter Depth (g/kg)

The specific humidity at the 2-meter depth is determined from the similarity theory:

$$\mathbf{Q2M} = P^{\kappa}\frac{(}{q_*}k(\psi_{h_{2m}} + \psi_g) + q_{surf}) = P^{\kappa}(q_{surf} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(q_{sl} - q_{surf}))$$

where:

$$q_* = -\frac{(\overline{w'q'})}{u_*}$$

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, $\mathbf{Q2M}$ is undefined.

U10M - Zonal U-Wind at 10 Meter Depth (m/sec)

The u-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

$$\mathbf{U10M} = \frac{u_*}{k}\psi_{m_{10m}}\frac{u_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}}u_{sl}$$

where $\psi_m(10m)$ is the non-dimensional wind shear at ten meters, and the subscript sl refers to the height of the top of the surface layer.

V10M - Meridional V-Wind at 10 Meter Depth (m/sec)

The v-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

$$\mathbf{V10M} = \frac{u_*}{k}\psi_{m_{10m}}\frac{v_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}}v_{sl}$$

where $\psi_m(10m)$ is the non-dimensional wind shear at ten meters, and the subscript sl refers to the height of the top of the surface layer.

T10M - Temperature at 10 Meter Depth (deg K)

The temperature at the 10-meter depth is an interpolation between the surface potential temperature and the model lowest level potential temperature using the ratio of the non-dimensional temperature gradient at the two levels:

$$\mathbf{T10M} = P^{\kappa}(\frac{\theta*}{k}(\psi_{h_{10m}} + \psi_g) + \theta_{surf}) = P^{\kappa}(\theta_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(\theta_{sl} - \theta_{surf}))$$

where:

$$\theta_* = -\frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer.

Q10M - Specific Humidity at 10 Meter Depth (g/kg)

The specific humidity at the 10-meter depth is an interpolation between the surface specific humidity and the model lowest level specific humidity using the ratio of the non-dimensional temperature gradient at the two levels:

$$\mathbf{Q10M} = P^{\kappa}(\frac{q_*}{k}(\psi_{h_{10m}} + \psi_g) + q_{surf}) = P^{\kappa}(q_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(q_{sl} - q_{surf}))$$

where:

$$q_* = -\frac{(\overline{w'q'})}{u_*}$$

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer.

DTRAIN - Cloud Detrainment Mass Flux (kg/m^2)

The amount of cloud mass moved per RAS timestep at the cloud detrainment level is written:

$$\mathbf{DTRAIN} = \eta_{r_D} m_B$$

where r_D is the detrainment level, m_B is the cloud base mass flux, and η is the entrainment, defined in Section 8.5.3.2.

QFILL - Filling of negative Specific Humidity (g/kg/day)

Due to computational errors associated with the numerical scheme used for the advection of moisture, negative values of specific humidity may be generated. The specific humidity is checked for negative values after every dynamics timestep. If negative values have been produced, a filling algorithm is invoked which redistributes moisture from below. Diagnostic **QFILL** is equal to the net filling needed to eliminate negative specific humidity, scaled to a per-day rate:

$$\mathbf{QFILL} = q_{final}^{n+1} - q_{initial}^{n+1}$$

where

$$q^{n+1} = (\pi q)^{n+1}/\pi^{n+1}$$

8.5.3.5 Key subroutines, parameters and files

8.5.3.6 Dos and don'ts

8.5.3.7 Fizhi Reference

8.5.3.8 Experiments and tutorials that use fizhi

- Global atmosphere experiment with realistic SST and topography in fizhi-cs-32x32x10 verification directory.
- Global atmosphere aqua planet experiment in fizhi-cs-aqualev20 verification directory.

8.6 Ice and Sea Ice Packages

8.6.1 THSICE: The Thermodynamic Sea Ice Package

Important note: This document has been written by Stephanie Dutkiewicz and describes an earlier implementation of the sea-ice package. This needs to be updated to reflect the recent changes (JMC).

This thermodynamic ice model is based on the 3-layer model by Winton (2000). and the energy-conserving LANL CICE model (Bitz and Lipscomb, 1999). The model considers two equally thick ice layers; the upper layer has a variable specific heat resulting from brine pockets, the lower layer has a fixed heat capacity. A zero heat capacity snow layer lies above the ice. Heat fluxes at the top and bottom surfaces are used to calculate the change in ice and snow layer thickness. Grid cells of the ocean model are either fully covered in ice or are open water. There is a provision to parametrize ice fraction (and leads) in this package. Modifications are discussed in small font following the subroutine descriptions.

8.6.1.1 Key parameters and Routines

The ice model is called from *thermodynamics.F*, subroutine *ice_forcing.F* is called in place of *external_forcing_surf.F*.

In *ice_forcing.F*, we calculate the freezing potential of the ocean model surface layer of water:

$$\mathbf{frzmlt} = (T_f - SST)\frac{c_{sw}\rho_{sw}\Delta z}{\Delta t}$$

where c_{sw} is seawater heat capacity, ρ_{sw} is the seawater density, Δz is the ocean model upper layer thickness and Δt is the model (tracer) timestep. The freezing temperature, $T_f = \mu S$ is a function of the salinity.

1. Provided there is no ice present and **frzmlt** is less than 0, the surface tendencies of wind, heat and freshwater are calculated as usual (ie. as in *external_forcing_surf.F*).
2. If there is ice present in the grid cell we call the main ice model routine *ice_therm.F* (see below). Output from this routine gives net heat and freshwater flux affecting the top of the ocean.

Subroutine *ice_forcing.F* uses these values to find the sea surface tendencies in grid cells. When there is ice present, the surface stress tendencies are set to zero; the ice model is purely thermodynamic and the effect of ice motion on the sea-surface is not examined.

Relaxation of surface T and S is only allowed equatorward of **relaxlat** (see **DATA.ICE below**), and no relaxation is allowed under the ice at any latitude.

(Note that there is provision for allowing grid cells to have both open water and seaice; if **compact** is between 0 and 1)

subroutine ICE_FREEZE

This routine is called from *thermodynamics.F* after the new temperature calculation, *calc_gt.F*, but before *calc_gs.F*. In *ice_freeze.F*, any ocean upper layer grid cell with no ice cover, but with temperature below freezing, $T_f = \mu S$ has ice initialized. We calculate **frzmlt** from all the grid cells in the water column that have a temperature less than freezing. In this routine, any water below the surface that is below freezing is set to T_f. A call to *ice_start.F* is made if **frzmlt** > 0, and salinity tendancy is updated for brine release.

(There is a provision for fractional ice: In the case where the grid cell has less ice coverage than **icemaskmax** we allow *ice_start.F* to be called)

subroutine ICE_START

The energy available from freezing the sea surface is brought into this routine as **esurp**. The enthalpy of the 2 layers of any new ice is calculated as:

$$\begin{aligned} q_1 = & \quad -c_i * T_f + L_i \\ q_2 = & \quad -c_f T_{mlt} + c_i(T_{mlt} - Tf) + L_i(1 - \frac{T_{mlt}}{T_f}) \end{aligned}$$

where c_f is specific heat of liquid fresh water, c_i is the specific heat of fresh ice, L_i is latent heat of freezing, ρ_i is density of ice and T_{mlt} is melting temperature of ice with salinity of 1. The height of a new layer of ice is

$$h_{inew} = \frac{\mathbf{esurp}\Delta t}{qi_{0av}}$$

where $qi_{0av} = -\frac{\rho_i}{2}(q_1 + q_2)$.

The surface skin temperature T_s and ice temperatures T_1, T_2 and the sea surface temperature are set at T_f.

(There is provision for fractional ice: new ice is formed over open water; the first freezing in the cell must have a height of **himin0**; this determines the ice fraction **compact**. If there is already ice in the grid cell, the new ice must have the same height and the new ice fraction is

$$i_f = (1 - \hat{i_f})\frac{h_{inew}}{h_i}$$

where $\hat{i_f}$ is ice fraction from previous timestep and h_i is current ice height. Snow is redistributed over the new ice fraction. The ice fraction is not allowed to become larger than **iceMaskmax** and if the ice height is above **hihig** then freezing energy comes from the full grid cell, ice growth does not occur under orginal ice due to freezing water.)

subroutine ICE_THERM

The main subroutine of this package is *ice_therm.F* where the ice temperatures are calculated and the changes in ice and snow thicknesses are determined. Output provides the net heat and fresh water fluxes that force the top layer of the ocean model.

If the current ice height is less than **himin** then the ice layer is set to zero and the ocean model upper layer temperature is allowed to drop lower than its freezing temperature; and atmospheric fluxes are allowed to effect the grid cell. If the ice height is greater than **himin** we proceed with the ice model calculation.

We follow the procedure of Winton (1999) – see equations 3 to 21 – to calculate the surface and internal ice temperatures. The surface temperature is found from the balance of the flux at the surface F_s, the shortwave heat flux absorbed by the ice, **fswint**, and the upward conduction of heat through the snow and/or ice, F_u. We linearize F_s about the surface temperature, $\hat{T_s}$, at the previous timestep (where $\hat{}$ indicates the value at the previous timestep):

$$F_s(T_s) = F_s(\hat{T_s}) + \frac{\partial F_s(\hat{T_s})}{\partial T_s}(T_s - \hat{T_s})$$

where,

$$F_s = F_{sensible} + F_{latent} + F^{down}_{longwave} + F^{up}_{longwave} + (1 - \alpha)F_{shortwave}$$

and

$$\frac{dF_s}{dT} = \frac{dF_{sensible}}{dT} + \frac{dF_{latent}}{dT} + \frac{dF^{up}_{longwave}}{dT}.$$

F_s and $\frac{dF_s}{dT}$ are currently calculated from the **BULKF** package described separately, but could also be provided by an atmospheric model. The surface albedo is calculated from the ice height and/or surface temperature (see below, *srf_albedo.F*) and the shortwave flux absorbed in the ice is

$$\mathbf{fswint} = (1 - e^{\kappa_i h_i})(1 - \alpha)F_{shortwave}$$

where κ_i is bulk extinction coefficient.

The conductive flux to the surface is

$$F_u = K_{1/2}(T_1 - T_s)$$

where $K_{1/2}$ is the effective conductive coupling of the snow-ice layer between the surface and the mid-point of the upper layer of ice :math:‘ K_{1/2}=frac{4 K_i K_s}{K_s h_i + 4 K_i h_s} . *:math:‘K_i* and K_s are constant thermal conductivities of seaice and snow.

From the above equations we can develop a system of equations to find the skin surface temperature, T_s and the two ice layer temperatures (see Winton, 1999, for details). We solve these equations iteratively until the change in T_s is small. When the surface temperature is greater then the melting temperature of the surface, the temperatures are recalculated setting T_s to 0. The enthalpy of the ice layers are calculated in order to keep track of the energy in the ice model. Enthalpy is defined, here, as the energy required to melt a unit mass of seaice with temperature T. For the upper layer (1) with brine pockets and the lower fresh layer (2):

$$\begin{aligned} q_1 = &\quad -c_f T_f + c_i(T_f - T) + L_i(1 - \frac{T_f}{T}) \\ q_2 = &\quad -c_i T + L_i \end{aligned}$$

where c_f is specific heat of liquid fresh water, c_i is the specific heat of fresh ice, and L_i is latent heat of melting fresh ice.

From the new ice temperatures, we can calculate the energy flux at the surface available for melting (if T_s=0) and the energy at the ocean-ice interface for either melting or freezing.

$$\begin{aligned} E_{top} = &\quad (F_s - K_{1/2}(T_s - T_1))\Delta t \\ E_{bot} = &\quad (\frac{4K_i(T_2 - T_f)}{h_i} - F_b)\Delta t \end{aligned}$$

where F_b is the heat flux at the ice bottom due to the sea surface temperature variations from freezing. If T_{sst} is above freezing, $F_b = c_{sw}\rho_{sw}\gamma(T_{sst} - T_f)u^*$, γ is the heat transfer coefficient and $u^* = QQ$ is frictional velocity between ice and water. If T_{sst} is below freezing, $F_b = (T_f - T_{sst})c_f\rho_f\Delta z/\Delta t$ and set T_{sst} to T_f. We also include the energy from lower layers that drop below freezing, and set those layers to T_f.

If $E_{top} > 0$ we melt snow from the surface, if all the snow is melted and there is energy left, we melt the ice. If the ice is all gone and there is still energy left, we apply the left over energy to heating the ocean model upper layer (See Winton, 1999, equations 27-29). Similarly if $E_{bot} > 0$ we melt ice from the bottom. If all the ice is melted, the snow is melted (with energy from the ocean model upper layer if necessary). If $E_{bot} < 0$ we grow ice at the bottom

$$\Delta h_i = \frac{-E_{bot}}{(q_{bot}\rho_i)}$$

where $q_{bot} = -c_i T_f + L_i$ is the enthalpy of the new ice, The enthalpy of the second ice layer, q_2 needs to be modified:

$$q_2 = \frac{\hat{h_i}/2\hat{q_2} + \Delta h_i q_{bot}}{\hat{h_i}/2 + \Delta h_i}$$

If there is a ice layer and the overlying air temperature is below 0^oC then any precipitation, P joins the snow layer:

$$\Delta h_s = -P\frac{\rho_f}{\rho_s}\Delta t,$$

ρ_f and ρ_s are the fresh water and snow densities. Any evaporation, similarly, removes snow or ice from the surface. We also calculate the snow age here, in case it is needed for the surface albedo calculation (see *srf_albedo.F* below).

For practical reasons we limit the ice growth to **hilim** and snow is limited to **hslim**. We converts any ice and/or snow above these limits back to water, maintaining the salt balance. Note however, that heat is not conserved in this conversion; sea surface temperatures below the ice are not recalculated.

If the snow/ice interface is below the waterline, snow is converted to ice (see Winton, 1999, equations 35 and 36). The subroutine *new_layers_winton.F*, described below, repartitions the ice into equal thickness layers while conserving energy.

The subroutine *ice_therm.F* now calculates the heat and fresh water fluxes affecting the ocean model surface layer. The heat flux:

$$q_{net} = \mathbf{fswocn} - F_b - \frac{\mathbf{esurp}}{\Delta t}$$

is composed of the shortwave flux that has passed through the ice layer and is absorbed by the water, **fswocn**$= QQ$, the ocean flux to the ice F_b, and the surplus energy left over from the melting, **esurp**. The fresh water flux is determined from the amount of fresh water and salt in the ice/snow system before and after the timestep.

(There is a provision for fractional ice: If ice height is above **hihig** then all energy from freezing at sea surface is used only in the open water aparts of the cell (ie. F_b will only have the conduction term). The melt energy is partitioned by **frac_energy** between melting ice height and ice extent. However, once ice height drops below **himon0** then all energy melts ice extent.)

subroutine SFC_ALBEDO

The routine *ice_therm.F* calls this routine to determine the surface albedo. There are two calculations provided here:

1. from LANL CICE model

   $$\alpha = f_s\alpha_s + (1 - f_s)(\alpha_{i_{min}} + (\alpha_{i_{max}} - \alpha_{i_{min}})(1 - e^{-h_i/h_\alpha}))$$

   where f_s is 1 if there is snow, 0 if not; the snow albedo, α_s has two values depending on whether $T_s < 0$ or not; $\alpha_{i_{min}}$ and $\alpha_{i_{max}}$ are ice albedos for thin melting ice, and thick bare ice respectively, and h_α is a scale height.

2. From GISS model (Hansen et al 1983)

   $$\alpha = \alpha_i e^{-h_s/h_a} + \alpha_s(1 - e^{-h_s/h_a})$$

   where α_i is a constant albedo for bare ice, h_a is a scale height and α_s is a variable snow albedo.

   $$\alpha_s = \alpha_1 + \alpha_2 e^{-\lambda_a a_s}$$

   where α_1 is a constant, α_2 depends on T_s, a_s is the snow age, and λ_a is a scale frequency. The snow age is calculated in *ice_therm.F* and is given in equation 41 in Hansen et al (1983).

subroutine NEW_LAYERS_WINTON

The subroutine *new_layers_winton.F* repartitions the ice into equal thickness layers while conserving energy. We pass to this subroutine, the ice layer enthalpies after melting/growth and the new height of the ice layers. The ending layer height should be half the sum of the new ice heights from *ice_therm.F*. The enthalpies of the ice layers are adjusted accordingly to maintain total energy in the ice model. If layer 2 height is greater than layer 1 height then layer 2 gives ice to layer 1 and:

$$q_1 = f_1\hat{q_1} + (1 - f1)\hat{q_2}$$

where f_1 is the fraction of the new to old upper layer heights. T_1 will therefore also have changed. Similarly for when ice layer height 2 is less than layer 1 height, except here we need to to be careful that the new T_2 does not fall below the melting temperature.

Initializing subroutines

ice_init.F: Set ice variables to zero, or reads in pickup information from **pickup.ic** (which was written out in *checkpoint.F*)

ice_readparms.F: Reads **data.ice**

Diagnostic subroutines

ice_ave.F: Keeps track of means of the ice variables

ice_diags.F: Finds averages and writes out diagnostics

Common Blocks

ICE.h: Ice Varibles, also **relaxlat** and **startIceModel**

ICE_DIAGS.h: matrices for diagnostics: averages of fields from *ice_diags.F*

BULKF_ICE_CONSTANTS.h (in **BULKF** package): all the parameters need by the ice model

Input file DATA.ICE

Here we need to set **StartIceModel**: which is 1 if the model starts from no ice; and 0 if there is a pickup file with the ice matrices (**pickup.ic**) which is read in *ice_init.F* and written out in *checkpoint.F*. The parameter **relaxlat** defines the latitude poleward of which there is no relaxing of surface T or S to observations. This avoids the relaxation forcing the ice model at these high latitudes.

(Note: **hicemin** is set to 0 here. If the provision for allowing grid cells to have both open water and seaice is ever implemented, this would be greater than 0)

8.6.1.2 Important Notes

1. heat fluxes have different signs in the ocean and ice models.
2. **StartIceModel** must be changed in **data.ice**: 1 (if starting from no ice), 0 (if using pickup.ic file).

8.6.1.3 THSICE Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
SI_Fract|  1 |SM P    M1      |0-1             |Sea-Ice fraction  [0-1]
SI_Thick|  1 |SM PC197M1      |m               |Sea-Ice thickness (area weighted␣
 ↪average)
SI_SnowH|  1 |SM PC197M1      |m               |Snow thickness over Sea-Ice (area␣
 ↪weighted)
SI_Tsrf |  1 |SM  C197M1      |degC            |Surface Temperature over Sea-Ice␣
 ↪(area weighted)
SI_Tice1|  1 |SM  C197M1      |degC            |Sea-Ice Temperature, 1srt layer (area␣
 ↪weighted)
SI_Tice2|  1 |SM  C197M1      |degC            |Sea-Ice Temperature, 2nd  layer (area␣
 ↪weighted)
```

(continues on next page)

(continued from previous page)

```
SI_Qice1|  1 |SM  C198M1      |J/kg            |Sea-Ice enthalpy, 1srt layer (mass␣
↪weighted)
SI_Qice2|  1 |SM  C198M1      |J/kg            |Sea-Ice enthalpy, 2nd  layer (mass␣
↪weighted)
SIalbedo|  1 |SM PC197M1      |0-1             |Sea-Ice Albedo [0-1] (area weighted␣
↪average)
SIsnwAge|  1 |SM P    M1      |s               |snow age over Sea-Ice
SIsnwPrc|  1 |SM  C197M1      |kg/m^2/s        |snow precip. (+=dw) over Sea-Ice␣
↪(area weighted)
SIflxAtm|  1 |SM      M1      |W/m^2           |net heat flux from the Atmosphere␣
↪(+=dw)
SIfrwAtm|  1 |SM      M1      |kg/m^2/s        |fresh-water flux to the Atmosphere␣
↪(+=up)
SIflx2oc|  1 |SM      M1      |W/m^2           |heat flux out of the ocean (+=up)
SIfrw2oc|  1 |SM      M1      |m/s             |fresh-water flux out of the ocean␣
↪(+=up)
SIsaltFx|  1 |SM      M1      |psu.kg/m^2      |salt flux out of the ocean (+=up)
SItOcMxL|  1 |SM      M1      |degC            |ocean mixed layer temperature
SIsOcMxL|  1 |SM P    M1      |psu             |ocean mixed layer salinity
```

8.6.1.4 References

Bitz, C.M. and W.H. Lipscombe, 1999: An Energy-Conserving Thermodynamic Model of Sea Ice. *Journal of Geophysical Research*, 104, 15,669 – 15,677.

Hansen, J., G. Russell, D. Rind, P. Stone, A. Lacis, S. Lebedeff, R. Ruedy and L.Travis, 1983: Efficient Three-Dimensional Global Models for Climate Studies: Models I and II. *Monthly Weather Review*, 111, 609 – 662.

Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual. LACC-98-16v.2. (note: this documentation is no longer available as CICE has progressed to a very different version 3)

Winton, M, 2000: A reformulated Three-layer Sea Ice Model. *Journal of Atmospheric and Ocean Technology*, 17, 525 – 531.

8.6.1.5 Experiments and tutorials that use thsice

- Global atmosphere experiment in aim.5l_cs verification directory, input from input.thsice directory.
- Global ocean experiment in global_ocean.cs32x15 verification directory, input from input.thsice directory.

8.6.2 SEAICE Package

Authors: Martin Losch, Dimitris Menemenlis, An Nguyen, Jean-Michel Campin, Patrick Heimbach, Chris Hill and Jinlun Zhang

8.6.2.1 Introduction

Package seaice provides a dynamic and thermodynamic interactive sea ice model.

CPP options enable or disable different aspects of the package (Section 8.6.2.2). Run-time options, flags, filenames and field-related dates/times are set in `data.seaice` (Section 8.6.2.3). A description of key subroutines is given in Section 8.6.2.5. Available diagnostics output is listed in Section 8.6.2.6.

8.6.2.2 SEAICE configuration and compiling

Compile-time options

As with all MITgcm packages, SEAICE can be turned on or off at compile time (see Section 3.5)

- using the `packages.conf` file by adding `seaice` to it
- or using genmake2 adding `-enable=seaice` or `-disable=seaice` switches
- **required packages and CPP options**: seaice requires the external forcing package pkg/exf to be enabled; no additional CPP options are required.

Parts of the seaice code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in SEAICE_OPTIONS.h. Table 8.13 summarizes the most important ones. For more options see SEAICE_OPTIONS.h.

Table 8.13: Some of the most relevant CPP preporocessor flags in the seaice package.

CPP option	Default	Description"
SEAICE_DEBUG	#undef	enhance STDOUT for debugging
SEAICE_ALLOW_DYNAMICS	#define	sea ice dynamics code
SEAICE_CGRID	#define	LSR solver on C-grid (rather than original B-grid)
SEAICE_ALLOW_EVP	#define	enable use of EVP rheology solver
SEAICE_ALLOW_JFNK	#define	enable use of JFNK rheology solver
SEAICE_ALLOW_KRYLOV	#define	enable use of Krylov rheology solver
SEAICE_LSR_ZEBRA	#undef	use a coloring method for LSR solver
SEAICE_EXTERNAL_FLUXES	#define	use pkg/exf-computed fluxes as starting point
SEAICE_ZETA_SMOOTHREG	#define	use differentiable regularization for viscosities
SEAICE_DELTA_SMOOTHREG	#undef	use differentiable regularization for $1/\Delta$
SEAICE_ALLOW_BOTTOMDRAG	#undef	enable grounding parameterization for improved fastice in shallow seas
SEAICE_ITD	#undef	run with dynamical sea Ice Thickness Distribution (ITD)
SEAICE_VARIABLE_SALINITY	#undef	enable sea ice with variable salinity
ALLOW_SITRACER	#undef	enable sea ice tracer package
SEAICE_BICE_STRESS	#undef	B-grid only for backward compatiblity: turn on ice-stress on ocean
EXPLICIT_SSH_SLOPE	#undef	B-grid only for backward compatiblity: use ETAN for tilt computations rather than geostrophic velocities

8.6.2.3 Run-time parameters

Run-time parameters (see Table 8.14) are set in *data.seaice* (read in pkg/seaice/seaice_readparms.F).

Enabling the package

seaice package is switched on/off at runtime by setting useSEAICE = `.TRUE.` in `data.pkg`.

General flags and parameters

Table 8.14 lists most run-time parameters.

Table 8.14: Run-time parameters and default values

Name	Default value	Description
SEAICEwriteState	FALSE	write sea ice state to file
SEAICEuseDYNAMICS	TRUE	use dynamics
SEAICEuseJFNK	FALSE	use the JFNK-solver
SEAICEuseTEM	FALSE	use truncated ellipse method
SEAICEuseStrImpCpl	FALSE	use strength implicit coupling in LSR/JFNK
SEAICEuseMetricTerms	TRUE	use metric terms in dynamics
SEAICEuseEVPpickup	TRUE	use EVP pickups
SEAICEuseFluxForm	TRUE	use flux form for 2nd central difference advection scheme
SEAICErestoreUnderIce	FALSE	enable restoring to climatology under ice
SEAICEupdateOceanStress	TRUE	update ocean surface stress accounting for sea ice cover
SEAICEscaleSurfStress	TRUE	scale atmosphere and ocean-surface stress on ice by concentration (AREA)
SEAICEaddSnowMass	TRUE	in computing seaiceMass, add snow contribution
useHB87stressCoupling	FALSE	turn on ice-ocean stress coupling following
usePW79thermodynamics	TRUE	flag to turn off zero-layer-thermodynamics for testing
SEAICEadvHeff	TRUE	flag to turn off advection of scalar variable HEFF
SEAICEadvArea	TRUE	flag to turn off advection of scalar variable AREA
SEAICEadvSnow	TRUE	flag to turn off advection of scalar variable HSNOW
SEAICEadvSalt	TRUE	flag to turn off advection of scalar variable HSALT
SEAICEadvScheme	77	set advection scheme for seaice scalar state variables
SEAICEuseFlooding	TRUE	use flood-freeze algorithm
SEAICE_no_slip	FALSE	use no-slip boundary conditions instead of free-slip
SEAICE_deltaTtherm	dTtracerLev (1)	time step for seaice thermodynamics (s)
SEAICE_deltaTdyn	dTtracerLev (1)	time step for seaice dynamics (s)
SEAICE_deltaTevp	0.0	EVP sub-cycling time step (s); values > 0 turn on EVP
SEAICEuseEVPstar	FALSE	use modified EVP* instead of EVP, following [LKT+12]
SEAICEuseEVPrev	FALSE	"revisited form" variation on EVP*, following [BFLM13]
SEAICEnEVPstarSteps	unset	number of modified EVP* iterations
SEAICE_evpAlpha	unset	EVP* parameter (non-dim.), to replace 2*SEAICE_evpTauRelax/SEAICE_deltaTevp
SEAICE_evpBeta	unset	EVP* parameter (non-dim.), to replace SEAICE_deltaTdyn/SEAICE_deltaTevp
SEAICEaEVPcoeff	unset	largest stabilized frequency for adaptive EVP (non-dim.)
SEAICEaEVPcStar	4.0	aEVP multiple of stability factor (non-dim.), see [KDL16] $\alpha * \beta = c^* * \gamma$
SEAICEaEVPalphaMin	5.0	aEVP lower limit of alpha and beta (non-dim.), see [KDL16]
SEAICE_elasticParm	0.33333333	EVP parameter E_0 (non-dim.), sets relaxation timescale SEAICE_evpTauRelaxtau = SEAICE_elasticParm * SEAICE_deltaTdyn
SEAICE_evpTauRelax	dTtracerLev (1) * SEAICE_elasticParm	relaxation time scale T for EVP waves (s)
SEAICE_OLx	OLx - 2	overlap for LSR-solver or preconditioner, x-dimension
SEAICE_OLy	OLy - 2	overlap for LSR-solver or preconditioner, y-dimension
SEAICEnonLinIterMax	2/10	maximum number of non-linear (outer loop) iterations (LSR/JFNK)
SEAICElinearIterMax	1500/10	maximum number of linear iterations (LSR/JFNK)
SEAICE_JFNK_lsIter	(off)	start line search after "lsIter" Newton iterations
SEAICEnonLinTol	1.0E-05	non-linear tolerance parameter for JFNK solver

Continued on next page

Table 8.14 – continued from previous page

Name	Default value	Description
JFNKgamma_lin_min	0.10	minimum tolerance parameter for linear JFNK solver
JFNKgamma_lin_max	0.99	maximum tolerance parameter for linear JFNK solver
JFNKres_tFac	unset	tolerance parameter for FGMRES residual
SEAICE_JFNKepsilon	1.0E-06	step size for the FD-gradient in s/r seaice_jacvec
SEAICE_dumpFreq	dumpFreq	dump frequency (s)
SEAICE_dump_mdsio	TRUE	write snapshot using /pkg/mdsio
SEAICE_dump_mnc	FALSE	write snapshot using /pkg/mnc
SEAICE_initialHEFF	0.0	initial sea ice thickness averaged over grid cell (m)
SEAICE_drag	1.0E-03	air-ice drag coefficient (non-dim.)
OCEAN_drag	1.0E-03	air-ocean drag coefficient (non-dim.)
SEAICE_waterDrag	5.5E-03	water-ice drag coefficient (non-dim.)
SEAICE_dryIceAlb	0.75	winter sea ice albedo
SEAICE_wetIceAlb	0.66	summer sea ice albedo
SEAICE_drySnowAlb	0.84	dry snow albedo
SEAICE_wetSnowAlb	0.70	wet snow albedo
SEAICE_waterAlbedo	0.10	water albedo (not used if #define SEAICE_EXTERNAL_FLUXES)
SEAICE_strength	2.75E+04	sea ice strength constant P^* (N/m^2)
SEAICE_cStar	20.0	sea ice strength constant C^* (non-dim.)
SEAICE_rhoAir	1.3 (or pkg/exf value)	density of air (kg/m^3)
SEAICE_cpAir	1004.0 (or pkg/exf value)	specific heat of air (J/kg/K)
SEAICE_lhEvap	2.5E+06 (or pkg/exf value)	latent heat of evaporation (J/kg)
SEAICE_lhFusion	3.34E+05 (or pkg/exf value)	latent heat of fusion (J/kg)
SEAICE_dalton	1.75E-03	ice-ocean transfer coefficient for latent and sensible heat (non-dim.)
useMaykutSatVapPoly	FALSE	use Maykut polynomial to compute saturation vapor pressure
SEAICE_iceConduct	2.16560E+00	sea ice conductivity (W m^{-1} K^{-1})
SEAICE_snowConduct	3.10000E-01	snow conductivity (W m^{-1} K^{-1})
SEAICE_emissivity	0.970018 (or pkg/exf value)	longwave ocean surface emissivity (non-dim.)
SEAICE_snowThick	0.15	cutoff snow thickness to use snow albedo (m)
SEAICE_shortwave	0.30	ice penetration shortwave radiation factor (non-dim.)
SEAICE_saltFrac	0.0	salinity newly formed ice (as fraction of ocean surface salinity)
SEAICE_frazilFrac	1.0 (or computed from other parms)	frazil to sea ice conversion rate, as fraction (relative to the local freezing point of sea ice water)
SEAICEstressFactor	1.0	scaling factor for ice area in computing total ocean stress (non-dim.)
HeffFile	unset	filename for initial sea ice eff. thickness field HEFF (m)
AreaFile	unset	filename for initial fraction sea ice cover AREA (non-dim.)
HsnowFile	unset	filename for initial eff. snow thickness field HSNOW (m)
HsaltFile	unset	filename for initial eff. sea ice salinity field HSALT (g/m^2)
LSR_ERROR	1.0E-04	sets accuracy of LSR solver
DIFF1	0.0	parameter used in advect.F

Continued on next page

Table 8.14 – continued from previous page

Name	Default value	Description
HO	0.5	lead closing parameter h_0 (m); demarcation thickness between thick and thin ice which determines partition between vertical and lateral ice growth
MIN_ATEMP	-50.0	minimum air temperature (^{o}C)
MIN_LWDOWN	60.0	minimum downward longwave (W/m^2)
MIN_TICE	-50.0	minimum ice temperature (^{o}C)
IMAX_TICE	10	number of iterations for ice surface temperature solution
SEAICE_EPS	1.0E-10	a "small number" used in various routines
SEAICE_area_reg	1.0E-5	minimum concentration to regularize ice thickness
SEAICE_hice_reg	0.05	minimum ice thickness (m) for regularization
SEAICE_multDim	1	number of ice categories for thermodynamics
SEAICE_useMultDimSnow	TRUE	use same fixed pdf for snow as for multi-thickness-category ice

The following dynamical ice thickness distribution and ridging parameters in Table 8.15 are only active with #define SEAICE_ITD. All parameters are non-dimensional unless indicated.

Table 8.15: Thickness distribution and ridging parameters

Name	Default value	Description
useHibler79IceStrength	TRUE	use [Hib79] ice strength; do not use [Rot75] with #define SEAICE_ITD
SEAICEsimpleRidging	TRUE	use simple ridging a la [Hib79]
SEAICE_cf	17.0	scaling parameter of [Rot75] ice strength parameterization
SEAICEpartFunc	0	use partition function of [TRMC75]
SEAICEredistFunc	0	use redistribution function of [Hib80]
SEAICEridgingIterMax	10	maximum number of ridging sweeps
SEAICEshearParm	0.5	fraction of shear to be used for ridging
SEAICEgStar	0.15	max. ice conc. that participates in ridging [TRMC75]
SEAICEhStar	25.0	ridging parameter for [TRMC75], [LHMJ07]
SEAICEaStar	0.05	similar to SEAICEgStar for [LHMJ07] participation function
SEAICEmuRidging	3.0	similar to SEAICEhStar for [LHMJ07] ridging function
SEAICEmaxRaft	1.0	regularization parameter for rafting
SEAICEsnowFracRidge	0.5	fraction of snow that remains on ridged ice
SEAICEuseLinRemapITD	TRUE	use linear remapping scheme of [Lip01]
Hlimit	unset	nITD+1-array of ice thickness category limits (m)
Hlimit_c1, Hlimit_c2, Hlimit_c3	3.0, 15.0, 3.0	when Hlimit is not set, then these parameters determine Hlimit from a simple function following [Lip01]

8.6.2.4 Description

The MITgcm sea ice model is based on a variant of the viscous-plastic (VP) dynamic-thermodynamic sea ice model (Zhang and Hibler 1997 [ZH97]) first introduced in Hibler (1979) and Hibler (1980) [Hib79][Hib80]. In order to adapt this model to the requirements of coupled ice-ocean state estimation, many important aspects of the original code have been modified and improved, see Losch et al. (2010) [LMC+10]:

- the code has been rewritten for an Arakawa C-grid, both B- and C-grid variants are available; the C-grid code allows for no-slip and free-slip lateral boundary conditions;
- three different solution methods for solving the nonlinear momentum equations have been adopted: LSOR

(Zhang and Hibler 1997 [ZH97]), EVP (Hunke and Dukowicz 1997 [HD97]), JFNK (Lemieux et al. 2010 [LTSedlavcek+10], Losch et al. 2014 [LFLV14]);

- ice-ocean stress can be formulated as in Hibler and Bryan (1987) [HB87] or as in Campin et al. (2008) [CMF08];
- ice variables are advected by sophisticated, conservative advection schemes with flux limiting;
- growth and melt parameterizations have been refined and extended in order to allow for more stable automatic differentiation of the code.

The sea ice model is tightly coupled to the ocean compontent of the MITgcm. Heat, fresh water fluxes and surface stresses are computed from the atmospheric state and, by default, modified by the ice model at every time step.

The ice dynamics models that are most widely used for large-scale climate studies are the viscous-plastic (VP) model (Hilber 1979 [Hib79]), the cavitating fluid (CF) model (Flato and Hibler 1992 [FWDH92]), and the elastic-viscous-plastic (EVP) model (Hunke and Dukowicz 1997 [HD97]). Compared to the VP model, the CF model does not allow ice shear in calculating ice motion, stress, and deformation. EVP models approximate VP by adding an elastic term to the equations for easier adaptation to parallel computers. Because of its higher accuracy in plastic solution and relatively simpler formulation, compared to the EVP model, we decided to use the VP model as the default dynamic component of our ice model. To do this we extended the line successive over relaxation (LSOR) method of Zhang and Hibler (1997) [ZH97] for use in a parallel configuration. An EVP model and a free-drift implementation can be selected with run-time flags.

Compatibility with ice-thermodynamics package pkg/thsice

By default pkg/seaice includes the original so-called zero-layer thermodynamics with a snow cover as in the appendix of Semtner (1976) [Sem76]. The zero-layer thermodynamic model assumes that ice does not store heat and, therefore, tends to exaggerate the seasonal variability in ice thickness. This exaggeration can be significantly reduced by using Winton's (Winton 2000 [Win00]) three-layer thermodynamic model that permits heat storage in ice.

The Winton (2000) sea-ice thermodynamics have been ported to MITgcm; they currently reside under pkg/thsice, described in Section 8.6.1. It is fully compatible with the packages seaice and exf. When turned on together with seaice, the zero-layer thermodynamics are replaced by the Winton thermodynamics. In order to use package seaice with the thermodynamics of pkg/thsice, compile both packages and turn both package on in `data.pkg`; see an example in verification/global_ocean.cs32x15/input.icedyn. Note, that once thsice is turned on, the variables and diagnostics associated to the default thermodynamics are meaningless, and the diagnostics of thsice must be used instead.

Surface forcing

The sea ice model requires the following input fields: 10 m winds, 2 m air temperature and specific humidity, downward longwave and shortwave radiations, precipitation, evaporation, and river and glacier runoff. The sea ice model also requires surface temperature from the ocean model and the top level horizontal velocity. Output fields are surface wind stress, evaporation minus precipitation minus runoff, net surface heat flux, and net shortwave flux. The sea-ice model is global: in ice-free regions bulk formulae (by default computed in package exf) are used to estimate oceanic forcing from the atmospheric fields.

Dynamics

The momentum equation of the sea-ice model is

$$m\frac{D\mathbf{u}}{Dt} = -mf\mathbf{k}\times\mathbf{u} + \tau_{\text{air}} + \tau_{\text{ocean}} - m\nabla\phi(0) + \mathbf{F} \tag{8.2}$$

where $m = m_i + m_s$ is the ice and snow mass per unit area; $\mathbf{u} = u\mathbf{i} + v\mathbf{j}$ is the ice velocity vector; **i**, **j**, and **k** are unit vectors in the x, y, and z directions, respectively; f is the Coriolis parameter; τ_{air} and τ_{ocean} are the wind-ice and ocean-ice stresses, respectively; g is the gravity accelation; $\nabla\phi(0)$ is the gradient (or tilt) of the sea surface height; $\phi(0) = g\eta + p_a/\rho_0 + mg/\rho_0$ is the sea surface height potential in response to ocean dynamics ($g\eta$), to atmospheric pressure loading (p_a/ρ_0, where ρ_0 is a reference density) and a term due to snow and ice loading ; and $\mathbf{F} = \nabla \cdot \sigma$ is the divergence of the internal ice stress tensor σ_{ij}. Advection of sea-ice momentum is neglected. The wind and ice-ocean stress terms are given by

$$\begin{aligned}\tau_{\text{air}} =& \rho_{\text{air}} C_{\text{air}} |\mathbf{U}_{\text{air}} - \mathbf{u}| R_{\text{air}} (\mathbf{U}_{\text{air}} - \mathbf{u}) \\ \tau_{\text{ocean}} =& \rho_{\text{ocean}} C_{\text{ocean}} |\mathbf{U}_{\text{ocean}} - \mathbf{u}| R_{\text{ocean}} (\mathbf{U}_{\text{ocean}} - \mathbf{u})\end{aligned}$$

where $\mathbf{U}_{\text{air/ocean}}$ are the surface winds of the atmosphere and surface currents of the ocean, respectively; $C_{\text{air/ocean}}$ are air and ocean drag coefficients; $\rho_{\text{air/ocean}}$ are reference densities; and $R_{\text{air/ocean}}$ are rotation matrices that act on the wind/current vectors.

Viscous-Plastic (VP) Rheology

For an isotropic system the stress tensor σ_{ij} $(i, j = 1, 2)$ can be related to the ice strain rate and strength by a nonlinear viscous-plastic (VP) constitutive law:

$$\sigma_{ij} = 2\eta(\dot{\epsilon}_{ij}, P)\dot{\epsilon}_{ij} + [\zeta(\dot{\epsilon}_{ij}, P) - \eta(\dot{\epsilon}_{ij}, P)]\, \dot{\epsilon}_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij} \tag{8.3}$$

The ice strain rate is given by

$$\dot{\epsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

The maximum ice pressure $P_{\max}$, a measure of ice strength, depends on both thickness h and compactness (concentration) c:

$$P_{\max} = P^* c\, h\, \exp\{-C^* \cdot (1 - c)\}, \tag{8.4}$$

with the constants P^* (run-time parameter SEAICE_strength) and C^* (run-time parameter SEAICE_cStar). The nonlinear bulk and shear viscosities η and ζ are functions of ice strain rate invariants and ice strength such that the principal components of the stress lie on an elliptical yield curve with the ratio of major to minor axis e equal to 2; they are given by:

$$\begin{aligned}\zeta =& \min\left(\frac{P_{\max}}{2\max(\Delta, \Delta_{\min})}, \zeta_{\max}\right) \\ \eta =& \frac{\zeta}{e^2}\end{aligned}$$

with the abbreviation

$$\Delta = \left[(\dot{\epsilon}_{11} + \dot{\epsilon}_{22})^2 + e^{-2}\left((\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + \dot{\epsilon}_{12}^2\right)\right]^{\frac{1}{2}}$$

The bulk viscosities are bounded above by imposing both a minimum $\Delta_{\min}$ (for numerical reasons, run-time parameter SEAICE_deltaMin is set to a default value of $10^{-10}\,\text{s}^{-1}$, the value of SEAICE_EPS) and a maximum $\zeta_{\max} = P_{\max}/(2\Delta^*)$, where $\Delta^* = (2 \times 10^4/5 \times 10^{12})\,\text{s}^{-1} = 2 \times 10^{-9}\,\text{s}^{-1}$. Obviously, this corresponds to regularizing Δ with the typical value of SEAICE_deltaMin $= 2 \times 10^{-9}$. Clearly, some of this regularization is redundant. (There is also the option of bounding ζ from below by setting run-time parameter SEAICE_zetaMin > 0, but this is generally not recommended). For stress tensor computation the replacement pressure $P = 2\,\Delta\zeta$ is used so that the stress state always lies on the elliptic yield curve by definition.

Defining the CPP-flag SEAICE_ZETA_SMOOTHREG in SEAICE_OPTIONS.h before compiling replaces the method for bounding ζ by a smooth (differentiable) expression:

$$\begin{aligned}\zeta &= \zeta_{\max} \tanh\left(\frac{P}{2\,\min(\Delta,\Delta_{\min})\,\zeta_{\max}}\right) \\ &= \frac{P}{2\Delta^*} \tanh\left(\frac{\Delta^*}{\min(\Delta,\Delta_{\min})}\right)\end{aligned} \tag{8.5}$$

where $\Delta_{\min} = 10^{-20}\,\mathrm{s}^{-1}$ should be chosen to avoid divisions by zero.

LSR and JFNK solver

In matrix notation, the discretized momentum equations can be written as

$$\mathbf{A}(\mathbf{x})\,\mathbf{x} = \mathbf{b}(\mathbf{x}). \tag{8.6}$$

The solution vector $\mathbf{x}$ consists of the two velocity components u and v that contain the velocity variables at all grid points and at one time level. The standard (and default) method for solving Eq. (8.6) in the sea ice component of MITgcm is an iterative Picard solver: in the k-th iteration a linearized form $\mathbf{A}(\mathbf{x}^{k-1})\,\mathbf{x}^k = \mathbf{b}(\mathbf{x}^{k-1})$ is solved (in the case of MITgcm it is a Line Successive (over) Relaxation (LSR) algorithm). Picard solvers converge slowly, but in practice the iteration is generally terminated after only a few nonlinear steps and the calculation continues with the next time level. This method is the default method in MITgcm. The number of nonlinear iteration steps or pseudo-time steps can be controlled by the run-time parameter SEAICEnonLinIterMax (default is 2).

In order to overcome the poor convergence of the Picard-solver, Lemieux et al. (2010) [LTSedlavcek+10] introduced a Jacobian-free Newton-Krylov solver for the sea ice momentum equations. This solver is also implemented in MITgcm (see Losch et al. 2014 [LFLV14]). The Newton method transforms minimizing the residual $\mathbf{F}(\mathbf{x}) = \mathbf{A}(\mathbf{x})\,\mathbf{x} - \mathbf{b}(\mathbf{x})$ to finding the roots of a multivariate Taylor expansion of the residual $\mathbf{F}$ around the previous $(k-1)$ estimate $\mathbf{x}^{k-1}$:

$$\mathbf{F}(\mathbf{x}^{k-1} + \delta\mathbf{x}^k) = \mathbf{F}(\mathbf{x}^{k-1}) + \mathbf{F}'(\mathbf{x}^{k-1})\,\delta\mathbf{x}^k \tag{8.7}$$

with the Jacobian $\mathbf{J} \equiv \mathbf{F}'$. The root $\mathbf{F}(\mathbf{x}^{k-1} + \delta\mathbf{x}^k) = 0$ is found by solving

$$\mathbf{J}(\mathbf{x}^{k-1})\,\delta\mathbf{x}^k = -\mathbf{F}(\mathbf{x}^{k-1}) \tag{8.8}$$

for $\delta\mathbf{x}^k$. The next (k-th) estimate is given by $\mathbf{x}^k = \mathbf{x}^{k-1} + a\,\delta\mathbf{x}^k$. In order to avoid overshoots the factor a is iteratively reduced in a line search ($a = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$) until $\|\mathbf{F}(\mathbf{x}^k)\| < \|\mathbf{F}(\mathbf{x}^{k-1})\|$, where $\|\cdot\| = \int \cdot\, dx^2$ is the L_2-norm. In practice, the line search is stopped at $a = \frac{1}{8}$. The line search starts after SEAICE_JFNK_lsIter nonlinear Newton iterations (off by default).

Forming the Jacobian $\mathbf{J}$ explicitly is often avoided as "too error prone and time consuming". Instead, Krylov methods only require the action of $\mathbf{J}$ on an arbitrary vector $\mathbf{w}$ and hence allow a matrix free algorithm for solving (8.8). The action of $\mathbf{J}$ can be approximated by a first-order Taylor series expansion:

$$\mathbf{J}(\mathbf{x}^{k-1})\,\mathbf{w} \approx \frac{\mathbf{F}(\mathbf{x}^{k-1} + \epsilon\mathbf{w}) - \mathbf{F}(\mathbf{x}^{k-1})}{\epsilon} \tag{8.9}$$

or computed exactly with the help of automatic differentiation (AD) tools. SEAICE_JFNKepsilon sets the step size ϵ.

We use the Flexible Generalized Minimum RESidual (FMGRES) method with right-hand side preconditioning to solve (8.8) iteratively starting from a first guess of $\delta\mathbf{x}_0^k = 0$. For the preconditioning matrix $\mathbf{P}$ we choose a simplified form of the system matrix $\mathbf{A}(\mathbf{x}^{k-1})$ where $\mathbf{x}^{k-1}$ is the estimate of the previous Newton step $k-1$. The transformed equation (8.8) becomes

$$\mathbf{J}(\mathbf{x}^{k-1})\,\mathbf{P}^{-1}\delta\mathbf{z} = -\mathbf{F}(\mathbf{x}^{k-1}), \quad \text{with} \quad \delta\mathbf{z} = \mathbf{P}\delta\mathbf{x}^k \tag{8.10}$$

The Krylov method iteratively improves the approximate solution to (8.10) in subspace ($\mathbf{r}_0$, $\mathbf{JP}^{-1}\mathbf{r}_0$, $(\mathbf{JP}^{-1})^2\mathbf{r}_0$, ..., $(\mathbf{JP}^{-1})^m\mathbf{r}_0$) with increasing m; $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}^{k-1}) - \mathbf{J}(\mathbf{x}^{k-1})\,\delta\mathbf{x}_0^k$ is the initial residual of (8.8); $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}^{k-1})$ with the first guess $\delta\mathbf{x}_0^k = 0$. We allow a Krylov-subspace of dimension $m = 50$ and we do allow restarts for more than 50 Krylov iterations. The preconditioning operation involves applying $\mathbf{P}^{-1}$ to the basis vectors $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ of the Krylov subspace. This operation is approximated by solving the linear system $\mathbf{P}\,\mathbf{w} = \mathbf{v}_i$. Because $\mathbf{P} \approx \mathbf{A}(\mathbf{x}^{k-1})$, we can use the LSR-algorithm already implemented in the Picard solver. Each preconditioning operation uses a fixed number of 10 LSR-iterations avoiding any termination criterion. More details and results can be found in Losch et al. (2014) [LFLV14]).

To use the JFNK-solver set SEAICEuseJFNK = `.TRUE.,` in the namelist file `data.seaice`; `#define` SEAICE_ALLOW_JFNK in SEAICE_OPTIONS.h and we recommend using a smooth regularization of ζ by `#define` SEAICE_ZETA_SMOOTHREG (see above) for better convergence. The nonlinear Newton iteration is terminated when the L_2-norm of the residual is reduced by γ_{nl} (run-time parameter SEAICEnonLinTol = `1.E-4,` will already lead to expensive simulations) with respect to the initial norm: $\|\mathbf{F}(\mathbf{x}^k)\| < \gamma_{\mathrm{nl}}\|\mathbf{F}(\mathbf{x}^0)\|$. Within a nonlinear iteration, the linear FGMRES solver is terminated when the residual is smaller than $\gamma_k\|\mathbf{F}(\mathbf{x}^{k-1})\|$ where γ_k is determined by

$$\gamma_k = \begin{cases} \gamma_0 & \text{for } \|\mathbf{F}(\mathbf{x}^{k-1})\| \geq r, \\ \max\left(\gamma_{\min}, \frac{\|\mathbf{F}(\mathbf{x}^{k-1})\|}{\|\mathbf{F}(\mathbf{x}^{k-2})\|}\right) & \text{for } \|\mathbf{F}(\mathbf{x}^{k-1})\| < r, \end{cases} \tag{8.11}$$

so that the linear tolerance parameter γ_k decreases with the nonlinear Newton step as the nonlinear solution is approached. This inexact Newton method is generally more robust and computationally more efficient than exact methods. Typical parameter choices are $\gamma_0 =$ JFNKgamma_lin_max $= 0.99$, $\gamma_{\min} =$ JFNKgamma_lin_min $= 0.1$, and $r =$ JFNKres_tFac $\times\|\mathbf{F}(\mathbf{x}^0)\|$ with JFNKres_tFac $= 0.5$. We recommend a maximum number of nonlinear iterations SEAICEnewtonIterMax $= 100$ and a maximum number of Krylov iterations SEAICEkrylovIterMax $= 50$, because the Krylov subspace has a fixed dimension of 50 (but restarts are allowed for SEAICEkrylovIterMax > 50).

Setting SEAICEuseStrImpCpl = `.TRUE.,` turns on "strength implicit coupling" (see Hutchings et al. 2004 [HJL04]) in the LSR-solver and in the LSR-preconditioner for the JFNK-solver. In this mode, the different contributions of the stress divergence terms are reordered so as to increase the diagonal dominance of the system matrix. Unfortunately, the convergence rate of the LSR solver is increased only slightly, while the JFNK-convergence appears to be unaffected.

Elastic-Viscous-Plastic (EVP) Dynamics

Hunke and Dukowicz (1997) [HD97] introduced an elastic contribution to the strain rate in order to regularize (8.3) in such a way that the resulting elastic-viscous-plastic (EVP) and VP models are identical at steady state,

$$\frac{1}{E}\frac{\partial\sigma_{ij}}{\partial t} + \frac{1}{2\eta}\sigma_{ij} + \frac{\eta-\zeta}{4\zeta\eta}\sigma_{kk}\delta_{ij} + \frac{P}{4\zeta}\delta_{ij} = \dot{\epsilon}_{ij}. \tag{8.12}$$

The EVP-model uses an explicit time stepping scheme with a short timestep. According to the recommendation in Hunke and Dukowicz (1997) [HD97], the EVP-model should be stepped forward in time 120 times (SEAICE_deltaTevp = SEAICE_deltaTdyn /120) within the physical ocean model time step (although this parameter is under debate), to allow for elastic waves to disappear. Because the scheme does not require a matrix inversion it is fast in spite of the small internal timestep and simple to implement on parallel computers. For completeness, we repeat the equations for the components of the stress tensor $\sigma_1 = \sigma_{11} + \sigma_{22}$, $\sigma_2 = \sigma_{11} - \sigma_{22}$, and σ_{12}. Introducing the divergence $D_D = \dot{\epsilon}_{11} + \dot{\epsilon}_{22}$, and the horizontal tension and shearing strain rates, $D_T = \dot{\epsilon}_{11} - \dot{\epsilon}_{22}$ and $D_S = 2\dot{\epsilon}_{12}$, respectively, and using the above abbreviations, the equations (8.12) can be written as:

$$\frac{\partial\sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P}{2T} = \frac{P}{2T\Delta}D_D \tag{8.13}$$

$$\frac{\partial\sigma_2}{\partial t} + \frac{\sigma_2 e^2}{2T} = \frac{P}{2T\Delta}D_T \tag{8.14}$$

$$\frac{\partial\sigma_{12}}{\partial t} + \frac{\sigma_{12}e^2}{2T} = \frac{P}{4T\Delta}D_S \tag{8.15}$$

Here, the elastic parameter E is redefined in terms of a damping timescale T for elastic waves

$$E = \frac{\zeta}{T}$$

$T = E_0 \Delta t$ with the tunable parameter $E_0 < 1$ and the external (long) timestep Δt. $E_0 = \frac{1}{3}$ is the default value in the code and close to what and recommend.

To use the EVP solver, make sure that both `#define` SEAICE_CGRID and `#define` SEAICE_ALLOW_EVP are set in SEAICE_OPTIONS.h (both are defined by default). The solver is turned on by setting the sub-cycling time step SEAICE_deltaTevp to a value larger than zero. The choice of this time step is under debate. Hunke and Dukowicz (1997) [HD97] recommend order 120 time steps for the EVP solver within one model time step Δt (deltaTmom). One can also choose order 120 time steps within the forcing time scale, but then we recommend adjusting the damping time scale T accordingly, by setting either SEAICE_elasticParm (E_0), so that $E_0 \Delta t =$ forcing time scale, or directly SEAICE_evpTauRelax (T) to the forcing time scale. (**NOTE**: with the improved EVP variants of the next section, the above recommendations are obsolete. Use mEVP or aEVP instead.)

More stable variants of Elastic-Viscous-Plastic Dynamics: EVP* , mEVP, and aEVP

The genuine EVP scheme appears to give noisy solutions (see Hunke 2001, Lemieux et al. 2012, Bouillon et a1. 2013 [Hun01][LKT+12][BFLM13]). This has led to a modified EVP or EVP* (Lemieux et al. 2012, Bouillon et a1. 2013, Kimmritz et al. 2015 [LKT+12][BFLM13][KDL15]); here, we refer to these variants by modified EVP (mEVP) and adaptive EVP (aEVP). The main idea is to modify the "natural" time-discretization of the momentum equations:

$$m \frac{D\mathbf{u}}{Dt} \approx m \frac{\mathbf{u}^{p+1} - \mathbf{u}^n}{\Delta t} + \beta^* \frac{\mathbf{u}^{p+1} - \mathbf{u}^p}{\Delta t_{\mathrm{EVP}}} \tag{8.16}$$

where n is the previous time step index, and p is the previous sub-cycling index. The extra "intertial" term $m\,(\mathbf{u}^{p+1} - \mathbf{u}^n)/\Delta t)$ allows the definition of a residual $|\mathbf{u}^{p+1} - \mathbf{u}^p|$ that, as $\mathbf{u}^{p+1} \to \mathbf{u}^{n+1}$, converges to 0. In this way EVP can be re-interpreted as a pure iterative solver where the sub-cycling has no association with time-relation (through Δt_{EVP}). Using the terminology of Kimmritz et al. 2015 [KDL15], the evolution equations of stress σ_{ij} and momentum $\mathbf{u}$ can be written as:

$$\sigma_{ij}^{p+1} = \sigma_{ij}^p + \frac{1}{\alpha}\Big(\sigma_{ij}(\mathbf{u}^p) - \sigma_{ij}^p\Big), \tag{8.17}$$

$$\mathbf{u}^{p+1} = \mathbf{u}^p + \frac{1}{\beta}\Big(\frac{\Delta t}{m}\nabla \cdot \sigma^{p+1} + \frac{\Delta t}{m}\mathbf{R}^p + \mathbf{u}_n - \mathbf{u}^p\Big) \tag{8.18}$$

$\mathbf{R}$ contains all terms in the momentum equations except for the rheology terms and the time derivative; α and β are free parameters (SEAICE_evpAlpha, SEAICE_evpBeta) that replace the time stepping parameters SEAICE_deltaTevp (Δt_{EVP}), SEAICE_elasticParm (E_0), or SEAICE_evpTauRelax (T). α and β determine the speed of convergence and the stability. Usually, it makes sense to use $\alpha = \beta$, and SEAICEnEVPstarSteps $\gg (\alpha, \beta)$ (Kimmritz et al. 2015 [KDL15]). Currently, there is no termination criterion and the number of mEVP iterations is fixed to SEAICEnEVPstarSteps.

In order to use mEVP in MITgcm, set SEAICEuseEVPstar = `.TRUE.,` in `data.seaice`. If SEAICEuseEVPrev =`.TRUE.,` the actual form of equations (8.17) and (8.18) is used with fewer implicit terms and the factor of e^2 dropped in the stress equations (8.14) and (8.15). Although this modifies the original EVP-equations, it turns out to improve convergence (Bouillon et al. 2013 [BFLM13]).

Another variant is the aEVP scheme (Kimmritz et al. 2016 [KDL16]), where the value of α is set dynamically based on the stability criterion

$$\alpha = \beta = \max\left(\tilde{c}\pi\sqrt{c\frac{\zeta}{A_c}\frac{\Delta t}{\max(m, 10^{-4}\,\mathrm{kg})}}, \alpha_{\min}\right) \tag{8.19}$$

with the grid cell area A_c and the ice and snow mass m. This choice sacrifices speed of convergence for stability with the result that aEVP converges quickly to VP where α can be small and more slowly in areas where the equations are stiff. In practice, aEVP leads to an overall better convergence than mEVP (Kimmritz et al. 2016 [KDL16]). To use aEVP in MITgcm set SEAICEaEVPcoeff $= \tilde{c}$; this also sets the default values of SEAICEaEVPcStar ($c = 4$) and SEAICEaEVPalphaMin ($\alpha_{\min} = 5$). Good convergence has been obtained with these values (Kimmritz et al. 2016 [KDL16]): SEAICEaEVPcoeff $= 0.5$, SEAICEnEVPstarSteps $= 500$, SEAICEuseEVPstar = `.TRUE.`, SEAICEuseEVPrev = `.TRUE.`.

Note, that probably because of the C-grid staggering of velocities and stresses, mEVP may not converge as successfully as in Kimmritz et al. (2015) [KDL15], see also Kimmritz et al. (2016) [KDL16], and that convergence at very high resolution (order 5 km) has not been studied yet.

Truncated ellipse method (TEM) for yield curve

In the so-called truncated ellipse method the shear viscosity η is capped to suppress any tensile stress:

$$\eta = \min\left(\frac{\zeta}{e^2}, \frac{\frac{P}{2} - \zeta(\dot{\epsilon}_{11} + \dot{\epsilon}_{22})}{\sqrt{\max(\Delta_{\min}^2, (\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + 4\dot{\epsilon}_{12}^2)}}\right). \tag{8.20}$$

To enable this method, set `#define SEAICE_ALLOW_TEM` in SEAICE_OPTIONS.h and turn it on with SEAICEuseTEM in `data.seaice`.

Ice-Ocean stress

Moving sea ice exerts a stress on the ocean which is the opposite of the stress τ_{ocean} in (8.2). This stress is applied directly to the surface layer of the ocean model. An alternative ocean stress formulation is given by Hibler and Bryan (1987) [HB87]. Rather than applying τ_{ocean} directly, the stress is derived from integrating over the ice thickness to the bottom of the oceanic surface layer. In the resulting equation for the *combined* ocean-ice momentum, the interfacial stress cancels and the total stress appears as the sum of windstress and divergence of internal ice stresses: $\delta(z)(\tau_{\text{air}} + \mathbf{F})/\rho_0$, see also Eq. (2) of Hibler and Bryan (1987) [HB87]. The disadvantage of this formulation is that now the velocity in the surface layer of the ocean that is used to advect tracers, is really an average over the ocean surface velocity and the ice velocity leading to an inconsistency as the ice temperature and salinity are different from the oceanic variables. To turn on the stress formulation of Hibler and Bryan (1987) [HB87], set useHB87StressCoupling = `.TRUE.`, in `data.seaice`.

Finite-volume discretization of the stress tensor divergence

On an Arakawa C grid, ice thickness and concentration and thus ice strength P and bulk and shear viscosities ζ and η are naturally defined a C-points in the center of the grid cell. Discretization requires only averaging of ζ and η to vorticity or Z-points (or ζ-points, but here we use Z in order avoid confusion with the bulk viscosity) at the bottom left corner of the cell to give $\overline{\zeta}^Z$ and $\overline{\eta}^Z$. In the following, the superscripts indicate location at Z or C points, distance across the cell (F), along the cell edge (G), between u-points (U), v-points (V), and C-points (C). The control volumes of the u- and v-equations in the grid cell at indices (i, j) are $A^w_{i,j}$ and $A^s_{i,j}$, respectively. With these definitions (which follow

the model code documentation except that ζ-points have been renamed to Z-points), the strain rates are discretized as:

$$\begin{aligned}
\dot{\epsilon}_{11} &= \partial_1 u_1 + k_2 u_2 \\
\Rightarrow (\epsilon_{11})^C_{i,j} &= \frac{u_{i+1,j} - u_{i,j}}{\Delta x^F_{i,j}} + k^C_{2,i,j} \frac{v_{i,j+1} + v_{i,j}}{2} \\
\dot{\epsilon}_{22} &= \partial_2 u_2 + k_1 u_1 \\
\Rightarrow (\epsilon_{22})^C_{i,j} &= \frac{v_{i,j+1} - v_{i,j}}{\Delta y^F_{i,j}} + k^C_{1,i,j} \frac{u_{i+1,j} + u_{i,j}}{2} \\
\dot{\epsilon}_{12} = \dot{\epsilon}_{21} &= \frac{1}{2}\Big(\partial_1 u_2 + \partial_2 u_1 - k_1 u_2 - k_2 u_1 \Big) \\
\Rightarrow (\epsilon_{12})^Z_{i,j} &= \frac{1}{2}\Bigg(\frac{v_{i,j} - v_{i-1,j}}{\Delta x^V_{i,j}} + \frac{u_{i,j} - u_{i,j-1}}{\Delta y^U_{i,j}} \\
&\qquad - k^Z_{1,i,j} \frac{v_{i,j} + v_{i-1,j}}{2} - k^Z_{2,i,j} \frac{u_{i,j} + u_{i,j-1}}{2} \Bigg),
\end{aligned}$$

so that the diagonal terms of the strain rate tensor are naturally defined at C-points and the symmetric off-diagonal term at Z-points. No-slip boundary conditions ($u_{i,j-1} + u_{i,j} = 0$ and $v_{i-1,j} + v_{i,j} = 0$ across boundaries) are implemented via "ghost-points"; for free slip boundary conditions $(\epsilon_{12})^Z = 0$ on boundaries.

For a spherical polar grid, the coefficients of the metric terms are $k_1 = 0$ and $k_2 = -\tan\phi/a$, with the spherical radius a and the latitude ϕ; $\Delta x_1 = \Delta x = a\cos\phi\Delta\lambda$, and $\Delta x_2 = \Delta y = a\Delta\phi$. For a general orthogonal curvilinear grid, k_1 and k_2 can be approximated by finite differences of the cell widths:

$$\begin{aligned}
k^C_{1,i,j} &= \frac{1}{\Delta y^F_{i,j}} \frac{\Delta y^G_{i+1,j} - \Delta y^G_{i,j}}{\Delta x^F_{i,j}} \\
k^C_{2,i,j} &= \frac{1}{\Delta x^F_{i,j}} \frac{\Delta x^G_{i,j+1} - \Delta x^G_{i,j}}{\Delta y^F_{i,j}} \\
k^Z_{1,i,j} &= \frac{1}{\Delta y^U_{i,j}} \frac{\Delta y^C_{i,j} - \Delta y^C_{i-1,j}}{\Delta x^V_{i,j}} \\
k^Z_{2,i,j} &= \frac{1}{\Delta x^V_{i,j}} \frac{\Delta x^C_{i,j} - \Delta x^C_{i,j-1}}{\Delta y^U_{i,j}}
\end{aligned}$$

The stress tensor is given by the constitutive viscous-plastic relation $\sigma_{\alpha\beta} = 2\eta\dot{\epsilon}_{\alpha\beta} + [(\zeta - \eta)\dot{\epsilon}_{\gamma\gamma} - P/2]\delta_{\alpha\beta}$. The stress tensor divergence $(\nabla\sigma)_\alpha = \partial_\beta \sigma_{\beta\alpha}$, is discretized in finite volumes . This conveniently avoids dealing with further metric terms, as these are "hidden" in the differential cell widths. For the u-equation ($\alpha = 1$) we have:

$$\begin{aligned}
(\nabla\sigma)_1 : \quad & \frac{1}{A^w_{i,j}} \int_{\text{cell}} (\partial_1\sigma_{11} + \partial_2\sigma_{21})\, dx_1\, dx_2 \\
=& \frac{1}{A^w_{i,j}} \left\{ \int_{x_2}^{x_2+\Delta x_2} \sigma_{11} dx_2 \Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1} \sigma_{21} dx_1 \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
\approx& \frac{1}{A^w_{i,j}} \left\{ \Delta x_2 \sigma_{11} \Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1 \sigma_{21} \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
=& \frac{1}{A^w_{i,j}} \Big\{ (\Delta x_2\sigma_{11})^C_{i,j} - (\Delta x_2\sigma_{11})^C_{i-1,j} \\
& \qquad + (\Delta x_1\sigma_{21})^Z_{i,j+1} - (\Delta x_1\sigma_{21})^Z_{i,j} \Big\}
\end{aligned}$$

with

$$
\begin{aligned}
(\Delta x_2 \sigma_{11})^C_{i,j} = \;& \Delta y^F_{i,j} (\zeta+\eta)^C_{i,j} \frac{u_{i+1,j}-u_{i,j}}{\Delta x^F_{i,j}} \\
& + \Delta y^F_{i,j} (\zeta+\eta)^C_{i,j} k^C_{2,i,j} \frac{v_{i,j+1}+v_{i,j}}{2} \\
& + \Delta y^F_{i,j} (\zeta-\eta)^C_{i,j} \frac{v_{i,j+1}-v_{i,j}}{\Delta y^F_{i,j}} \\
& + \Delta y^F_{i,j} (\zeta-\eta)^C_{i,j} k^C_{1,i,j} \frac{u_{i+1,j}+u_{i,j}}{2} \\
& - \Delta y^F_{i,j} \frac{P}{2} \\
(\Delta x_1 \sigma_{21})^Z_{i,j} = \;& \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} \frac{u_{i,j}-u_{i,j-1}}{\Delta y^U_{i,j}} \\
& + \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} \frac{v_{i,j}-v_{i-1,j}}{\Delta x^V_{i,j}} \\
& - \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} k^Z_{2,i,j} \frac{u_{i,j}+u_{i,j-1}}{2} \\
& - \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} k^Z_{1,i,j} \frac{v_{i,j}+v_{i-1,j}}{2}
\end{aligned}
$$

Similarly, we have for the v-equation ($\alpha = 2$):

$$
\begin{aligned}
(\nabla\sigma)_2 : \quad & \frac{1}{A^s_{i,j}} \int_{\text{cell}} (\partial_1 \sigma_{12} + \partial_2 \sigma_{22})\, dx_1\, dx_2 \\
=& \frac{1}{A^s_{i,j}} \left\{ \int_{x_2}^{x_2+\Delta x_2} \sigma_{12} dx_2 \Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1} \sigma_{22} dx_1 \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
\approx& \frac{1}{A^s_{i,j}} \left\{ \Delta x_2 \sigma_{12} \Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1 \sigma_{22} \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
=& \frac{1}{A^s_{i,j}} \Big\{ (\Delta x_2 \sigma_{12})^Z_{i+1,j} - (\Delta x_2 \sigma_{12})^Z_{i,j} \\
& \qquad + (\Delta x_1 \sigma_{22})^C_{i,j} - (\Delta x_1 \sigma_{22})^C_{i,j-1} \Big\}
\end{aligned}
$$

with

$$
\begin{aligned}
(\Delta x_1 \sigma_{12})^Z_{i,j} = \;& \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} \frac{u_{i,j}-u_{i,j-1}}{\Delta y^U_{i,j}} \\
& + \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} \frac{v_{i,j}-v_{i-1,j}}{\Delta x^V_{i,j}} \\
& - \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} k^Z_{2,i,j} \frac{u_{i,j}+u_{i,j-1}}{2} \\
& - \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} k^Z_{1,i,j} \frac{v_{i,j}+v_{i-1,j}}{2} \\
(\Delta x_2 \sigma_{22})^C_{i,j} = \;& \Delta x^F_{i,j} (\zeta-\eta)^C_{i,j} \frac{u_{i+1,j}-u_{i,j}}{\Delta x^F_{i,j}} \\
& + \Delta x^F_{i,j} (\zeta-\eta)^C_{i,j} k^C_{2,i,j} \frac{v_{i,j+1}+v_{i,j}}{2} \\
& + \Delta x^F_{i,j} (\zeta+\eta)^C_{i,j} \frac{v_{i,j+1}-v_{i,j}}{\Delta y^F_{i,j}} \\
& + \Delta x^F_{i,j} (\zeta+\eta)^C_{i,j} k^C_{1,i,j} \frac{u_{i+1,j}+u_{i,j}}{2} \\
& - \Delta x^F_{i,j} \frac{P}{2}
\end{aligned}
$$

Again, no-slip boundary conditions are realized via ghost points and $u_{i,j-1} + u_{i,j} = 0$ and $v_{i-1,j} + v_{i,j} = 0$ across boundaries. For free-slip boundary conditions the lateral stress is set to zeros. In analogy to $(\epsilon_{12})^Z = 0$ on boundaries, we set $\sigma^Z_{21} = 0$, or equivalently $\eta^Z_{i,j} = 0$, on boundaries.

Thermodynamics

NOTE: THIS SECTION IS STILL NOT COMPLETE

In its original formulation the sea ice model uses simple thermodynamics following the appendix of Semtner (1976) [Sem76]. This formulation does not allow storage of heat, that is, the heat capacity of ice is zero. Upward conductive heat flux is parameterized assuming a linear temperature profile and together with a constant ice conductivity. It is expressed as $(K/h)(T_w - T_0)$, where K is the ice conductivity, h the ice thickness, and $T_w - T_0$ the difference between water and ice surface temperatures. This type of model is often refered to as a "zero-layer" model. The surface heat flux is computed in a similar way to that of Parkinson and Washington (1979) [PW79] and Manabe et al. (1979) [MBS79]. All fluxes are assumed to instanteously reach a stationary balance ($\partial T_0/\partial t = 0$):

$$\begin{aligned}\rho\, c_p \frac{\partial T_0}{\partial t} = 0 = &\frac{K}{h}(T_w - T_0) + Q_{SW\downarrow}(1 - \text{albedo}) + \epsilon Q_{LW\downarrow} - Q_{LW\uparrow}(T_0) \\ &+ Q_{LH}(T_0) + Q_{SH}(T_0)\end{aligned} \tag{8.21}$$

where ϵ is the emissivity of the surface (snow or ice), $Q_{S/LW\downarrow}$ the downwelling shortwave and longwave radiation to be prescribed, and $Q_{LW\uparrow} = \epsilon\sigma_B T_0^4$ the emitted long wave radiation with the Stefan-Boltzmann constant σ_B. With explicit expressions in T_0 for the turbulent fluxes of latent heat:

$$Q_{LH} = \rho_{\text{air}} C_E (\Lambda_v + \Lambda_f) |\mathbf{U}_{\text{air}}| \, [q_{\text{air}} - q_{\text{sat}}(T_0)]$$

where ρ_{air} is the air density (parameter SEAICE_rhoAir), C_E is the ice-ocean transfer coefficient for sensible and latent heat (parameter SEAICE_dalton), Λ_v and Λ_f are the latent heat of vaporization and fusion, respectively (parameters SEAICE_lhEvap and SEAICE_lhFusion). Similarly, for sensible heat:

$$Q_{SH} = \rho_{\text{air}} c_p C_E |\mathbf{U}_{\text{air}}| \, [T_{10\text{m}} - T_0]$$

where c_p is the specific heat of air (parameter SEAICE_cpAir). (8.21) can be solved for T_0 with an iterative Ralphson-Newton method which usually converges very quickly, in less that 10 iterations. For the latent heat Q_{LH} a choice can be made between the old polynomial expression for saturation humidity $q_{\text{sat}}(T_0)$ (by setting useMaykutSatVapPoly to `.TRUE.`) and the default exponential relation approximation that is more accurate at low temperatures.

In the zero-layer model of Semtner (1976) [Sem76], the conductive heat flux depends strongly on the ice thickness h. However, the ice thickness in the model represents a mean over a potentially very heterogeneous thickness distribution. In order to parameterize a sub-grid scale distribution for heat flux computations, the mean ice thickness h is split into N thickness categories H_n that are equally distributed between $2h$ and a minimum imposed ice thickness of 5 cm by $H_n = \frac{2n-1}{7} h$ for $n \in [1, N]$. The heat fluxes computed for each thickness category are area-averaged to give the total heat flux (see Hibler 1984 [Hib84]). To use this thickness category parameterization set SEAICE_multDim to the number of desired categories in `data.seaice` (7 is a good guess, for anything larger than 7 modify SEAICE_SIZE.h). Note that this requires different restart files and switching this flag on in the middle of an integration is not advised. As an alternative to the flat distribution, the run-time parameter SEAICE_PDF (1D-array of lenght nITD) can be used to prescribe an arbitrary distribution of ice thicknesses, for example derived from observed distributions (Castro-Morales et al. 2014 [CMKL+14]). In order to include the ice thickness distribution also for snow, set SEAICE_useMultDimSnow = `.TRUE.` (this is the default); only then, the parameterization of always having a fraction of thin ice is efficient and generally thicker ice is produced (see Castro-Morales et al. 2014 [CMKL+14]).

The atmospheric heat flux is balanced by an oceanic heat flux from below. The oceanic flux is proportional to $\rho\, c_p\, (T_w - T_{fr})$ where ρ and c_p are the density and heat capacity of sea water and T_{fr} is the local freezing point temperature that is a function of salinity. This flux is not assumed to instantaneously melt or create ice, but a time

scale of three days (run-time parameter SEAICE_gamma_t) is used to relax T_w to the freezing point. The parameterization of lateral and vertical growth of sea ice follows that of Hibler (1979) and Hibler (1980) [Hib79][Hib80]; the so-called lead closing parameter h_0 (run-time parameter HO) has a default value of 0.5 meters.

On top of the ice there is a layer of snow that modifies the heat flux and the albedo (Zhang et al. 1998 [ZWDHSR98]). Snow modifies the effective conductivity according to

$$\frac{K}{h} \rightarrow \frac{1}{\frac{h_s}{K_s} + \frac{h}{K}},$$

where K_s is the conductivity of snow and h_s the snow thickness. If enough snow accumulates so that its weight submerges the ice and the snow is flooded, a simple mass conserving parameterization of snowice formation (a flood-freeze algorithm following Archimedes' principle) turns snow into ice until the ice surface is back at $z = 0$ (see Leppäranta 1983 [Lepparanta83]). The flood-freeze algorithm is turned on with run-time parameter SEAICEuseFlooding=.TRUE..

Advection of thermodynamic variables

Effective ice thickness (ice volume per unit area, $c \cdot h$), concentration c and effective snow thickness ($c \cdot h_s$) are advected by ice velocities:

$$\frac{\partial X}{\partial t} = -\nabla \cdot (\mathbf{u}\, X) + \Gamma_X + D_X \tag{8.22}$$

where Γ_X are the thermodynamic source terms and D_X the diffusive terms for quantities $X = (c \cdot h), c, (c \cdot h_s)$. From the various advection schemes that are available in MITgcm, we recommend flux-limited schemes to preserve sharp gradients and edges that are typical of sea ice distributions and to rule out unphysical over- and undershoots (negative thickness or concentration). These schemes conserve volume and horizontal area and are unconditionally stable, so that we can set $D_X = 0$. Run-time flags: SEAICEadvScheme (default=77, is a 2nd-order flux limited scheme), DIFF1 = $D_X/\Delta x$ (default=0).

The MITgcm sea ice model provides the option to use the thermodynamics model of Winton (2000) [Win00], which in turn is based on the 3-layer model of Semtner (1976) [Sem76] which treats brine content by means of enthalpy conservation; the corresponding package thsice is described in section Section 8.6.1. This scheme requires additional state variables, namely the enthalpy of the two ice layers (instead of effective ice salinity), to be advected by ice velocities. The internal sea ice temperature is inferred from ice enthalpy. To avoid unphysical (negative) values for ice thickness and concentration, a positive 2nd-order advection scheme with a SuperBee flux limiter (Roe 1985 [Roe85]) should be used to advect all sea-ice-related quantities of the Winton (2000) [Win00] thermodynamic model (run-time flag thSIceAdvScheme = 77 and thSIce_diffK = $D_X = 0$ in `data.ice`, defaults are 0). Because of the nonlinearity of the advection scheme, care must be taken in advecting these quantities: when simply using ice velocity to advect enthalpy, the total energy (i.e., the volume integral of enthalpy) is not conserved. Alternatively, one can advect the energy content (i.e., product of ice-volume and enthalpy) but then false enthalpy extrema can occur, which then leads to unrealistic ice temperature. In the currently implemented solution, the sea-ice mass flux is used to advect the enthalpy in order to ensure conservation of enthalpy and to prevent false enthalpy extrema.

Dynamical Ice Thickness Distribution (ITD)

The ice thickness distribution model used by MITgcm follows the implementation in the Los Alamos sea ice model CICE (https://github.com/CICE-Consortium/CICE). There are two parts to it that are closely connected: the participation and ridging functions that determine which thickness classes take part in ridging and which thickness classes receive ice during ridging based on Thorndike et al. (1975) [TRMC75], and the ice strength parameterization by Rothrock (1975) [Rot75] which uses this information. The following description is slightly modified from Ungermann et al. (2017) [UTML17]. Verification experiment seaice_itd uses the ITD model.

Distribution, participation and redistribution functions in ridging

When SEAICE_ITD is defined in SEAICE_OPTIONS.h, the ice thickness is described by the ice thickness distribution $g(h, \mathbf{x}, t)$ for the subgrid-scale (see Thorndike et al. 1975 [TRMC75]), a probability density function for thickness h following the evolution equation

$$\frac{\partial g}{\partial t} = -\nabla \cdot (\mathbf{u} g) - \frac{\partial}{\partial h}(f g) + \Psi. \tag{8.23}$$

Here $f = \frac{\mathrm{d}h}{\mathrm{d}t}$ is the thermodynamic growth rate and Ψ a function describing the mechanical redistribution of sea ice during ridging or lead opening.

The mechanical redistribution function Ψ generates open water in divergent motion and creates ridged ice during convergent motion. The ridging process depends on total strain rate and on the ratio between shear (run-time parameter SEAICEshearParm) and divergent strain. In the single category model, ridge formation is treated implicitly by limiting the ice concentration to a maximum of one (see Hibler 1979 [Hib79]), so that further volume increase in convergent motion leads to thicker ice. (This is also the default for ITD models; to change from the default, set run-time parameter SEAICEsimpleRidging =`.FALSE.` in `data.seaice`). For the ITD model, the ridging mode in convergence

$$\omega_r(h) = \frac{-a(h) + n(h)}{N}$$

gives the effective change for the ice volume with thickness between h and $h+\mathrm{d}h$ as the normalized difference between the ice $n(h)$ generated by ridging and the ice $a(h)$ participating in ridging.

The participation function $a(h) = b(h)g(h)$ can be computed either following Thorndike et al. (1975) [TRMC75] (run-time parameter SEAICEpartFunc =0) or Lipscomb et al. (2007) [LHMJ07] (SEAICEpartFunc =1), and similarly the ridging function $n(h)$ can be computed following Hilber (1980) [Hib80] (run-time parameter SEAICEredistFunc =0) or Lipscomb et al. (2007) [LHMJ07] (SEAICEredistFunc =1). As an example, we show here the functions that Lipscomb et al. (2007) [LHMJ07] suggested to avoid noise in the solutions. These functions are smooth and avoid non-differentiable discontinuities, but so far we did not find any noise issues as in Lipscomb et al. (2007) [LHMJ07].

With SEAICEpartFunc =1 in `data.seaice`, the participation function with the relative amount of ice of thickness h weighted by an exponential function

$$b(h) = b_0 \exp[-G(h)/a^*]$$

where $G(h) = \int_0^h g(h)\mathrm{d}h$ is the cumulative thickness distribution function, b_0 a normalization factor, and a^* (SEAICEaStar) the exponential constant that determines which relative amount of thicker and thinner ice take part in ridging.

With SEAICEredistFunc =1 in `data.seaice`, the ice generated by ridging is calculated as

$$n(h) = \int_0^\infty a(h_1)\gamma(h_1, h)\mathrm{d}h_1$$

where the density function $\gamma(h_1, h)$ of resulting thickness h for ridged ice with an original thickness of h_1 is taken as

$$\gamma(h_1, h) = \frac{1}{k\lambda} \exp\left[\frac{-(h - h_{\min})}{\lambda}\right]$$

for $h \geq h_{\min}$, with $\gamma(h_1, h) = 0$ for $h < h_{\min}$. In this parameterization, the normalization factor $k = \frac{h_{\min}+\lambda}{h_1}$, the e-folding scale $\lambda = \mu h_1^{1/2}$ and the minimum ridge thickness $h_{\min} = \min(2h_1, h_1 + h_{\text{raft}})$ all depend on the original thickness h_1. The maximal ice thickness allowed to raft h_{raft} is constant (SEAICEmaxRaft, default =1 m) and μ (SEAICEmuRidging) is a tunable parameter.

In the numerical model these equations are discretized into a set of n (nITD defined in SEAICE_SIZE.h) thickness categories employing the delta function scheme of Bitz et al. (2001) [BHWE01]. For each thickness category in an

ITD configuration, the volume conservation equation (8.22) is evaluated using the heat flux with the category-specific values for ice and snow thickness, so there are no conceptual differences in the thermodynamics between the single category and ITD configurations. The only difference is that only in the thinnest category the creation of new ice of thickness H_0 (run-time parameter HO) is possible, all other categories are limited to basal growth. The conservation of ice area is replaced by the evolution equation of the ITD (8.23) that is discretized in thickness space with $n+1$ category limits given by run-time parameter Hlimit. If Hlimit is not set in `data.seaice`, a simple recursive formula following Lipscomb (2001) [Lip01] is used to compute Hlimit:

$$H_{\text{limit}}(k) = H_{\text{limit}}(k-1) + \frac{c_1}{n} + \frac{c_1 c_2}{n}[1 + \tanh c_3(\frac{k-1}{n} - 1)]$$

with $H_{\text{limit}}(0) = 0\,\text{m}$ and $H_{\text{limit}}(n) = 999.9\,\text{m}$. The three constants are the run-time parameters Hlimit_c1, Hlimit_c2, and Hlimit_c3. The total ice concentration and volume can then be calculated by summing up the values for each category.

Ice strength parameterization

In the default approach of equation (8.4), the ice strength is parameterized following Hibler (1979) [Hib79] and P depends only on average ice concentration and thickness per grid cell and the constant ice strength parameters P^* (SEAICE_strength) and C^* (SEAICE_cStar). With an ice thickness distribution, it is possible to use a different parameterization following Rothrock (1975) [Rot75]

$$P = C_f C_p \int_0^\infty h^2 \omega_r(h) \mathrm{d}h \tag{8.24}$$

by considering the production of potential energy and the frictional energy loss in ridging. The physical constant $C_p = \rho_i(\rho_w - \rho_i)\hat{g}/(2\rho_w)$ is a combination of the gravitational acceleration $\hat{g}$ and the densities ρ_i, ρ_w of ice and water, and C_f (SEAICE_cf) is a scaling factor relating the amount of work against gravity necessary for ridging to the amount of work against friction. To calculate the integral, this parameterization needs information about the ITD in each grid cell, while the default parameterization (8.4) can be used for both ITD and single thickness category models. In contrast to (8.4), which is based on the plausible assumption that thick and compact ice is stronger than thin and loose drifting ice, this parameterization (8.24) clearly contains the more physical assumptions about energy conservation. For that reason alone this parameterization is often considered to be more physically realistic than (8.4), but in practice, the success is not so clear (Ungermann et al. 2007 [UTML17]). Ergo, the default is to use (8.4); set useHibler79IceStrength =`.FALSE.` in `data.seaice` to change this behavior.

8.6.2.5 Key subroutines

Top-level routine: pkg/seaice/seaice_model.F

```
C     !CALLING SEQUENCE:
c ...
c  seaice_model (TOP LEVEL ROUTINE)
c  |
c  |-- #ifdef SEAICE_CGRID
c  |     SEAICE_DYNSOLVER
c  |     |
c  |     |-- < compute proxy for geostrophic velocity >
c  |     |
c  |     |-- < set up mass per unit area and Coriolis terms >
c  |     |
c  |     |-- < dynamic masking of areas with no ice >
c  |     |
c  |     |
```

(continues on next page)

(continued from previous page)

```
c  |   #ELSE
c  |     DYNSOLVER
c  |   #ENDIF
c  |
c  |-- if ( useOBCS )
c  |     OBCS_APPLY_UVICE
c  |
c  |-- if ( SEAICEadvHeff .OR. SEAICEadvArea .OR. SEAICEadvSnow .OR. SEAICEadvSalt )
c  |     SEAICE_ADVDIFF
c  |
c  |   SEAICE_REG_RIDGE
c  |
c  |-- if ( usePW79thermodynamics )
c  |     SEAICE_GROWTH
c  |
c  |-- if ( useOBCS )
c  |     if ( SEAICEadvHeff ) OBCS_APPLY_HEFF
c  |     if ( SEAICEadvArea ) OBCS_APPLY_AREA
c  |     if ( SEAICEadvSALT ) OBCS_APPLY_HSALT
c  |     if ( SEAICEadvSNOW ) OBCS_APPLY_HSNOW
c  |
c  |-- < do various exchanges >
c  |
c  |-- < do additional diagnostics >
c  |
c  o
```

8.6.2.6 SEAICE diagnostics

Diagnostics output is available via the diagnostics package (see Section 9.1). Available output fields are summarized in the following table:

```
---------+----------+----------------+-----------------
 <-Name->|<- grid ->|<--  Units   -->|<- Tile (max=80c)
---------+----------+----------------+-----------------
 sIceLoad|SM      U1|kg/m^2          |sea-ice loading (in Mass of ice+snow / area␣
↪unit)
---
SEA ICE STATE:
---
 SIarea  |SM      M1|m^2/m^2         |SEAICE fractional ice-covered area [0 to 1]
 SIheff  |SM      M1|m               |SEAICE effective ice thickness
 SIhsnow |SM      M1|m               |SEAICE effective snow thickness
 SIhsalt |SM      M1|g/m^2           |SEAICE effective salinity
 SIuice  |UU      M1|m/s             |SEAICE zonal ice velocity, >0 from West to East
 SIvice  |VV      M1|m/s             |SEAICE merid. ice velocity, >0 from South to␣
↪North
---
ATMOSPHERIC STATE AS SEEN BY SEA ICE:
---
 SItices |SM  C   M1|K               |Surface Temperature over Sea-Ice (area weighted)
 SIuwind |UM      U1|m/s             |SEAICE zonal 10-m wind speed, >0 increases uVel
 SIvwind |VM      U1|m/s             |SEAICE meridional 10-m wind speed, >0 increases␣
↪uVel
 SIsnPrcp|SM      U1|kg/m^2/s        |Snow precip. (+=dw) over Sea-Ice (area weighted)
```

(continues on next page)

(continued from previous page)

```
---
FLUXES ACROSS ICE-OCEAN INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):
---
 SIfu    |UU      U1|N/m^2           |SEAICE zonal surface wind stress, >0 increases␣
↪uVel
 SIfv    |VV      U1|N/m^2           |SEAICE merid. surface wind stress, >0 increases␣
↪vVel
 SIqnet  |SM      U1|W/m^2           |Ocean surface heatflux, turb+rad, >0 decreases␣
↪theta
 SIqsw   |SM      U1|W/m^2           |Ocean surface shortwave radiat., >0 decreases␣
↪theta
 SIempmr |SM      U1|kg/m^2/s        |Ocean surface freshwater flux, > 0 increases␣
↪salt
 SIqneto |SM      U1|W/m^2           |Open Ocean Part of SIqnet, turb+rad, >0 decr␣
↪theta
 SIqneti |SM      U1|W/m^2           |Ice Covered Part of SIqnet, turb+rad, >0 decr␣
↪theta
---
FLUXES ACROSS ATMOSPHERE-ICE INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):
---
 SIatmQnt|SM      U1|W/m^2           |Net atmospheric heat flux, >0 decreases theta
 SIatmFW |SM      U1|kg/m^2/s        |Net freshwater flux from atmosphere & land␣
↪(+=down)
 SIfwSubl|SM      U1|kg/m^2/s        |Freshwater flux of sublimated ice, >0 decreases␣
↪ice
---
THERMODYNAMIC DIAGNOSTICS:
---
 SIareaPR|SM      M1|m^2/m^2         |SIarea preceeding ridging process
 SIareaPT|SM      M1|m^2/m^2         |SIarea preceeding thermodynamic growth/melt
 SIheffPT|SM      M1|m               |SIheff preceeeding thermodynamic growth/melt
 SIhsnoPT|SM      M1|m               |SIhsnow preceeeding thermodynamic growth/melt
 SIaQbOCN|SM      M1|m/s             |Potential HEFF rate of change by ocean ice flux
 SIaQbATC|SM      M1|m/s             |Potential HEFF rate of change by atm flux over␣
↪ice
 SIaQbATO|SM      M1|m/s             |Potential HEFF rate of change by open ocn atm␣
↪flux
 SIdHbOCN|SM      M1|m/s             |HEFF rate of change by ocean ice flux
 SIdSbATC|SM      M1|m/s             |HSNOW rate of change by atm flux over sea ice
 SIdSbOCN|SM      M1|m/s             |HSNOW rate of change by ocean ice flux
 SIdHbATC|SM      M1|m/s             |HEFF rate of change by atm flux over sea ice
 SIdHbATO|SM      M1|m/s             |HEFF rate of change by open ocn atm flux
 SIdHbFLO|SM      M1|m/s             |HEFF rate of change by flooding snow
 SIdAbATO|SM      M1|m^2/m^2/s       |Potential AREA rate of change by open ocn atm␣
↪flux
 SIdAbATC|SM      M1|m^2/m^2/s       |Potential AREA rate of change by atm flux over␣
↪ice
 SIdAbOCN|SM      M1|m^2/m^2/s       |Potential AREA rate of change by ocean ice flux
 SIdA    |SM      M1|m^2/m^2/s       |AREA rate of change (net)
---
DYNAMIC/RHEOLOGY DIAGNOSTICS:
---
 SIpress |SM      M1|N/m             |SEAICE strength (with upper and lower limit)
 SIzeta  |SM      M1|kg/s            |SEAICE nonlinear bulk viscosity
 SIeta   |SM      M1|kg/s            |SEAICE nonlinear shear viscosity
 SIsig1  |SM      M1|no units        |SEAICE normalized principle stress, component␣
↪one
```

(continues on next page)

(continued from previous page)

```
 SIsig2  |SM      M1|no units        |SEAICE normalized principle stress, component␣
↪two
 SIshear |SM      M1|1/s            |SEAICE shear deformation rate
 SIdelta |SM      M1|1/s            |SEAICE Delta deformation rate
 SItensil|SM      M1|N/m            |SEAICE maximal tensile strength
---
ADVECTIVE/DIFFUSIVE FLUXES OF SEA ICE variables:
---
 ADVxHEFF|UU      M1|m.m^2/s        |Zonal      Advective Flux of eff ice thickn
 ADVyHEFF|VV      M1|m.m^2/s        |Meridional Advective Flux of eff ice thickn
 SIuheff |UU      M1|m^2/s          |Zonal      Transport of eff ice thickn␣
↪(centered)
 SIvheff |VV      M1|m^2/s          |Meridional Transport of eff ice thickn␣
↪(centered)
 DFxEHEFF|UU      M1|m^2/s          |Zonal      Diffusive Flux of eff ice thickn
 DFyEHEFF|VV      M1|m^2/s          |Meridional Diffusive Flux of eff ice thickn
 ADVxAREA|UU      M1|m^2/m^2.m^2/s  |Zonal      Advective Flux of fract area
 ADVyAREA|VV      M1|m^2/m^2.m^2/s  |Meridional Advective Flux of fract area
 DFxEAREA|UU      M1|m^2/m^2.m^2/s  |Zonal      Diffusive Flux of fract area
 DFyEAREA|VV      M1|m^2/m^2.m^2/s  |Meridional Diffusive Flux of fract area
 ADVxSNOW|UU      M1|m.m^2/s        |Zonal      Advective Flux of eff snow thickn
 ADVySNOW|VV      M1|m.m^2/s        |Meridional Advective Flux of eff snow thickn
 DFxESNOW|UU      M1|m.m^2/s        |Zonal      Diffusive Flux of eff snow thickn
 DFyESNOW|VV      M1|m.m^2/s        |Meridional Diffusive Flux of eff snow thickn
 ADVxSSLT|UU      M1|psu.m^2/s      |Zonal      Advective Flux of seaice salinity
 ADVySSLT|VV      M1|psu.m^2/s      |Meridional Advective Flux of seaice salinity
 DFxESSLT|UU      M1|psu.m^2/s      |Zonal      Diffusive Flux of seaice salinity
 DFyESSLT|VV      M1|psu.m^2/s      |Meridional Diffusive Flux of seaice salinity
```

8.6.2.7 Experiments and tutorials that use seaice

- verification/lab_sea: Labrador Sea experiment
- verification/seaice_obcs, based on lab_sea
- verification/offline_exf_seaice, idealized topography in a zonally re-entrant channel
- verification/seaice_itd, based on offline_exf_seaice, tests ice thickness distribution
- verification/global_ocean.cs32x15, global cubed-sphere-experiment with combinations of pkg/seaice and pkg/thsice
- verification/1D_ocean_ice_column, just thermodynamics

8.6.3 SHELFICE Package

Authors: Martin Losch, Jean-Michel Campin

8.6.3.1 Introduction

pkg/shelfice provides a thermodynamic model for basal melting underneath floating ice shelves.

CPP options enable or disable different aspects of the package (Section 8.6.3.2). Run-Time options, flags, filenames and field-related dates/times are described in Section 8.6.3.3. A description of key subroutines is given in Section 8.6.3.5. Available diagnostics output is listed in Section Section 8.6.3.6.

8.6.3.2 SHELFICE configuration

As with all MITgcm packages, pkg/shelfice can be turned on or off at compile time:

- using the `packages.conf` file by adding `shelfice` to it,
- or using genmake2 adding `-enable=shelfice` or `disable=shelfice` switches

pkg/shelfice does not require any additional packages, but it will only work with conventional vertical z-coordinates (pressure coordinates are not implemented). If you use it together with vertical mixing schemes, be aware that non-local parameterizations are turned off, e.g., such as pkg/kpp.

Parts of the pkg/shelfice code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in SHELFICE_OPTIONS.h:

CPP Flag Name	Default	Description
ALLOW_SHELFICE_DEBUG	#undef	include code for enhanced diagnostics and debug output
ALLOW_ISOMIP_TD	#define	include code for for simplified ISOMIP thermodynamics
SHI_ALLOW_GAMMAFRICT	#define	allow friction velocity-dependent transfer coefficient following Holland and Jenkins (1999) [HJ99]

8.6.3.3 SHELFICE Run-time Parameters

pkg/shelfice is switched on/off at run-time by setting useSHELFICE to `.TRUE.` in file `data.pkg`. Run-time parameters are set in file `data.shelfice` (read in pkg/shelfice/shelfice_readparms.F),as listed below.

The data file specifying under-ice topography of ice shelves (SHELFICEtopoFile) is in meters; upwards is positive, and as for the bathymetry files, negative values are required for topography below the sea-level. The data file for the pressure load anomaly at the bottom of the ice shelves SHELFICEloadAnomalyFile is in pressure units (Pa). This field is absolutely required to avoid large excursions of the free surface during initial adjustment processes, obtained by integrating an approximate density from the surface at $z = 0$ down to the bottom of the last fully dry cell within the ice shelf, see (8.28). Note however the file SHELFICEloadAnomalyFile must not be p_{top}, but $p_{top} - g \sum_{k'=1}^{n-1} \rho_0 \Delta z_{k'}$, with $\rho_0 =$ rhoConst, so that in the absence of a ρ^* that is different from ρ_0, the anomaly is zero.

Parameter	Group	Default	Description
useISOMIPTD	SHELFICE_PARM01	FALSE	use simplified ISOMIP thermodynamics on/off flag
SHELFICEconserve	SHELFICE_PARM01	FALSE	use conservative form of temperature boundary conditions on/off flag
SHELFICEboundaryLayer	SHELFICE_PARM01	FALSE	use simple boundary layer mixing parameterization on/off flag
SHELFICEloadAnomalyFile	SHELFICE_PARM01	' '	initial geopotential anomaly
SHELFICEtopoFile	SHELFICE_PARM01	' '	filename for under-ice topography of ice shelves
SHELFICElatentHeat	SHELFICE_PARM01	334.0E+03	latent heat of fusion (J/kg)
SHELFICEHeatCapacity_Cp	SHELFICE_PARM01	2000.0E+00	specific heat capacity of ice (J/kg/K)
rhoShelfIce	SHELFICE_PARM01	917.0E+00	(constant) mean density of ice shelf (kg/m^3)
SHELFICEheatTransCoeff	SHELFICE_PARM01	1.0E-04	transfer coefficient (exchange velocity) for temperature (m/s)
SHELFICEsaltTransCoeff	SHELFICE_PARM01	5.05E-03 × SHELFICEheatTransCoeff	transfer coefficient (exchange velocity) for salinity (m/s)
SHELFICEkappa	SHELFICE_PARM01	1.54E-06	temperature diffusion coefficient of the ice shelf (m^2/s)
SHELFICEthetaSurface	SHELFICE_PARM01	-20.0E+00	(constant) surface temperature above the ice shelf (^{o}C)
no_slip_shelfice	SHELFICE_PARM01	no_slip_bottom	slip along bottom of ice shelf on/off flag
SHELFICEDragLinear	SHELFICE_PARM01	bottomDragLinear	linear drag coefficient at bottom ice shelf (m/s)
SHELFICEDragQuadratic	SHELFICE_PARM01	bottomDragQuadratic	quadratic drag coefficient at bottom ice shelf (non-dim.)
SHELFICEwriteState	SHELFICE_PARM01	FALSE	write ice shelf state to file on/off flag
SHELFICE_dumpFreq	SHELFICE_PARM01	dumpFreq	dump frequency (s)
SHELFICE_dump_mnc	SHELFICE_PARM01	snapshot_mnc	write snapshot using MNC on/off flag

8.6.3.4 SHELFICE Description

In the light of isomorphic equations for pressure and height coordinates, the ice shelf topography on top of the water column has a similar role as (and in the language of Marshall et al. (2004) [MAC+04], is isomorphic to) the orography and the pressure boundary conditions at the bottom of the fluid for atmospheric and oceanic models in pressure coordinates. The total pressure p_{tot} in the ocean can be divided into the pressure at the top of the water column p_{top}, the hydrostatic pressure and the non-hydrostatic pressure contribution p_{NH}:

$$p_{tot} = p_{top} + \int_z^{\eta-h} g\,\rho\,dz + p_{NH} \tag{8.25}$$

with the gravitational acceleration g, the density ρ, the vertical coordinate z (positive upwards), and the dynamic sea-surface height η. For the open ocean, $p_{top} = p_a$ (atmospheric pressure) and $h = 0$. Underneath an ice-shelf that is assumed to be floating in isostatic equilibrium, p_{top} at the top of the water column is the atmospheric pressure p_a plus the weight of the ice-shelf. It is this weight of the ice-shelf that has to be provided as a boundary condition at the top of the water column (in run-time parameter SHELFICEloadAnomalyFile). The weight is conveniently computed by integrating a density profile ρ^*, that is constant in time and corresponds to the sea-water replaced by ice, from $z = 0$

to a "reference" ice-shelf draft at $z = -h$ (Beckmann et al. (1999) [BHT99]), so that

$$p_{top} = p_a + \int_{-h}^{0} g\,\rho^*\,dz \tag{8.26}$$

Underneath the ice shelf, the "sea-surface height" η is the deviation from the "reference" ice-shelf draft h. During a model integration, η adjusts so that the isostatic equilibrium is maintained for sufficiently slow and large scale motion.

In MITgcm, the total pressure anomaly p'_{tot} which is used for pressure gradient computations is defined by substracting a purely depth dependent contribution $-g\rho_0 z$ with a constant reference density ρ_0 from p_{tot}. (8.25) becomes

$$p_{tot} = p_{top} - g\rho_0(z+h) + g\rho_0\eta + \int_{z}^{\eta-h} g(\rho-\rho_0)\,dz + p_{NH} \tag{8.27}$$

and after rearranging

$$p'_{tot} = p'_{top} + g\rho_0\eta + \int_{z}^{\eta-h} g(\rho-\rho_0)\,dz + p_{NH}$$

with $p'_{tot} = p_{tot} + g\,\rho_0\,z$ and $p'_{top} = p_{top} - g\,\rho_0\,h$. The non-hydrostatic pressure contribution p_{NH} is neglected in the following.

In practice, the ice shelf contribution to p_{top} is computed by integrating (8.26) from $z = 0$ to the bottom of the last fully dry cell within the ice shelf:

$$p_{top} = g \sum_{k'=1}^{n-1} \rho^*_{k'} \Delta z_{k'} + p_a \tag{8.28}$$

where n is the vertical index of the first (at least partially) "wet" cell and $\Delta z_{k'}$ is the thickness of the k'-th layer (counting downwards). The pressure anomaly for evaluating the pressure gradient is computed in the center of the "wet" cell k as

$$p'_k = p'_{top} + g\rho_n\eta + g \sum_{k'=n}^{k} \left((\rho_{k'} - \rho_0)\Delta z_{k'} \frac{1 + H(k'-k)}{2} \right) \tag{8.29}$$

where $H(k'-k) = 1$ for $k' < k$ and 0 otherwise.

Setting SHELFICEboundaryLayer `=.TRUE.` introduces a simple boundary layer that reduces the potential noise problem at the cost of increased vertical mixing. For this purpose the water temperature at the k-th layer abutting ice shelf topography for use in the heat flux parameterizations is computed as a mean temperature $\overline{\theta}_k$ over a boundary layer of the same thickness as the layer thickness Δz_k:

$$\overline{\theta}_k = \theta_k h_k + \theta_{k+1}(1 - h_k) \tag{8.30}$$

where $h_k \in [0, 1]$ is the fractional layer thickness of the k-th layer (see Figure 8.10). The original contributions due to ice shelf-ocean interaction g_θ to the total tendency terms G_θ in the time-stepping equation $\theta^{n+1} = f(\theta^n, \Delta t, G^n_\theta)$ are

$$g_{\theta,k} = \frac{Q}{\rho_0 c_p h_k \Delta z_k} \text{ and } g_{\theta,k+1} = 0 \tag{8.31}$$

for layers k and $k+1$ (c_p is the heat capacity). Averaging these terms over a layer thickness Δz_k (e.g., extending from the ice shelf base down to the dashed line in cell C) and applying the averaged tendency to cell A (in layer k) and to the appropriate fraction of cells C (in layer $k+1$) yields

$$g^*_{\theta,k} = \frac{Q}{\rho_0 c_p \Delta z_k} \tag{8.32}$$

$$g^*_{\theta,k+1} = \frac{Q}{\rho_0 c_p \Delta z_k} \frac{\Delta z_k (1 - h_k)}{\Delta z_{k+1}} \tag{8.33}$$

(8.33) describes averaging over the part of the grid cell $k+1$ that is part of the boundary layer with tendency $g^*_{\theta,k}$ and the part with no tendency. Salinity is treated in the same way. The momentum equations are not modified.

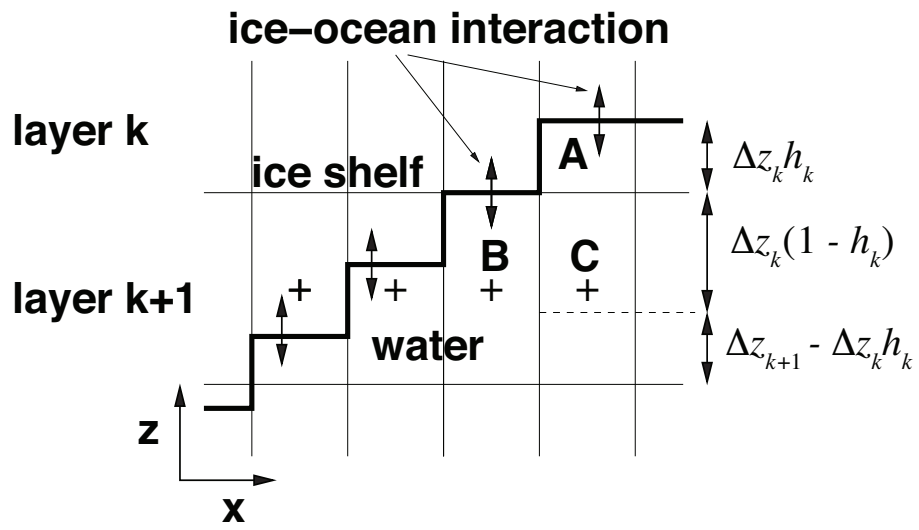

Figure 8.10: Schematic of a vertical section of the grid at the base of an ice shelf. Grid lines are thin; the thick line is the model's representation of the ice shelf-water interface. Plus signs mark the position of pressure points for pressure gradient computations. The letters A, B, and C mark specific grid cells for reference. h_k is the fractional cell thickness so that $h_k \Delta z_k$ is the actual cell thickness.

Three-Equations-Thermodynamics

Freezing and melting form a boundary layer between ice shelf and ocean. Phase transitions at the boundary between saline water and ice imply the following fluxes across the boundary: the freshwater mass flux q (< 0 for melting); the heat flux that consists of the diffusive flux through the ice, the latent heat flux due to melting and freezing and the heat that is carried by the mass flux; and the salinity that is carried by the mass flux, if the ice has a non-zero salinity S_I. Further, the position of the interface between ice and ocean changes because of q, so that, say, in the case of melting the volume of sea water increases. As a consequence salinity and temperature are modified.

The turbulent exchange terms for tracers at the ice-ocean interface are generally expressed as diffusive fluxes. Following Jenkins et al. (2001) [JHH01], the boundary conditions for a tracer take into account that this boundary is not a material surface. The implied upward freshwater flux q (in mass units, negative for melting) is included in the boundary conditions for the temperature and salinity equation as an advective flux:

$$\rho K \frac{\partial X}{\partial z}\bigg|_b = (\rho \gamma_X - q)(X_b - X) \tag{8.34}$$

where tracer X stands for either temperature T or salinity S. X_b is the tracer at the interface (taken to be at freezing), X is the tracer at the first interior grid point, ρ is the density of seawater, and γ_X is the turbulent exchange coefficient (in units of an exchange velocity). The left hand side of (8.34) is shorthand for the (downward) flux of tracer X across the boundary. T_b, S_b and the freshwater flux q are obtained from solving a system of three equations that is derived from the heat and freshwater balance at the ice ocean interface.

In this so-called three-equation-model (e.g., Hellmer and Olbers (1989) [HO89], Jenkins et al. (2001) [JHH01]) the heat balance at the ice-ocean interface is expressed as

$$c_p \rho \gamma_T (T - T_b) + \rho_I c_{p,I} \kappa \frac{(T_S - T_b)}{h} = -Lq \tag{8.35}$$

where ρ is the density of sea-water, $c_p = 3974\,\mathrm{J\,kg^{-1}K^{-1}}$ is the specific heat capacity of water and γ_T the turbulent exchange coefficient of temperature. The value of γ_T is discussed in Holland and Jenkins (1999) [HJ99]. $L =$

$334000\,\mathrm{J\,kg^{-1}}$ is the latent heat of fusion. $\rho_I = 920\,\mathrm{kg\,m^{-3}}$, $c_{p,I} = 2000\,\mathrm{J\,kg^{-1}K^{-1}}$, and T_S are the density, heat capacity and the surface temperature of the ice shelf; $\kappa = 1.54 \times 10^{-6}\,\mathrm{m^2 s^{-1}}$ is the heat diffusivity through the ice-shelf and h is the ice-shelf draft. The second term on the right hand side describes the heat flux through the ice shelf. A constant surface temperature $T_S = -20°\mathrm{C}$ is imposed. T is the temperature of the model cell adjacent to the ice-water interface. The temperature at the interface T_b is assumed to be the in-situ freezing point temperature of sea-water T_f which is computed from a linear equation of state

$$T_f = (0.0901 - 0.0575\,S_b)^\circ - 7.61 \times 10^{-4}\,\frac{\mathrm{K}}{\mathrm{dBar}}\,p_b \tag{8.36}$$

with the salinity S_b and the pressure p_b (in dBar) in the cell at the ice-water interface. From the salt budget, the salt flux across the shelf ice-ocean interface is equal to the salt flux due to melting and freezing:

$$\rho\gamma_S(S - S_b) = -q\,(S_b - S_I) \tag{8.37}$$

where $\gamma_S = 5.05 \times 10^{-3}\gamma_T$ is the turbulent salinity exchange coefficient, and S and S_b are defined in analogy to temperature as the salinity of the model cell adjacent to the ice-water interface and at the interface, respectively. Note, that the salinity of the ice shelf is generally neglected ($S_I = 0$). (8.35) to (8.37) can be solved for S_b, T_b, and the freshwater flux q due to melting. These values are substituted into expression (8.34) to obtain the boundary conditions for the temperature and salinity equations of the ocean model. This formulation tends to yield smaller melt rates than the simpler formulation of the ISOMIP protocol because the freshwater flux due to melting decreases the salinity which raises the freezing point temperature and thus leads to less melting at the interface. For a simpler thermodynamics model where S_b is not computed explicitly, for example as in the ISOMIP protocol, (8.34) cannot be applied directly. In this case (8.37) can be used with (8.34) to obtain:

$$\rho K \frac{\partial S}{\partial z}\bigg|_b = q\,(S - S_I)$$

This formulation can be used for all cases for which (8.37) is valid. Further, in this formulation it is obvious that melting ($q < 0$) leads to a reduction of salinity.

The default value of SHELFICEconserve = `.FALSE.` removes the contribution $q(X_b - X)$ from (8.34), making the boundary conditions for temperature non-conservative.

ISOMIP-Thermodynamics

A simpler formulation follows the ISOMIP protocol. The freezing and melting in the boundary layer between ice shelf and ocean is parameterized following Grosfeld et al. (1997) [GGD97]. In this formulation (8.35) reduces to

$$c_p\rho\gamma_T(T - T_b) = -Lq \tag{8.38}$$

and the fresh water flux q is computed from

$$q = -\frac{c_p\rho\gamma_T(T - T_b)}{L} \tag{8.39}$$

In order to use this formulation, set run-time parameter useISOMIPTD = `.TRUE.` in `data.shelfice`.

Remark

The shelfice package and experiments demonstrating its strengths and weaknesses are also described in Losch (2008) [Los08]. However, note that unfortunately the description of the thermodynamics in the appendix of Losch (2008) is wrong.

8.6.3.5 Key subroutines

The main routine is shelfice_thermodynamics.F but note that /pkg/shelfice routines are also called when solving the momentum equations.

```
C     !CALLING SEQUENCE:
C ...
C |-FORWARD_STEP           :: Step forward a time-step ( AT LAST !!! )
C ...
C | |-DO_OCEANIC_PHY       :: Control oceanic physics and parameterization
C ...
C | | |-SHELFICE_THERMODYNAMICS :: main routine for thermodynamics
C                                  with diagnostics
C ...
C | |-THERMODYNAMICS       :: theta, salt + tracer equations driver.
C ...
C | | |-EXTERNAL_FORCING_T :: Problem specific forcing for temperature.
C | | |-SHELFICE_FORCING_T :: apply heat fluxes from ice shelf model
C ...
C | | |-EXTERNAL_FORCING_S :: Problem specific forcing for salinity.
C | | |-SHELFICE_FORCING_S :: apply fresh water fluxes from ice shelf model
C ...
C | |-DYNAMICS             :: Momentum equations driver.
C ...
C | | |-MOM_FLUXFORM       :: Flux form mom eqn. package ( see
C ...
C | | | |-SHELFICE_U_DRAG  :: apply drag along ice shelf to u-equation
C                              with diagnostics
C ...
C | | |-MOM_VECINV         :: Vector invariant form mom eqn. package ( see
C ...
C | | | |-SHELFICE_V_DRAG  :: apply drag along ice shelf to v-equation
C                              with diagnostics
C ...
C  o
```

8.6.3.6 SHELFICE diagnostics

Diagnostics output is available via the diagnostics package (see Section 9). Available output fields are summarized as follows:

```
---------+----+----+---------------+-----------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
---------+----+----+---------------+-----------------
 SHIfwFlx|  1 |SM  |kg/m^2/s       |Ice shelf fresh water flux (positive upward)
 SHIhtFlx|  1 |SM  |W/m^2          |Ice shelf heat flux  (positive upward)
 SHIUDrag| 30 |UU  |m/s^2          |U momentum tendency from ice shelf drag
 SHIVDrag| 30 |VV  |m/s^2          |V momentum tendency from ice shelf drag
 SHIForcT|  1 |SM  |W/m^2          |Ice shelf forcing for theta, >0 increases theta
 SHIForcS|  1 |SM  |g/m^2/s        |Ice shelf forcing for salt, >0 increases salt
```

8.6.3.7 Experiments and tutorials that use shelfice

See the verification experiment isomip for example usage of pkg/shelfice.

8.6.4 STREAMICE Package

Author: Daniel Goldberg

8.6.4.1 Introduction

Package STREAMICE provides a dynamic land ice model for MITgcm. It was created primarily to develop a TAF- and OpenAD-generated ice model adjoint and to provide synchronous ice-ocean coupling through the SHELFICE package. It solves a set of dynamic equations appropriate for floating ice-shelf flow as well as ice-stream and slower ice-sheet flow. It has been tested at the scale of one or several ice streams, but has not been tested at the continental scale.

8.6.4.2 STREAMICE configuration

Compile-time options

pkg/streamice can be included on at compile time in the `packages.conf` file by adding a line `streamice` (see Section 8.1.1).

Parts of the pkg/streamice code can be enabled or disabled at compile time via CPP flags. These options are set in STREAMICE_OPTIONS.h.

Table 8.16: CPP flags used by pkg/streamice.

CPP Flag Name	Default	Description
STREAMICE_CONSTRUCT_MATRIX	#define	explicit construction of matrix for Picard iteration for velocity
STREAMICE_HYBRID_STRESS	#undef	use L1L2 formulation for stress balance (default shallow shelf approx.)
USE_ALT_RLOW	#undef	use package array for rLow rather than model
STREAMICE_GEOM_FILE_SETUP	#undef	use files rather than parameters in STREAMICE_PARM03 to configure boundaries
ALLOW_PETSC	#undef	enable interface to PETSc for velocity solver matrix solve

Enabling the package

Once it has been compiled, pkg/streamice is switched on/off at run-time by setting useSTREAMICE to `.TRUE.` in file `data.pkg`.

Runtime parmeters: general flags and parameters

Run-time parameters are set in file `data.streamice` (read in streamice_readparms.F). General pkg/streamice parameters are set under STREAMICE_PARM01 as described in Table 8.17.

Table 8.17: Run-time parameters and default values (defined under STREAMICE_PARM01 namelist)

Parameter	Default	Description
streamice_density	910	the (uniform) density of land ice (kg/m^3)

Continued on next page

Table 8.17 – continued from previous page

Parameter	Default	Description
streamice_density_ocean_avg	1024	the (uniform) density of ocean (kg/m^3)
n_glen	3	Glen's Flow Law exponent (non-dim.)
eps_glen_min	1e-12	minimum strain rate in Glen's Law (ε_0, yr^{-1})
eps_u_min	1e-6	minimum speed in nonlinear sliding law (u_0, m/yr)
n_basal_friction	1	exponent in nonlinear sliding law (non-dim.)
streamice_cg_tol	1e-6	tolerance of conjugate gradient of linear solve of Picard iteration for velocity
streamice_lower_cg_tol	TRUE	lower CG tolerance when nonlinear residual decreases by fixed factor
streamice_max_cg_iter	2000	maximum iterations in linear solve
streamice_maxcgiter_cpl	0	as above when coupled with pkg/shelfice
streamice_nonlin_tol	1e-6	tolerance of nonlinear residual for velocity (relative to initial)
streamice_max_nl_iter	100	maximum Picard iterations in solve for velocity
streamice_maxnliter_cpl	0	as above when coupled with pkg/shelfice
streamice_nonlin_tol_fp	1e-6	tolerance of relative change for velocity iteration (relative to magnitude)
streamice_err_norm	0	type of norm evaluated for error (p in p-norm; 0 is ∞)
streamice_chkfixedptconvergence	FALSE	terminate velocity iteration based on relative change per iteration
streamice_chkresidconvergence	TRUE	terminate velocity iteration based on residual
streamicethickInit	FILE	method by which to initialize thickness (`FILE` or `PARAM`)
streamicethickFile	' '	thickness initialization file, in meters (rather than parameters in STREAMICE_PARM03)
streamice_move_front	FALSE	allow ice shelf front to advance
streamice_calve_to_mask	FALSE	if streamice_move_front TRUE do not allow to advance beyond streamice_calve_mask
streamicecalveMaskFile	' '	file to initialize streamice_calve_mask
streamice_diagnostic_only	FALSE	do not update ice thickness (velocity solve only)
streamice_CFL_factor	0.5	CFL factor which determine maximum time step for thickness subcycling
streamice_adjDump	0.0	frequency (s) of writing of adjoint fields to file (TAF only)
streamicebasalTracConfig	UNIFORM	method by which to initialize basal traction (`FILE` or `UNIFORM`)
streamicebasalTracFile	' '	basal trac initialization file (see *Units of input files* for units)
C_basal_fric_const	31.71	uniform basal traction value (see *Units of input files* for units)
streamiceGlenConstConfig	UNIFORM	method by which to initialize Glen's constant (`FILE` or `UNIFORM`)
streamiceGlenConstFile	' '	Glen's constant initialization file (see *Units of input files* for units)
B_glen_isothermal	9.461e-18	uniform Glen's constant value (see *Units of input files* for units)
streamiceBdotFile	' '	file to initialize time-indep melt rate (m/yr)
streamiceBdotTimeDepFile	' '	file to initialize time-varying melt rate (m/yr), based on streamice_forcing_period
streamiceTopogFile	' '	topography initialization file (m); requires #define USE_ALT_RLOW
streamiceHmaskFile	' '	streamice_hmask initialization file; requires #define STREAMICE_GEOM_FILE_SETUP
streamiceuFaceBdryFile	' '	streamice'STREAMICE_ufacemask_bdry' initialization file; requires #define STREAMICE_GEOM_FILE_SETUP
streamicevFaceBdryFile	' '	streamice'STREAMICE_vfacemask_bdry'' initialization file; requires #define STREAMICE_GEOM_FILE_SETUP
streamiceuMassFluxFile	' '	mass flux at u-faces init. file (m^2/yr); requires #define STREAMICE_GEOM_FILE_SETUP

Continued on next page

Table 8.17 – continued from previous page

Parameter	Default	Description
streamicevMassFluxFile	' '	mass flux at v-faces init. file (m^2/yr); requires #define STREAMICE_GEOM_FILE_SETUP
streamiceuFluxTimeDepFile	' '	time-depend. mass flux at u-faces file (m^2/yr); requires #define STREAMICE_GEOM_FILE_SETUP
streamicevFluxTimeDepFile	' '	time-depend. mass flux at v-faces file (m^2/yr); requires #define STREAMICE_GEOM_FILE_SETUP
streamiceuNormalStressFile	' '	calving front normal stress parm along u-faces (non-dim.; see *Boundary Stresses*)
streamicevNormalStressFile	' '	calving front normal stress parm along v-faces (non-dim.; see *Boundary Stresses*)
streamiceuShearStressFile	' '	calving front normal stress parm along u-faces (non-dim.; see *Boundary Stresses*)
streamicevShearStressFile	' '	calving front normal stress parm along v-faces (non-dim.; see *Boundary Stresses*)
streamiceuNormalTimeDepFile	' '	time-dependent version of streamiceuNormalStressFile
streamicevNormalTimeDepFile	' '	time-dependent version of streamicevNormalStressFile
streamiceuShearTimeDepFile	' '	time-dependent version of streamiceuShearStressFile
streamicevShearTimeDepFile	' '	time-dependent version of streamicevShearStressFile
streamice_adot_uniform	0	time/space uniform surface accumulation rate (m/yr)
streamice_forcing_period	0	file input frequency for streamice time-dependent forcing fields (s)
streamice_smooth_gl_width	0	thickness range parameter in basal traction smoothing across grounding line (m)

Configuring domain through files

The STREAMICE_GEOM_FILE_SETUP CPP option allows versatility in defining the domain. With this option, the array streamice_hmask must be initialized through a file (streamiceHmaskFile) as must streamice_ufacemask_bdry and streamice_vfacemask_bdry (through streamiceuFaceBdryFile and streamicevFaceBdryFile) as well as u_flux_bdry_SI and v_flux_bdry_SI, volume flux at the boundaries, where appropriate (through streamiceuMassFluxFile and streamicevMassFluxFile). Thickness must be initialized through a file as well (streamicethickFile); streamice_hmask is set to zero where ice thickness is zero, and boundaries between in-domain and out-of-domain cells (according to streamice_hmask) are no-slip by default.

When using this option, it is important that for all internal boundaries, streamice_ufacemask_bdry and streamice_vfacemask_bdry are -1 (this will not be the case if streamiceuFaceBdryFile and streamicevFaceBdryFile are undefined).

In fact, if streamice_hmask is configured correctly, streamice_ufacemask_bdry and streamice_vfacemask_bdry can be set uniformly to -1, UNLESS there are no-stress or flux-condition boundaries in the domain. Where streamice_ufacemask_bdry and streamice_vfacemask_bdry are set to -1, they will be overridden at (a) boundaries where streamice_hmask changes from 1 to -1 (which become no-slip boundaries), and (b) boundaries where streamice_hmask changes from 1 to 0 (which become calving front boundaries).

An example of domain configuration through files can be found in verification/halfpipe_streamice. By default, verification/halfpipe_streamice is compiled with STREAMICE_GEOM_FILE_SETUP undefined, but the user can modify this option. The file verification/halfpipe_streamice/input/data.streamice_geomSetup represents an alternative version of verification/halfpipe_streamice/input/data.streamice in which the appropriate binary files are specified.

Configuring domain through parameters

For a very specific type of domain the boundary conditions and initial thickness can be set via parameters in `data.streamice`. Such a domain will be rectangular. In order to use this option, the STREAMICE_GEOM_FILE_SETUP CPP flag should be undefined.

There are different boundary condition types (denoted within the parameter names) that can be set:

- `noflow`: x- and y-velocity will be zero along this boundary.
- `nostress`: velocity normal to boundary will be zero; there will be no tangential stress along the boundary.
- `fluxbdry`: a mass volume flux is specified along this boundary, which becomes a boundary condition for the thickness advection equation (see *Equations Solved*). Velocities will be zero. The corresponding parameters flux_bdry_val_NORTH, flux_bdry_val_SOUTH, flux_bdry_val_EAST and flux_bdry_val_WEST then set the values.
- `CFBC`: calving front boundary condition, a Neumann condition based on ice thickness and bed depth, is imposed at this boundary (see *Equations Solved*).

Note the above only apply if there is dynamic ice in the cells at the boundary in question. The boundary conditions are then set by specifying the above conditions over ranges of each (north/south/east/west) boundary. The division of each boundary should be exhaustive and the ranges should not overlap. Parameters to initialize boundary conditions (defined under STREAMICE_PARM03 namelist) are listed in Table 8.18.

Table 8.18: Parameters to initialize boundary conditions (defined under STREAMICE_PARM03 namelist)

Parameter	Default	Description
min_x_noflow_NORTH	0	western limit of no-flow region on northern boundary (m)
max_x_noflow_NORTH	0	eastern limit of no-flow region on northern boundary (m)
min_x_noflow_SOUTH	0	western limit of no-flow region on southern boundary (m)
max_x_noflow_SOUTH	0	eastern limit of no-flow region on southern boundary (m)
min_y_noflow_EAST	0	southern limit of no-flow region on eastern boundary (m)
max_y_noflow_EAST	0	northern limit of no-flow region on eastern boundary (m)
min_y_noflow_WEST	0	southern limit of no-flow region on western boundary (m)
max_y_noflow_WEST	0	northern limit of no-flow region on eastern boundary (m)
min_x_nostress_NORTH	0	western limit of no-stress region on northern boundary (m)
max_x_nostress_NORTH	0	eastern limit of no-stress region on northern boundary (m)
min_x_nostress_SOUTH	0	western limit of no-stress region on southern boundary (m)
max_x_nostress_SOUTH	0	eastern limit of no-stress region on southern boundary (m)
min_y_nostress_EAST	0	southern limit of no-stress region on eastern boundary (m)
max_y_nostress_EAST	0	northern limit of no-stress region on eastern boundary (m)
min_y_nostress_WEST	0	southern limit of no-stress region on western boundary (m)
max_y_nostress_WEST	0	northern limit of no-stress region on eastern boundary (m)
min_x_fluxbdry_NORTH	0	western limit of flux-boundary region on northern boundary (m)
max_x_fluxbdry_NORTH	0	eastern limit of flux-boundary region on northern boundary (m)
min_x_fluxbdry_SOUTH	0	western limit of flux-boundary region on southern boundary (m)
max_x_fluxbdry_SOUTH	0	eastern limit of flux-boundary region on southern boundary (m)
min_y_fluxbdry_EAST	0	southern limit of flux-boundary region on eastern boundary (m)
max_y_fluxbdry_EAST	0	northern limit of flux-boundary region on eastern boundary (m)
min_y_fluxbdry_WEST	0	southern limit of flux-boundary region on western boundary (m)
max_y_fluxbdry_WEST	0	northern limit of flux-boundary region on eastern boundary (m)
min_x_CFBC_NORTH	0	western limit of calving front condition region on northern boundary (m)
max_x_CFBC_NORTH	0	eastern limit of calving front condition region on northern boundary (m)

Continued on next page

Table 8.18 – continued from previous page

Parameter	Default	Description
min_x_CFBC_SOUTH	0	western limit of calving front condition region on southern boundary (m)
max_x_CFBC_SOUTH	0	eastern limit of calving front condition region on southern boundary (m)
min_y_CFBC_EAST	0	southern limit of calving front condition region on eastern boundary (m)
max_y_CFBC_EAST	0	northern limit of calving front condition region on eastern boundary (m)
min_y_CFBC_WEST	0	southern limit of calving front condition region on western boundary (m)
max_y_CFBC_WEST	0	northern limit of calving front condition region on eastern boundary (m)
flux_bdry_val_SOUTH	0	volume flux per width entering at flux-boundary on southern boundary (m^2/a)
flux_bdry_val_NORTH	0	volume flux per width entering at flux-boundary on southern boundary (m^2/a)
flux_bdry_val_EAST	0	volume flux per width entering at flux-boundary on southern boundary (m^2/a)
flux_bdry_val_WEST	0	volume flux per width entering at flux-boundary on southern boundary (m^2/a)

8.6.4.3 Description

Equations Solved

The model solves for 3 dynamic variables: x-velocity (u), y-velocity (v), and thickness (h). There is also a variable that tracks coverage of fractional cells, discussed in *Ice front advance*.

By default the model solves the "shallow shelf approximation" (SSA) for velocity. The SSA is appropriate for floating ice (ice shelf) or ice flowing over a low-friction bed (e.g., Macayeal (1989) [Mac89]). The SSA consists of the x-momentum balance:

$$\partial_x(h\nu(4\dot{\varepsilon}_{xx} + 2\dot{\varepsilon}_{yy})) + \partial_y(2h\nu\dot{\varepsilon}_{xy}) - \tau_{bx} = \rho g h \frac{\partial s}{\partial x} \tag{8.40}$$

the y-momentum balance:

$$\partial_x(2h\nu\dot{\varepsilon}_{xy}) + \partial_y(h\nu(4\dot{\varepsilon}_{yy} + 2\dot{\varepsilon}_{xx})) - \tau_{by} = \rho g h \frac{\partial s}{\partial y} \tag{8.41}$$

where ρ is ice density, g is gravitational acceleration, and s is surface elevation. ν, τ_{bi} and $\dot{\varepsilon}_{ij}$ are ice viscosity, basal drag, and the strain rate tensor, respectively, all explained below.

From the velocity field, thickness evolves according to the continuity equation:

$$h_t + \nabla \cdot (h\vec{u}) = \dot{a} - \dot{b} \tag{8.42}$$

Where $\dot{b}$ is a basal mass balance (e.g., melting due to contact with the ocean), positive where there is melting. This is a field that can be specified through a file. At the moment surface mass balance $\dot{a}$ can only be set as uniform. Where ice is grounded, surface elevation is given by

$$s = R + h$$

where $R(x, y)$ is the bathymetry, and the basal elevation b is equal to R. If ice is floating, then the assumption of hydrostasy and constant density gives

$$s = (1 - \frac{\rho}{\rho_w})h,$$

where ρ_w is a representative ocean density, and $b = -(\rho/\rho_w)h$. Again by hydrostasy, floation is assumed wherever

$$h \leq -\frac{\rho_w}{\rho} R$$

is satisfied. Floatation criteria is stored in float_frac_streamice, equal to 1 where ice is grounded, and equal to 0 where ice is floating.

The strain rates ε_{ij} are generalized to the case of orthogonal curvilinear coordinates, to include the "metric" terms that arise when casting the equations of motion on a sphere or projection on to a sphere (see *Finite-volume discretization of the stress tensor divergence*). Thus

$$\begin{aligned}\dot{\varepsilon}_{xx} =& u_x + k_1 v,\\ \dot{\varepsilon}_{yy} =& v_y + k_1 u,\\ \dot{\varepsilon}_{xy} =& \frac{1}{2}(u_y + v_x) + k_1 u + k_2 v.\end{aligned}$$

ν has the form arising from Glen's law

$$\nu = \frac{1}{2} A^{-\frac{1}{n}} \left(\dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xy}^2 + \dot{\varepsilon}_{min}^2\right)^{\frac{1-n}{2n}} \tag{8.43}$$

though the form is slightly different if a hybrid formulation is used.

Whether τ_b is nonzero depends on whether the floatation condition is satisfied. Currently this is determined simply on an instantaneous cell-by-cell basis (unless subgrid interpolation is used), as is the surface elevation s, but possibly this should be rethought if the effects of tides are to be considered. $\vec{\tau}_b$ has the form

$$\vec{\tau}_b = C(|\vec{u}|^2 + u_{min}^2)^{\frac{m-1}{2}} \vec{u} \tag{8.44}$$

Again, the form is slightly different if a hybrid formulation is to be used.

The momentum equations are solved together with appropriate boundary conditions, discussed below. In the case of a calving front boundary condition (CFBC), the boundary condition has the following form:

$$(h\nu(4\dot{\varepsilon}_{xx} + 2\dot{\varepsilon}_{yy}))n_x + (2h\nu\dot{\varepsilon}_{xy})n_y = \frac{1}{2} g \left(\rho h^2 - \rho_w b^2\right) n_x \tag{8.45}$$

$$(2h\nu\dot{\varepsilon}_{xy})n_x + (h\nu(4\dot{\varepsilon}_{yy} + 2\dot{\varepsilon}_{xx}))n_y = \frac{1}{2} g \left(\rho h^2 - \rho_w b^2\right) n_y. \tag{8.46}$$

Here $\vec{n}$ is the normal to the boundary, and $R(x, y)$ is the bathymetry.

Hybrid SIA-SSA stress balance

The SSA does not take vertical shear stress or strain rates (e.g., σ_{xz}, $\partial u/\partial z$) into account. Although there are other terms in the stress tensor, studies have found that in all but a few cases, vertical shear and longitudinal stresses (represented by the SSA) are sufficient to represent glaciological flow. pkg/streamice can allow for representation of vertical shear, although the approximation is made that longitudinal stresses are depth-independent. The stress balance is referred to as "hybrid" because it is a joining of the SSA and the "shallow ice approximation" (SIA), which accounts only for vertical shear. Such hybrid formulations have been shown to be valid over a larger range of conditions than SSA (Goldberg 2011) [Gol11].

In the hybrid formulation, $\overline{u}$ and $\overline{v}$, the depth-averaged $x-$ and $y-$ velocities, replace u and v in (8.40), (8.41), and (8.42), and gradients such as u_x are replaced by $(\overline{u})_x$. Viscosity becomes

$$\nu = \frac{1}{2} A^{-\frac{1}{n}} \left(\dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xy}^2 + \frac{1}{4}u_z^2 + \frac{1}{4}v_z^2 + \dot{\varepsilon}_{min}^2\right)^{\frac{1-n}{2n}}$$

In the formulation for τ_b, u_b, the horizontal velocity at u_b is used instead. The details are given in Goldberg (2011) [Gol11].

Ice front advance

By default all mass flux across calving boundaries is considered lost. However, it is possible to account for this flux and potential advance of the ice shelf front. If streamice_move_front is TRUE, then a partial-area formulation is used.

The algorithm is based on Albrecht et al. (2011) [AMH+11]. In this scheme, for empty or partial cells adjacent to a calving front, a **reference** thickness h_{ref} is found, defined as an average over the thickness of all neighboring cells that flow into the cell. The total volume input over a time step is added to the volume of ice already in the cell, whose partial area coverage is then updated based on the volume and reference thickness. If the area coverage reaches 100% in a time step, then the additional volume is cascaded into adjacent empty or partial cells.

If streamice_calve_to_mask is TRUE, this sets a limit to how far the front can advance, even if advance is allowed. The front will not advance into cells where the array streamice_calve_mask is not equal to 1. This mask must be set through a binary input file to allow the front to advance past its initial position.

No calving parameterization is implemented in pkg/streamice. However, front advancement is a precursor for such a development to be added.

Units of input files

The inputs for basal traction (streamicebasalTracFile, C_basal_fric_const) and ice stiffness (streamiceGlenConstFile, B_glen_isothermal) require specific units. For ice stiffness (*A* in (8.43)), $B = A^{-1/n}$ is specified; or, more accurately, its square root $A^{-1/(2n)}$ is specified (this is to ensure positivity of *B* by squaring the input). The units of streamiceGlenConstFile and B_glen_isothermal are $\mathrm{Pa}^{1/2}\ \mathrm{yr}^{1/(2n)}$ where n is n_glen.

streamicebasalTracFile and C_basal_fric_const initialize the basal traction (*C* in (8.44)). Again $C^{1/2}$ is directly specified rather than *C* to ensure positivity. The units are $\mathrm{Pa}^{1/2}(\mathrm{m\ yr}^{-1})^{n_b}$ where n_b is n_basal_friction.

8.6.4.4 Numerical Details

The momentum balance is solved via iteration on viscosity (Goldberg 2011 [Gol11]). At each iteration, a linear elliptic differential equation is solved via a finite-element method using bilinear basis functions. The velocity solution "lives" on cell corners, while thickness "lives" at cell centers (Figure 8.11). The cell-centered thickness is then evolved using a second-order slope-limited finite-volume scheme, with the velocity field from the previous solve. To represent the flow of floating ice, basal stress terms are multiplied by an array float_frac_streamice, a cell-centered array which determines where ice meets the floation condition.

The computational domain of pkg/streamice (which may be smaller than the array/grid as defined by SIZE.h and GRID.h) is determined by a number of mask arrays within pkg/streamice. They are

- $hmask$ (streamice_hmask): equal to 1 (ice-covered), 0 (open ocean), 2 (partly-covered), or -1 (out of domain)
- $umask$ (streamice_umask): equal to 1 (an "active" velocity node), 3 (a Dirichlet node), or 0 (zero velocity)
- $vmask$ (streamice_vmask): similar to umask
- $ufacemaskbdry$ (streamice_ufacemask_bdry): equal to -1 (interior face), 0 (no-slip), 1 (no-stress), 2 (calving stress front), or 4 (flux input boundary); when 4, then u_flux_bdry_SI must be initialized, through binary or parameter file
- $vfacemaskbdry$ (streamice_vfacemask_bdry): similar to $ufacemaskbdry$

$hmask$ is defined at cell centers, like h. $umask$ and $vmask$ are defined at cell nodes, like velocities. $ufacemaskbdry$ and $vfacemaskbdry$ are defined at cell faces, like velocities in a C-grid - but unless one sets `#define` STREAMICE_GEOM_FILE_SETUP in STREAMICE_OPTIONS.h, the values are only relevant at the boundaries of the grid.

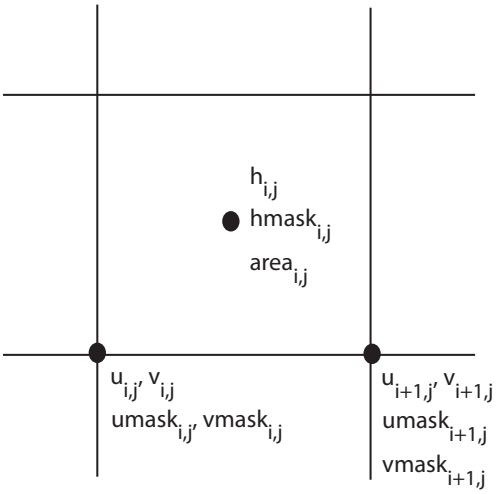

Figure 8.11: Grid locations of thickness (h), velocity (u,v), area, and various masks.

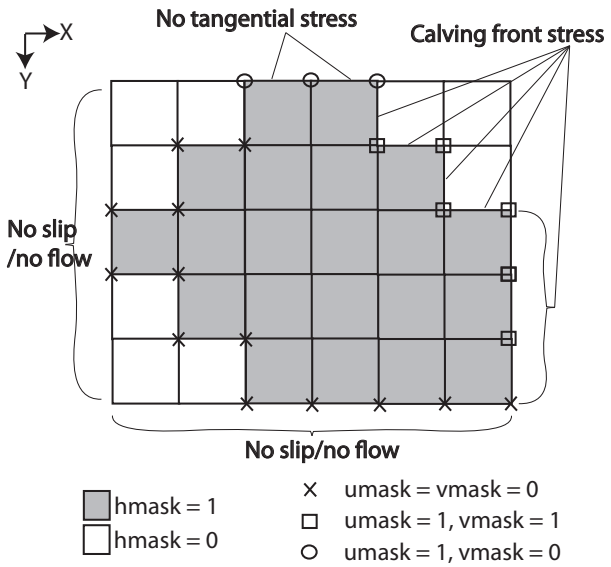

Figure 8.12: Hypothetical configuration, detailing the meaning of thickness and velocity masks and their role in controlling boundary conditions.

The values of $umask$ and $vmask$ determine which nodal values of u and v are involved in the solve for velocities. These masks are not configured directly by the user, but are re-initialized based on streamice_hmask, streamice_ufacemask_bdry and streamice_vfacemask_bdry at each time step. Figure 8.12 demonstrates how these values are set in various cells.

With $umask$ and $vmask$ appropriately initialized, subroutine streamice_vel_solve.F can proceed rather generally. Contributions are only evaluated if $hmask = 1$ in a given cell, and a given nodal basis function is only considered if $umask = 1$ or $vmask = 1$ at that node.

8.6.4.5 Additional Features

PETSc

There is an option to use PETSc for the matrix solve component of the velocity solve, and this has been observed to give a 3- or 4-fold improvement in performance over the inbuilt conjugate gradient solver in a number of cases. To use this option, the CPP option ALLOW_PETSC must be defined, and MITgcm must be compiled with the `-mpi` flag (see Section 3.5.4). However, often a system-specific installation of PETSc is required. If you wish to use PETSc with pkg/streamice, please contact the author.

Boundary Stresses

The calving front boundary conditions (8.45) and (8.46) are intended for ice fronts bordering open ocean. However, there may be reasons to apply different Neumann conditions at these locations, e.g., one might want to represent force associated with ice melange, or to represent parts of the ice shelf that are not resolved, as in Goldberg et al. (2015) [GHJS15]. The user can then modify these boundary conditions in the form

$$(h\nu(4\dot{\varepsilon}_{xx} + 2\dot{\varepsilon}_{yy}))n_x + (2h\nu\dot{\varepsilon}_{xy})n_y = \frac{1}{2}g\left(\rho h^2 - \rho_w b^2\right)n_x + \sigma n_x + \tau n_y$$

$$(2h\nu\dot{\varepsilon}_{xy})n_x + (h\nu(4\dot{\varepsilon}_{yy} + 2\dot{\varepsilon}_{xx}))n_y = \frac{1}{2}g\left(\rho h^2 - \rho_w b^2\right)n_y + \sigma n_y + \tau n_x$$

In these equations, σ and τ represent normal and shear stresses at the boundaries of cells. They are not specified directly, but through coefficients γ_σ and γ_τ:

$$\sigma = \frac{1}{2}g\left(\rho h^2 - \rho_w b^2\right)\gamma_\sigma$$

$$\tau = \frac{1}{2}g\left(\rho h^2 - \rho_w b^2\right)\gamma_\tau$$

γ_σ is specified through streamiceuNormalStressFile, streamicevNormalStressFile, streamiceuNormalTimeDepFile, streamicevNormalTimeDepFile and γ_τ is specified through streamiceuShearStressFile, streamicevShearStressFile, streamiceuShearTimeDepFile, and streamicevShearTimeDepFile. Within the file names, the `u` and `v` determine whether the values are specified along horizontal (u-) faces and vertical (v-) faces. The values will only have an effect if they are specified along calving front boundaries (see *Configuring domain through files*).

8.6.4.6 Adjoint

The STREAMICE package is adjointable using both TAF (Goldberg et al. 2013 [GH13]) and OpenAD (Goldberg et al. 2016 [GNHU16]). In OpenAD, the fixed-point method of [Chr94] is implemented, greatly reducing the memory requirements and also improving performance when PETSc is used.

Verification experiments with both OpenAD and TAF are located in the verification/halfpipe_streamice (see below).

8.6.4.7 Key Subroutines

Top-level routine: streamice_timestep.F (called from model/src/do_oceanic_phys.F)

```
   CALLING SEQUENCE
...
 streamice_timestep (called from DO_OCEANIC_PHYS)
 |
 |-- #ifdef ALLOW_STREAMICE_TIMEDEP_FORCING
 |    STREAMICE_FIELDS_LOAD
 |   #endif
 |
 |--#if (defined (ALLOW_STREAMICE_OAD_FP))
 |    STREAMICE_VEL_SOLVE_OPENAD
 |  #else
 |    STREAMICE_VEL_SOLVE
 |    |
 |    |-- STREAMICE_DRIVING_STRESS
 |    |
 |    | [ITERATE ON FOLLOWING]
 |    |
 |    |-- STREAMICE_CG_WRAPPER
 |    |   |
 |    |   |-- STREAMICE_CG_SOLVE
 |    |       #ifdef ALLOW_PETSC
 |    |        STREAMICE_CG_SOLVE_PETSC
 |    |       #endif
 |    |
 |    |-- #ifdef STREAMICE_HYBRID_STRESS
 |         STREAMICE_VISC_BETA_HYBRID
 |        #else
 |         STREAMICE_VISC_BETA
 |        #endif
 |
 |-- STREAMICE_ADVECT_THICKNESS
 |   |
 |   |-- STREAMICE_ADV_FRONT
 |
 |-- STREAMICE_UPD_FFRAC_UNCOUPLED
 |
```

8.6.4.8 STREAMICE diagnostics

Diagnostics output is available via the diagnostics package (*Packages II - Diagnostics and I/O*). Available output fields are summarized in the following table:

```
------------------------------------------------------------------------
<-Name->|Levs|  mate |<- code ->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
SI_Uvel |  1 |       |UZ      L1|m/a             |Ice stream x-velocity
SI_Vvel |  1 |       |VZ      L1|m/a             |Ice stream y-velocity
SI_Thick|  1 |       |SM      L1|m               |Ice stream thickness
SI_area |  1 |       |SM      L1|m^2             |Ice stream cell area coverage
SI_float|  1 |       |SM      L1|none            |Ice stream grounding indicator
SI_hmask|  1 |       |SM      L1|none            |Ice stream thickness mask
SI_usurf|  1 |       |SM      L1|none            |Ice stream surface x-vel
```

(continues on next page)

(continued from previous page)

```
SI_vsurf|  1 |       |SM      L1|none            |Ice stream surface y-vel
SI_ubase|  1 |       |SM      L1|none            |Ice stream basal x-vel
SI_vbase|  1 |       |SM      L1|none            |Ice stream basal y-vel
SI_taubx|  1 |       |SM      L1|none            |Ice stream basal x-stress
SI_tauby|  1 |       |SM      L1|none            |Ice stream basal y-stress
SI_selev|  1 |       |SM      L1|none            |Ice stream surface elev
```

8.6.4.9 Experiments and tutorials that use streamice

The verification/halfpipe_streamice experiment uses pkg/streamice.

8.7 Biogeochemistry Packages

8.7.1 GCHEM Package

8.7.1.1 Introduction

This package has been developed as interface to the PTRACERS package. The purpose is to provide a structure where various (any) tracer experiments can be added to the code. For instance there are placeholders for routines to read in parameters needed for any tracer experiments, a routine to read in extra fields required for the tracer code, routines for either external forcing or internal interactions between tracers and routines for additional diagnostics relating to the tracers. Note that the gchem package itself is only a means to call the subroutines used by specific biogeochemical experiments, and does not "do" anything on its own.

There are two examples: **cfc** which looks at 2 tracers with a simple external forcing and **dic** with 4,5 or 6 tracers whose tendency terms are related to one another. We will discuss these here only as how they provide examples to use this package.

8.7.1.2 Key subroutines and parameters

FRAMEWORK

The different biogeochemistry frameworks (e.g. cfc of dic) are specified in the packages_conf file. *GCHEM_OPTIONS.h* includes the compiler options to be used in any experiment. An important compiler option is #define GCHEM_SEPARATE_FORCING which determined how and when the tracer forcing is applied (see discussion on Forcing below). See section on dic for some additional flags that can be set for that experiment.

There are further runtime parameters set in *data.gchem* and kept in common block *GCHEM.h*. These runtime options include:

• Parameters to set the timing for periodic forcing files to be loaded are: *gchem_ForcingPeriod*, *gchem_ForcingCycle*. The former is how often to load, the latter is how often to cycle through those fields (eg. period couple be monthly and cycle one year). This is used in *dic* and *cfc*, with gchem_ForcingPeriod=0 meaning no periodic forcing.

• **nsubtime** is the integer number of extra timesteps required by the tracer experiment. This will give a timestep of **deltaTtracer**/**nsubtime** for the dependencies between tracers. The default is one.

• File names - these are several filenames than can be read in for external fields needed in the tracer forcing - for instance wind speed is needed in both DIC and CFC packages to calculate the air-sea exchange of gases. Not all file names will be used for every tracer experiment.

• **gchem_int_** are variable names for run-time set integer numbers. (Currently 1 through 5).

• **gchem_rl_** are variable names for run-time set real numbers. (Currently 1 through 5).

• Note that the old **tIter0** has been replaced by **PTRACERS_Iter0** which is set in data.ptracers instead.

INITIALIZATION

The values set at runtime in data.gchem are read in using *gchem_readparms.F* which is called from packages_readparms.F. This will include any external forcing files that will be needed by the tracer experiment. There are two routine used to initialize parameters and fields needed by the experiment packages. These are *gchem_init_fixed.F* which is called from *packages_init_fixed.F*, and *gchem_init_vari.F* called from packages_init_variable.F. The first should be used to call a subroutine specific to the tracer experiment which sets fixed parameters, the second should call a subroutine specific to the tracer experiment which sets (or initializes) time fields that will vary with time.

LOADING FIELDS

External forcing fields used by the tracer experiment are read in by a subroutine (specific to the tracer experiment) called from *gchem_fields_load.F*. This latter is called from *forward_step.F*.

FORCING

Tracer fields are advected-and-diffused by the ptracer package. Additional changes (e.g. surface forcing or interactions between tracers) to these fields are taken care of by the gchem interface. For tracers that are essentially passive (e.g. CFC's) but may have some surface boundary conditions this can easily be done within the regular tracer timestep. In this case *gchem_calc_tendency.F* is called from *forward_step.F*, where the reactive (as opposed to the advective diffusive) tendencies are computed. These tendencies, stored on the 3D field **gchemTendency**, are added to the passive tracer tendencies **gPtr** in *gchem_add_tendency.F*, which is called from *ptracers_forcing.F*. For tracers with more complicated dependencies on each other, and especially tracers which require a smaller timestep than deltaTtracer, it will be easier to use *gchem_forcing_sep.F* which is called from forward_step.F. There is a compiler option set in *GCHEM_OPTIONS.h* that determines which method is used: #define GCHEM_SEPARATE_FORCING does the latter where tracers are forced separately from the advection-diffusion code, and #undef GCHEM_SEPARATE_FORCING includes the forcing in the regular timestepping.

DIAGNOSTICS

This package also also used the passive tracer routine *ptracers_monitor.F* which prints out tracer statistics as often as the model dynamic statistic diagnostics (dynsys) are written (or as prescribed by the runtime flag **PTRACERS_monitorFreq**, set in *data.ptracers*). There is also a placeholder for any tracer experiment specific diagnostics to be calculated and printed to files. This is done in *gchem_diags.F*. For instance the time average CO2 air-sea fluxes, and sea surface pH (among others) are written out by *dic_biotic_diags.F* which is called from *gchem_diags.F*.

8.7.1.3 GCHEM Diagnostics

These diagnostics are particularly for the **dic** package.

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
DICBIOA | 15 |SM P    MR      |mol/m3/sec      |Biological Productivity (mol/m3/s)
DICCARB | 15 |SM P    MR      |mol eq/m3/sec   |Carbonate chg-biol prod and remin␣
↪(mol eq/m3/s)
DICTFLX |  1 |SM P    L1      |mol/m3/sec      |Tendency of DIC due to air-sea exch␣
↪(mol/m3/s)
DICOFLX |  1 |SM P    L1      |mol/m3/sec      |Tendency of O2 due to air-sea exch␣
↪(mol/m3/s)
DICCFLX |  1 |SM P    L1      |mol/m2/sec      |Flux of CO2 - air-sea exch (mol/m2/s)
```

(continues on next page)

(continued from previous page)

```
DICPCO2 |  1 |SM P    M1      |atm             |Partial Pressure of CO2 (atm)
DICPHAV |  1 |SM P    M1      |dimensionless   |pH (dimensionless)
```

8.7.1.4 Do's and Don'ts

The pkg ptracer is required with use with this pkg. Also, as usual, the runtime flag **useGCHEM** must be set to **.TRUE.** in **data.pkg**. By itself, gchem pkg will read in **data.gchem** and will write out gchem diagnostics. It requires tracer experiment specific calls to do anything else (for instance the calls to dic and cfc pkgs).

8.7.1.5 Reference Material

8.7.1.6 Experiments and tutorials that use gchem

- Global Ocean biogeochemical tutorial, in tutorial_global_oce_biogeo verification directory, described in section [sec:eg-biogeochem_tutorial] uses gchem and dic
- Global Ocean cfc tutorial, in tutorial_cfc_offline verification directory, uses gchem and cfc (and offline) described in [sec:eg-offline-cfc]
- Global Ocean online cfc example in cfc_example verification directory, uses gchem and cfc

8.7.2 DIC Package

8.7.2.1 Introduction

This is one of the biogeochemical packages handled from the pkg gchem. The main purpose of this package is to consider the cycling of carbon in the ocean. It also looks at the cycling of phosphorous and potentially oxygen and iron. There are four standard tracers DIC, ALK, $PO4$, DOP and also possibly $O2$ and Fe. The air-sea exchange of CO_2 and O_2 are handled as in the OCMIP experiments (reference). The export of biological matter is computed as a function of available light and PO_4 (and Fe). This export is remineralized at depth according to a Martin curve (again, this is the same as in the OCMIP experiments). There is also a representation of the carbonate flux handled as in the OCMIP experiments. The air-sea exchange on CO_2 is affected by temperature, salinity and the pH of the surface waters. The pH is determined following the method of Follows et al. For more details of the equations see section [sec:eg-biogeochem_tutorial].

8.7.2.2 Key subroutines and parameters

INITIALIZATION

DIC_ABIOTIC.h contains the common block for the parameters and fields needed to calculate the air-sea flux of CO_2 and O_2. The fixed parameters are set in *dic_abiotic_param* which is called from *gchem_init_fixed.F*. The parameters needed for the biotic part of the calculations are initialized in *dic_biotic_param* and are stored in *DIC_BIOTIC.h*. The first guess of pH is calculated in *dic_surfforcing_init.F*.

LOADING FIELDS

The air-sea exchange of CO_2 and O_2 need wind, atmospheric pressure (although the current version has this hardwired to 1), and sea-ice coverage. The calculation of pH needs silica fields. These fields are read in in *dic_fields_load.F*. These fields are initialized to zero in *dic_ini_forcing.F*. The fields for interpolating are in common block in *DIC_LOAD.h*.

FORCING

The tracers are advected-diffused in *ptracers_integrate.F*. The updated tracers are passed to *dic_biotic_forcing.F* where the effects of the air-sea exchange and biological activity and remineralization are calculated and the tracers are updated for a second time. Below we discuss the subroutines called from *dic_biotic_forcing.F*.

Air-sea exchange of CO_2 is calculated in *dic_surfforcing*. Air-Sea Exchange of CO_2 depends on T,S and pH. The determination of pH is done in *carbon_chem.F*. There are three subroutines in this file: *carbon_coeffs* which determines the coefficients for the carbon chemistry equations; *calc_pco2* which calculates the pH using a Newton-Raphson method; and *calc_pco2_approx* which uses the much more efficient method of Follows et al. The latter is hard-wired into this package, the former is kept here for completeness.

Biological productivity is determined following Dutkiewicz et al. (2005) and is calculated in *bio_export.F* The light in each latitude band is calculate in *insol.F*, unless using one of the flags listed below. The formation of hard tissue (carbonate) is linked to the biological productivity and has an effect on the alkalinity - the flux of carbonate is calculated in *car_flux.F*, unless using the flag listed below for the Friis et al (2006) scheme. The flux of phosphate to depth where it instantly remineralized is calculated in *phos_flux.F*.

The dilution or concentration of carbon and alkalinity by the addition or subtraction of freshwater is important to their surface patterns. These "virtual" fluxes can be calculated by the model in several ways. The older scheme is done following OCMIP protocols (see more in Dutkiewicz et al 2005), in the subroutines *dic_surfforcing.F* and *alk_surfforcing.F*. To use this you need to set in GCHEM_OPTIONS.h: #define ALLOW_OLD_VIRTUALFLUX. But this can also be done by the ptracers pkg if this is undefined. You will then need to set the concentration of the tracer in rainwater and potentially a reference tracer value in data.ptracer (PTRACERS_EvPrRn, and PTRACERS_ref respectively).

Oxygen air-sea exchange is calculated in *o2_surfforcing.F*.

Iron chemistry (the amount of free iron) is taken care of in *fe_chem.F*.

DIAGNOSTICS

Averages of air-sea exchanges, biological productivity, carbonate activity and pH are calculated. These are initialized to zero in *dic_biotic_init* and are stored in common block in *DIC_BIOTIC.h*.

COMPILE TIME FLAGS

These are set in GCHEM_OPTIONS.h:

DIC_BIOTIC: needs to be set for dic to work properly (should be fixed sometime).

ALLOW_O2: include the tracer oxygen.

ALLOW_FE: include the tracer iron. Note you will need an iron dust file set in data.gchem in this case.

MINFE: limit the iron, assuming precpitation of any excess free iron.

CAR_DISS: use the calcium carbonate scheme of Friis et al 2006.

ALLOW_OLD_VIRTUALFLUX: use the old OCMIP style virtual flux for alklinity adn carbon (rather than doing it through pkg/ptracers).

READ_PAR: read the light (photosynthetically available radiation) from a file set in data.gchem.

USE_QSW: use the numbers from QSW to be the PAR. Note that a file for Qsw must be supplied in data, or Qsw must be supplied by an atmospheric model.

If the above two flags are not set, the model calculates PAR in insol.F as a function of latitude and year day.

USE_QSW_UNDERICE: if using a sea ice model, or if the Qsw variable has the seaice fraction already taken into account, this flag must be set.

AD_SAFE: will use a tanh function instead of a max function - this is better if using the adjoint

DIC_NO_NEG: will include some failsafes in case any of the variables become negative. (This is advicable).
ALLOW_DIC_COST: was used for calculating cost function (but hasn't been updated or maintained, so not sure if it works still)

8.7.2.3 Do's and Don'ts

This package must be run with both ptracers and gchem enabled. It is set up for at least 4 tracers, but there is the provision for oxygen and iron. Note the flags above.

8.7.2.4 Reference Material

Dutkiewicz. S., A. Sokolov, J. Scott and P. Stone, 2005: A Three-Dimensional Ocean-Seaice-Carbon Cycle Model and its Coupling to a Two-Dimensional Atmospheric Model: Uses in Climate Change Studies, Report 122, Joint Program of the Science and Policy of Global Change, M.I.T., Cambridge, MA.

Follows, M., T. Ito and S. Dutkiewicz, 2006: A Compact and Accurate Carbonate Chemistry Solver for Ocean Biogeochemistry Models. *Ocean Modeling*, 12, 290-301.

Friis, K., R. Najjar, M.J. Follows, and S. Dutkiewicz, 2006: Possible overestimation of shallow-depth calcium carbonate dissolution in the ocean, *Global Biogeochemical Cycles*, 20, GB4019, doi:10.1029/2006GB002727.

8.7.2.5 Experiments and tutorials that use dic

- Global Ocean tutorial, in tutorial_global_oce_biogeo verification directory, described in section [sec:eg-biogeochem_tutorial]

CHAPTER 9

Packages II - Diagnostics and I/O

MITgcm includes several packages related to input and output during a model integration. The packages described in this chapter are related to the choice of input/output fields and their on-disk format.

9.1 pkg/diagnostics – A Flexible Infrastructure

9.1.1 Introduction

This section of the documentation describes the diagnostics package (pkg/diagnostics) available within MITgcm. A large selection of model diagnostics is available for output. In addition to the diagnostic quantities pre-defined within MITgcm, there exists the option, in any code setup, to define a new diagnostic quantity and include it as part of the diagnostic output with the addition of a single subroutine call in the routine where the field is computed. As a matter of philosophy, no diagnostic is enabled as default, thus each user must specify the exact diagnostic information required for an experiment. This is accomplished by enabling the specific diagnostics of interest from the list of *available diagnostics*. Additional diagnostic quantities, defined within different MITgcm packages, are available and are listed in the diagnostic list subsection of the manual section associated with each relevant package. Instructions for enabling diagnostic output and defining new diagnostic quantities are found in Section 9.1.4 of this document.

Once a diagnostic is enabled, MITgcm will continually increment an array specifically allocated for that diagnostic whenever the appropriate quantity is computed. A counter is defined which records how many times each diagnostic quantity has been incremented. Several special diagnostics are included in the list of *available diagnostics*. Quantities referred to as "counter diagnostics" are defined for selected diagnostics which record the frequency at which a diagnostic is incremented separately for each model grid location. Quantities referred to as "user diagnostics" are included to facilitate defining new diagnostics for a particular experiment.

9.1.2 Equations

Not relevant.

9.1.3 Key Subroutines and Parameters

There are several utilities within MITgcm available to users to enable, disable, clear, write and retrieve model diagnostics, and may be called from any routine. The available utilities and the CALL sequences are listed below.

diagnostics_addtolist.F: This routine is the underlying interface routine for defining a new permanent diagnostic in the main model or in a package. The calling sequence is:

```
  CALL DIAGNOSTICS_ADDTOLIST (
O     diagNum,
I     diagName, diagCode, diagUnits, diagTitle, diagMate,
I     myThid )

where:
  diagNum   = diagnostic Id number - Output from routine
  diagName  = name of diagnostic to declare
  diagCode  = parser code for this diagnostic
  diagUnits = field units for this diagnostic
  diagTitle = field description for this diagnostic
  diagMate  = diagnostic mate number
  myThid    = my Thread Id number
```

diagnostics_fill.F: This is the main user interface routine to the diagnostics package. This routine will increment the specified diagnostic quantity with a field sent through the argument list.

```
   CALL DIAGNOSTICS_FILL(
  I             inpFld, diagName,
  I             kLev, nLevs, bibjFlg, bi, bj, myThid )

where:
   inpFld   = Field to increment diagnostics array
   diagName = diagnostic identificator name (8 characters long)
   kLev     = Integer flag for vertical levels:
              > 0 (any integer): WHICH single level to increment in qdiag.
              0,-1 to increment "nLevs" levels in qdiag,
              0 : fill-in in the same order as the input array
              -1: fill-in in reverse order.
   nLevs    = indicates Number of levels of the input field array
              (whether to fill-in all the levels (kLev<1) or just one (kLev>0))
   bibjFlg  = Integer flag to indicate instructions for bi bj loop
            = 0 indicates that the bi-bj loop must be done here
            = 1 indicates that the bi-bj loop is done OUTSIDE
            = 2 indicates that the bi-bj loop is done OUTSIDE
                 AND that we have been sent a local array (with overlap regions)
                 (local array here means that it has no bi-bj dimensions)
            = 3 indicates that the bi-bj loop is done OUTSIDE
                 AND that we have been sent a local array
                 AND that the array has no overlap region (interior only)
              NOTE - bibjFlg can be NEGATIVE to indicate not to increment counter
   bi       = X-direction tile number - used for bibjFlg=1-3
   bj       = Y-direction tile number - used for bibjFlg=1-3
   myThid   =  my thread Id number
```

diagnostics_scale_fill.F: This is a possible alternative routine to diagnostics_fill.F which performs the same functions and has an additional option to scale the field before filling or raise the field to a power before filling.

```
   CALL DIAGNOSTICS_SCALE_FILL(
  I             inpFld, scaleFact, power, diagName,
```

(continues on next page)

(continued from previous page)

```
  I             kLev, nLevs, bibjFlg, bi, bj, myThid )


where all the arguments are the same as for DIAGNOSTICS_FILL with
the addition of:
   scaleFact   = Scaling factor to apply to the input field product
   power       = Integer power to which to raise the input field (after scaling)
```

diagnostics_fract_fill.F: This is a specific alternative routine to diagnostics_scale_fill.F for the case of a diagnostics which is associated to a fraction-weight factor (referred to as the diagnostics "counter-mate"). This fraction-weight field is expected to vary during the simulation and is provided as argument to diagnostics_fract_fill.F in order to perform fraction-weighted time-average diagnostics. Note that the fraction-weight field has to correspond to the diagnostics counter-mate which has to be filled independently with a call to diagnostics_fill.F.

```
   CALL DIAGNOSTICS_FRACT_FILL(
  I             inpFld, fractFld, scaleFact, power, diagName,
  I             kLev, nLevs, bibjFlg, bi, bj, myThid )


where all the arguments are the same as for DIAGNOSTICS_SCALE_FILL with
the addition of:
   fractFld    = fraction used for weighted average diagnostics
```

diagnostics_is_on.F: Function call to inquire whether a diagnostic is active and should be incremented. Useful when there is a computation that must be done locally before a call to diagnostics_fill.F. The call sequence:

```
   flag = DIAGNOSTICS_IS_ON( diagName, myThid )

where:
   diagName = diagnostic identificator name (8 characters long)
   myThid   = my thread Id number
```

diagnostics_count.F: This subroutine increments the diagnostics counter only. In general, the diagnostics counter is incremented at the same time as the diagnostics is filled, by calling diagnostics_fill.F. However, there are few cases where the counter is not incremented during the filling (e.g., when the filling is done level per level but level 1 is skipped) and needs to be done explicitly with a call to subroutine diagnostics_count.F. The call sequence is:

```
   CALL DIAGNOSTICS_COUNT(
  I                        diagName, bi, bj, myThid )

where:
   diagName  = name of diagnostic to increment the counter
   bi        = X-direction tile number, or 0 if called outside bi,bj loops
   bj        = Y-direction tile number, or 0 if called outside bi,bj loops
   myThid    = my thread Id number
```

The diagnostics are computed at various times and places within MITgcm. Because MITgcm may employ a staggered grid, diagnostics may be computed at grid box centers, corners, or edges, and at the middle or edge in the vertical. Some diagnostics are scalars, while others are components of vectors. An internal array is defined which contains information concerning various grid attributes of each diagnostic. The gdiag array (in common block diagnostics in file DIAGNOSTICS.h) is internally defined as a character*16 variable, and is equivalenced to a character*1 "parse" array in output in order to extract the grid-attribute information. The gdiag array is described in Table 9.1.

Table 9.1: Diagnostic Parsing Array

Array	Value	Description
parse(1)	→ S	scalar diagnostic
	→ U	U-vector component diagnostic
	→ V	V-vector component diagnostic
parse(2)	→ U	C-grid U-point
	→ V	C-grid V-point
	→ M	C-grid mass point
	→ Z	C-grid vorticity (corner) point
parse(3)	→	used for level-integrated output: cumulate levels
	→ r	same but cumulate product by model level thickness
	→ R	same but cumulate product by hFac & level thickness
parse(4)	→ P	positive definite diagnostic
	→ A	Adjoint variable diagnostic
parse(5)	→ C	with counter array
	→ P	post-processed (not filled up) from other diags
	→ D	disable diagnostic for output
parse(6-8)		retired, formerly 3-digit mate number
parse(9)	→ U	model level + $\frac{1}{2}$
	→ M	model level middle
	→ L	model level - $\frac{1}{2}$
parse(10)	→ 0	levels = 0
	→ 1	levels = 1
	→ R	levels = Nr
	→ L	levels = MAX(Nr,NrPhys)
	→ M	levels = MAX(Nr,NrPhys) - 1
	→ G	levels = ground_level number
	→ I	levels = seaice_level number
	→ X	free levels option (need to be set explicitly)

As an example, consider a diagnostic whose associated gdiag parameter is equal to "UURMR". From gdiag we can determine that this diagnostic is a U-vector component located at the C-grid U-point, model mid-level (M) with Nr levels (last R).

In this way, each diagnostic in the model has its attributes (i.e., vector or scalar, C-grid location, etc.) defined internally. The output routines use this information in order to determine what type of transformations need to be performed. Any interpolations are done at the time of output rather than during each model step. In this way the user has flexibility in determining the type of output gridded data.

9.1.4 Usage Notes

9.1.4.1 Using available diagnostics

To use the diagnostics package, other than enabling it in `packages.conf` and turning the useDiagnostics flag in `data.pkg` to `.TRUE.`, there are two further steps the user must take to enable the diagnostics package for output of quantities that are already defined in MITgcm under an experiment's configuration of packages. A parameter file `data.diagnostics` must be supplied in the run directory, and the file DIAGNOSTICS_SIZE.h must be included in the code directory. The steps for defining a new (permanent or experiment-specific temporary) diagnostic quantity will be outlined later.

The namelist in parameter file `data.diagnostics` will activate a user-defined list of diagnostics quantities to be computed, specify the frequency and type of output, the number of levels, and the name of all the separate output files.

A sample `data.diagnostics` namelist file:

```
# Diagnostic Package Choices
#--------------------
#  dumpAtLast (logical): always write output at the end of simulation (default=F)
#  diag_mnc   (logical): write to NetCDF files (default=useMNC)
#--for each output-stream:
#  fileName(n) : prefix of the output file name (max 80c long) for outp.stream n
#  frequency(n):< 0 : write snap-shot output every |frequency| seconds
#               > 0 : write time-average output every frequency seconds
#  timePhase(n)     : write at time = timePhase + multiple of |frequency|
#    averagingFreq  : frequency (in s) for periodic averaging interval
#    averagingPhase : phase     (in s) for periodic averaging interval
#    repeatCycle    : number of averaging intervals in 1 cycle
#  levels(:,n) : list of levels to write to file (Notes: declared as REAL)
#                when this entry is missing, select all common levels of this list
#  fields(:,n) : list of selected diagnostics fields (8.c) in outp.stream n
#                (see "available_diagnostics.log" file for the full list of diags)
#  missing_value(n) : missing value for real-type fields in output file "n"
#  fileFlags(n)     : specific code (8c string) for output file "n"
#--------------------
 &DIAGNOSTICS_LIST
  fields(1:2,1) = 'UVEL    ','VVEL    ',
   levels(1:5,1) = 1.,2.,3.,4.,5.,
   filename(1) = 'diagout1',
  frequency(1) = 86400.,
  fields(1:2,2) = 'THETA   ','SALT    ',
   filename(2) = 'diagout2',
  fileflags(2) = ' P1     ',
  frequency(2) = 3600.,
 &

 &DIAG_STATIS_PARMS
 &
```

In this example, there are two output files that will be generated for each tile and for each output time. The first set of output files has the prefix `diagout1`, does time averaging every 86400. seconds, (frequency is 86400.), and will write fields which are multiple-level fields at output levels 1-5. The names of diagnostics quantities are `UVEL` and `VVEL`. The second set of output files has the prefix diagout2, does time averaging every 3600. seconds, includes fields with all levels, and the names of diagnostics quantities are `THETA` and `SALT`.

The user must assure that enough computer memory is allocated for the diagnostics and the output streams selected for a particular experiment. This is accomplished by modifying the file DIAGNOSTICS_SIZE.h and including it in the experiment code directory. The parameters that should be checked are called numDiags, numLists, numperList, and diagSt_size.

numDiags (and diagSt_size): All MITgcm diagnostic quantities are stored in the single diagnostic array gdiag which is located in the file and has the form:

```
_RL  qdiag(1-Olx,sNx+Olx,1-Olx,sNx+Olx,numDiags,nSx,nSy)
_RL  qSdiag(0:nStats,0:nRegions,diagSt_size,nSx,nSy)
COMMON / DIAG_STORE_R / qdiag, qSdiag
```

The first two-dimensions of diagSt_size correspond to the horizontal dimension of a given diagnostic, and the third dimension of diagSt_size is used to identify diagnostic fields and levels combined. In order to minimize the memory requirement of the model for diagnostics, the default MITgcm executable is compiled with room for only one horizontal diagnostic array, or with numDiags set to Nr. In order for the user to enable more than one 3-D diagnostic, the size of the diagnostics common must be expanded to accommodate the desired diagnostics. This can be accomplished by

manually changing the parameter numDiags in the file . numDiags should be set greater than or equal to the sum of all the diagnostics activated for output each multiplied by the number of levels defined for that diagnostic quantity. For the above example, there are four multiple level fields, which the available diagnostics list (see below) indicates are defined at the MITgcm vertical resolution, Nr. The value of numDiags in DIAGNOSTICS_SIZE.h would therefore be equal to 4*Nr, or, say 40 if Nr=10.

numLists and numperList: The parameter numLists must be set greater than or equal to the number of separate output streams that the user specifies in the namelist file `data.diagnostics`. The parameter numperList corresponds to the maximum number of diagnostics requested per output streams.

9.1.4.2 Adjoint variables

The diagnostics package can also be used to print adjoint state variables. Using the diagnostics package as opposed to using the standard 'adjoint dump' options allows one to take advantage of all the averaging and post processing routines available to other diagnostics variables.

Currently, the available adjoint state variables are:

```
110 |ADJetan |  1 |       |SM A    M1|dJ/m            |dJ/dEtaN: Sensitivity to sea␣
↪surface height anomaly
111 |ADJuvel | 50 |   112 |UURA    MR|dJ/(m/s)        |dJ/dU: Sensitivity to zonal␣
↪velocity
112 |ADJvvel | 50 |   111 |VVRA    MR|dJ/(m/s)        |dJ/dV: Sensitivity to␣
↪meridional velocity
113 |ADJwvel | 50 |       |WM A    LR|dJ/(m/s)        |dJ/dW: Sensitivity to vertical␣
↪velocity
114 |ADJtheta| 50 |       |SMRA    MR|dJ/degC         |dJ/dTheta: Sensitivity to␣
↪potential temperature
115 |ADJsalt | 50 |       |SMRA    MR|dJ/psu          |dJ/dSalt: Sensitivity to␣
↪salinity
```

Some notes to the user

1. This feature is currently untested with OpenAD.
2. This feature does not work with the divided adjoint.
3. *adEtaN* is broken in addummy_in_stepping.F so the output through diagnostics is zeros just as with the standard 'adjoint dump' method.
4. The diagStats options are not available for these variables.
5. Adjoint variables are recognized by checking the 10 character variable *diagCode*. To add a new adjoint variable, set the 4th position of *diagCode* to A (notice this is the case for the list of available adjoint variables).

Using pkg/diagnostics for adjoint variables

1. Make sure the following flag is defined in either AUTODIFF_OPTIONS.h or *ECCO_CPPOPTIONS.h* if that is being used.

```
#define ALLOW_AUTODIFF_MONITOR
```

2. Be sure to increase *numlists* and *numDiags* appropriately in DIAGNOSTICS_SIZE.h. Safe values are e.g. 10-20 and 500-1000 respectively.

3. Specify desired variables in `data.diagnostics` as any other variable, as in the following example or as in this data.diagnostics. Note however, adjoint and forward diagnostic variables cannot be in the same list. That is, a single *fields(:,:)* list cannot contain both adjoint and forward variables.

```
&DIAGNOSTICS_LIST
# ---
  fields(1:5,1) = 'ADJtheta','ADJsalt ',
                     'ADJuvel ','ADJvvel ','ADJwvel '
  filename(1) = 'diags/adjState_3d_snaps',
  frequency(1)=-86400.0,
  timePhase(1)=0.0,
#---
  fields(1:5,2) = 'ADJtheta','ADJsalt ',
                     'ADJuvel ','ADJvvel ','ADJwvel '
  filename(2) = 'diags/adjState_3d_avg',
  frequency(2)= 86400.0,
#---
&
```

Note: the diagnostics package automatically provides a phase shift of $frequency/2$, so specify *timePhase = 0* to match output from *adjDumpFreq*.

9.1.4.3 Adding new diagnostics to the code

In order to define and include as part of the diagnostic output any field that is desired for a particular experiment, two steps must be taken. The first is to enable the "User Diagnostic" in `data.diagnostics`. This is accomplished by adding one of the "User Diagnostic" field names (see *available diagnostics*):`UDIAG1` through `UDIAG10`, for multi-level fields, or `SDIAG1` through `SDIAG10` for single level fields) to the `data.diagnostics` namelist in one of the output streams. The second step is to add a call to diagnostics_fill.F from the subroutine in which the quantity desired for diagnostic output is computed.

In order to add a new diagnostic to the permanent set of diagnostics that the main model or any package contains as part of its diagnostics menu, the subroutine diagnostics_addtolist.F should be called during the initialization phase of the main model or package. For the main model, the call should be made from subroutine diagnostics_main_init.F, and for a package, the call should probably be made from from inside the particular package's init_fixed routine. A typical code sequence to set the input arguments to diagnostics_addtolist.F would look like:

```
 diagName  = 'RHOAnoma'
 diagTitle = 'Density Anomaly (=Rho-rhoConst)'
 diagUnits = 'kg/m^3          '
 diagCode  = 'SMR     MR      '
 CALL DIAGNOSTICS\_ADDTOLIST( diagNum,
I          diagName, diagCode, diagUnits, diagTitle, 0, myThid )
```

If the new diagnostic quantity is associated with either a vector pair or a diagnostic counter, the diagMate argument must be provided with the proper index corresponding to the "mate". The output argument from diagnostics_addtolist.F that is called diagNum here contains a running total of the number of diagnostics defined in the code up to any point during the run. The sequence number for the next two diagnostics defined (the two components of the vector pair, for instance) will be diagNum+1 and diagNum+2. The definition of the first component of the vector pair must fill the "mate" segment of the diagCode as diagnostic number diagNum+2. Since the subroutine increments diagNum, the definition of the second component of the vector fills the "mate" part of diagCode with diagNum. A code sequence for this case would look like:

```
 diagName  = 'UVEL    '
 diagTitle = 'Zonal Component of Velocity (m/s)'
```

(continues on next page)

(continued from previous page)

```
 diagUnits = 'm/s             '
 diagCode  = 'UUR     MR      '
 diagMate  = diagNum + 2
 CALL DIAGNOSTICS_ADDTOLIST( diagNum,
I   diagName, diagCode, diagUnits, diagTitle, diagMate, myThid )

 diagName  = 'VVEL    '
 diagTitle = 'Meridional Component of Velocity (m/s)'
 diagUnits = 'm/s             '
 diagCode  = 'VVR     MR      '
 diagMate  = diagNum
 CALL DIAGNOSTICS_ADDTOLIST( diagNum,
I   diagName, diagCode, diagUnits, diagTitle, diagMate, myThid )
```

9.1.4.4 MITgcm kernel available diagnostics list:

```
------------------------------------------------------------------------
<-Name->|Levs|  mate |<- code ->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------------------------
SDIAG1  |  1 |       |SM      L1|user-defined    |User-Defined   Surface   Diagnostic␣
↪ #1
SDIAG2  |  1 |       |SM      L1|user-defined    |User-Defined   Surface   Diagnostic␣
↪ #2
SDIAG3  |  1 |       |SM      L1|user-defined    |User-Defined   Surface   Diagnostic␣
↪ #3
SDIAG4  |  1 |       |SM      L1|user-defined    |User-Defined   Surface   Diagnostic␣
↪ #4
SDIAG5  |  1 |       |SM      L1|user-defined    |User-Defined   Surface   Diagnostic␣
↪ #5
SDIAG6  |  1 |       |SM      L1|user-defined    |User-Defined   Surface   Diagnostic␣
↪ #6
SDIAG7  |  1 |       |SU      L1|user-defined    |User-Defined U.pt Surface␣
↪Diagnostic #7
SDIAG8  |  1 |       |SV      L1|user-defined    |User-Defined V.pt Surface␣
↪Diagnostic #8
SDIAG9  |  1 |    10 |UU      L1|user-defined    |User-Defined U.vector Surface Diag.␣
↪ #9
SDIAG10 |  1 |     9 |VV      L1|user-defined    |User-Defined V.vector Surface Diag.
↪#10
UDIAG1  | 50 |       |SM      MR|user-defined    |User-Defined Model-Level Diagnostic␣
↪ #1
UDIAG2  | 50 |       |SM      MR|user-defined    |User-Defined Model-Level Diagnostic␣
↪ #2
UDIAG3  | 50 |       |SMR     MR|user-defined    |User-Defined Model-Level Diagnostic␣
↪ #3
UDIAG4  | 50 |       |SMR     MR|user-defined    |User-Defined Model-Level Diagnostic␣
↪ #4
UDIAG5  | 50 |       |SU      MR|user-defined    |User-Defined U.pt Model-Level Diag.␣
↪ #5
UDIAG6  | 50 |       |SV      MR|user-defined    |User-Defined V.pt Model-Level Diag.␣
↪ #6
UDIAG7  | 50 |    18 |UUR     MR|user-defined    |User-Defined U.vector Model-Lev␣
↪Diag.#7
UDIAG8  | 50 |    17 |VVR     MR|user-defined    |User-Defined V.vector Model-Lev␣
↪Diag.#8
```

(continues on next page)

(continued from previous page)

```
UDIAG9   | 50 |        |SM      ML|user-defined    |User-Defined Phys-Level  Diagnostic␣
↪ #9
UDIAG10 | 50 |        |SM      ML|user-defined    |User-Defined Phys-Level  Diagnostic
↪#10
SDIAGC   |  1 |    22 |SM  C   L1|user-defined    |User-Defined Counted Surface␣
↪Diagnostic
SDIAGCC |  1 |        |SM      L1|count           |User-Defined Surface Diagnostic␣
↪Counter
ETAN     |  1 |        |SM      M1|m               |Surface Height Anomaly
ETANSQ   |  1 |        |SM P    M1|m^2             |Square of Surface Height Anomaly
DETADT2 |  1 |        |SM      M1|m^2/s^2         |Square of Surface Height Anomaly␣
↪Tendency
THETA    | 50 |        |SMR     MR|degC            |Potential Temperature
SALT     | 50 |        |SMR     MR|psu             |Salinity
RELHUM   | 50 |        |SMR     MR|percent         |Relative Humidity
SALTanom| 50 |        |SMR     MR|psu             |Salt anomaly (=SALT-35; g/kg)
UVEL     | 50 |    31 |UUR     MR|m/s             |Zonal Component of Velocity (m/s)
VVEL     | 50 |    30 |VVR     MR|m/s             |Meridional Component of Velocity (m/
↪s)
WVEL     | 50 |        |WM      LR|m/s             |Vertical Component of Velocity (r_
↪units/s)
THETASQ | 50 |        |SMRP    MR|degC^2          |Square of Potential Temperature
SALTSQ   | 50 |        |SMRP    MR|(psu)^2         |Square of Salinity
SALTSQan| 50 |        |SMRP    MR|(psu)^2         |Square of Salt anomaly (=(SALT-35)^
↪2 (g^2/kg^2)
UVELSQ   | 50 |    37 |UURP    MR|m^2/s^2         |Square of Zonal Comp of Velocity (m^
↪2/s^2)
VVELSQ   | 50 |    36 |VVRP    MR|m^2/s^2         |Square of Meridional Comp of␣
↪Velocity (m^2/s^2)
WVELSQ   | 50 |        |WM P    LR|m^2/s^2         |Square of Vertical Comp of Velocity
UE_VEL_C| 50 |    40 |UMR     MR|m/s             |Eastward Velocity (m/s) (cell␣
↪center)
VN_VEL_C| 50 |    39 |VMR     MR|m/s             |Northward Velocity (m/s) (cell␣
↪center)
UV_VEL_C| 50 |    41 |UMR     MR|m^2/s^2         |Product of horizontal Comp of␣
↪velocity (cell center)
UV_VEL_Z| 50 |    42 |UZR     MR|m^2/s^2         |Meridional Transport of Zonal␣
↪Momentum (m^2/s^2)
WU_VEL   | 50 |        |WU      LR|m.m/s^2         |Vertical Transport of Zonal Momentum
WV_VEL   | 50 |        |WV      LR|m.m/s^2         |Vertical Transport of Meridional␣
↪Momentum
UVELMASS| 50 |    46 |UUr     MR|m/s             |Zonal Mass-Weighted Comp of␣
↪Velocity (m/s)
VVELMASS| 50 |    45 |VVr     MR|m/s             |Meridional Mass-Weighted Comp of␣
↪Velocity (m/s)
WVELMASS| 50 |        |WM      LR|m/s             |Vertical Mass-Weighted Comp of␣
↪Velocity
PhiVEL   | 50 |    45 |SMR P   MR|m^2/s           |Horizontal Velocity Potential (m^2/
↪s)
PsiVEL   | 50 |    48 |SZ  P   MR|m.m^2/s         |Horizontal Velocity Stream-Function
UTHMASS  | 50 |    51 |UUr     MR|degC.m/s        |Zonal Mass-Weight Transp of Pot Temp
VTHMASS  | 50 |    50 |VVr     MR|degC.m/s        |Meridional Mass-Weight Transp of␣
↪Pot Temp
WTHMASS  | 50 |        |WM      LR|degC.m/s        |Vertical Mass-Weight Transp of Pot␣
↪Temp (K.m/s)
USLTMASS| 50 |    54 |UUr     MR|psu.m/s         |Zonal Mass-Weight Transp of Salinity
VSLTMASS| 50 |    53 |VVr     MR|psu.m/s         |Meridional Mass-Weight Transp of␣
↪Salinity
```

(continues on next page)

(continued from previous page)

```
WSLTMASS| 50 |        |WM      LR|psu.m/s          |Vertical Mass-Weight Transp of␣
↪Salinity
UVELTH  | 50 |    57 |UUR     MR|degC.m/s         |Zonal Transport of Pot Temp
VVELTH  | 50 |    56 |VVR     MR|degC.m/s         |Meridional Transport of Pot Temp
WVELTH  | 50 |       |WM      LR|degC.m/s         |Vertical Transport of Pot Temp
UVELSLT | 50 |    60 |UUR     MR|psu.m/s          |Zonal Transport of Salinity
VVELSLT | 50 |    59 |VVR     MR|psu.m/s          |Meridional Transport of Salinity
WVELSLT | 50 |       |WM      LR|psu.m/s          |Vertical Transport of Salinity
UVELPHI | 50 |    63 |UUr     MR|m^3/s^3          |Zonal Mass-Weight Transp of␣
↪Pressure Pot.(p/rho) Anomaly
VVELPHI | 50 |    62 |VVr     MR|m^3/s^3          |Merid. Mass-Weight Transp of␣
↪Pressure Pot.(p/rho) Anomaly
RHOAnoma| 50 |        |SMR     MR|kg/m^3           |Density Anomaly (=Rho-rhoConst)
RHOANOSQ| 50 |        |SMRP    MR|kg^2/m^6        |Square of Density Anomaly (=(Rho-
↪rhoConst)^2)
URHOMASS| 50 |    67 |UUr     MR|kg/m^2/s         |Zonal Transport of Density
VRHOMASS| 50 |    66 |VVr     MR|kg/m^2/s         |Meridional Transport of Density
WRHOMASS| 50 |       |WM      LR|kg/m^2/s         |Vertical Transport of Density
WdRHO_P | 50 |       |WM      LR|kg/m^2/s         |Vertical velocity times delta^
↪k(Rho)_at-const-P
WdRHOdP | 50 |       |WM      LR|kg/m^2/s         |Vertical velocity times delta^
↪k(Rho)_at-const-T,S
PHIHYD  | 50 |       |SMR     MR|m^2/s^2          |Hydrostatic Pressure Pot.(p/rho)␣
↪Anomaly
PHIHYDSQ| 50 |       |SMRP    MR|m^4/s^4          |Square of Hyd. Pressure Pot.(p/rho)␣
↪Anomaly
PHIBOT  |  1 |       |SM      M1|m^2/s^2          |Bottom Pressure Pot.(p/rho) Anomaly
PHIBOTSQ|  1 |       |SM P    M1|m^4/s^4          |Square of Bottom Pressure Pot.(p/
↪rho) Anomaly
PHIHYDcR| 50 |       |SMR     MR|m^2/s^2          |Hydrostatic Pressure Pot.(p/rho)␣
↪Anomaly @ const r
MXLDEPTH|  1 |       |SM      M1|m                |Mixed-Layer Depth (>0)
DRHODR  | 50 |       |SM      LR|kg/m^4          |Stratification: d.Sigma/dr (kg/m3/r_
↪unit)
CONVADJ | 50 |       |SMR     LR|fraction         |Convective Adjustment Index [0-1]
oceTAUX |  1 |    80 |UU      U1|N/m^2            |zonal surface wind stress, >0␣
↪increases uVel
oceTAUY |  1 |    79 |VV      U1|N/m^2            |meridional surf. wind stress, >0␣
↪increases vVel
atmPload|  1 |       |SM      U1|Pa               |Atmospheric pressure loading
sIceLoad|  1 |       |SM      U1|kg/m^2           |sea-ice loading (in Mass of␣
↪ice+snow / area unit)
oceFWflx|  1 |       |SM      U1|kg/m^2/s         |net surface Fresh-Water flux into␣
↪the ocean (+=down), >0 decreases salinity
oceSflux|  1 |       |SM      U1|g/m^2/s          |net surface Salt flux into the␣
↪ocean (+=down), >0 increases salinity
oceQnet |  1 |       |SM      U1|W/m^2            |net surface heat flux into the␣
↪ocean (+=down), >0 increases theta
oceQsw  |  1 |       |SM      U1|W/m^2            |net Short-Wave radiation (+=down), >
↪0 increases theta
oceFreez|  1 |       |SM      U1|W/m^2            |heating from freezing of sea-water␣
↪(allowFreezing=T)
TRELAX  |  1 |       |SM      U1|W/m^2            |surface temperature relaxation, >0␣
↪increases theta
SRELAX  |  1 |       |SM      U1|g/m^2/s          |surface salinity relaxation, >0␣
↪increases salt
surForcT|  1 |       |SM      U1|W/m^2            |model surface forcing for␣
↪Temperature, >0 increases theta
```

(continues on next page)

(continued from previous page)

```
surForcS|  1 |       |SM      U1|g/m^2/s         |model surface forcing for Salinity,␣
↪>0 increases salinity
TFLUX   |  1 |       |SM      U1|W/m^2           |total heat flux (match heat-content␣
↪variations), >0 increases theta
SFLUX   |  1 |       |SM      U1|g/m^2/s         |total salt flux (match salt-content␣
↪variations), >0 increases salt
RCENTER | 50 |       |SM      MR|m               |Cell-Center Height
RSURF   |  1 |       |SM      M1|m               |Surface Height
TOTUTEND| 50 |    97 |UUR     MR|m/s/day         |Tendency of Zonal Component of␣
↪Velocity
TOTVTEND| 50 |    96 |VVR     MR|m/s/day         |Tendency of Meridional Component of␣
↪Velocity
TOTTTEND| 50 |       |SMR     MR|degC/day        |Tendency of Potential Temperature
TOTSTEND| 50 |       |SMR     MR|psu/day         |Tendency of Salinity
------------------------------------------------------------------------------
<-Name->|Levs|  mate |<- code ->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------------
MoistCor| 50 |       |SM      MR|W/m^2           |Heating correction due to moist␣
↪thermodynamics
gT_Forc | 50 |       |SMR     MR|degC/s          |Potential Temp. forcing tendency
gS_Forc | 50 |       |SMR     MR|psu/s           |Salinity forcing tendency
AB_gT   | 50 |       |SMR     MR|degC/s          |Potential Temp. tendency from Adams-
↪Bashforth
AB_gS   | 50 |       |SMR     MR|psu/s           |Salinity tendency from Adams-
↪Bashforth
gTinAB  | 50 |       |SMR     MR|degC/s          |Potential Temp. tendency going in␣
↪Adams-Bashforth
gSinAB  | 50 |       |SMR     MR|psu/s           |Salinity tendency going in Adams-
↪Bashforth
AB_gU   | 50 |   108 |UUR     MR|m/s^2           |U momentum tendency from Adams-
↪Bashforth
AB_gV   | 50 |   107 |VVR     MR|m/s^2           |V momentum tendency from Adams-
↪Bashforth
ADVr_TH | 50 |       |WM      LR|degC.m^3/s      |Vertical   Advective Flux of Pot.
↪Temperature
ADVx_TH | 50 |   111 |UU      MR|degC.m^3/s      |Zonal      Advective Flux of Pot.
↪Temperature
ADVy_TH | 50 |   110 |VV      MR|degC.m^3/s      |Meridional Advective Flux of Pot.
↪Temperature
DFrE_TH | 50 |       |WM      LR|degC.m^3/s      |Vertical Diffusive Flux of Pot.
↪Temperature (Explicit part)
DFxE_TH | 50 |   114 |UU      MR|degC.m^3/s      |Zonal      Diffusive Flux of Pot.
↪Temperature
DFyE_TH | 50 |   113 |VV      MR|degC.m^3/s      |Meridional Diffusive Flux of Pot.
↪Temperature
DFrI_TH | 50 |       |WM      LR|degC.m^3/s      |Vertical Diffusive Flux of Pot.
↪Temperature (Implicit part)
SM_x_TH | 50 |   117 |UM      MR|degC            |Pot.Temp.   1rst Order Moment Sx
SM_y_TH | 50 |   116 |VM      MR|degC            |Pot.Temp.   1rst Order Moment Sy
SM_z_TH | 50 |       |SM      MR|degC            |Pot.Temp.   1rst Order Moment Sz
SMxx_TH | 50 |   120 |UM      MR|degC            |Pot.Temp.   2nd Order Moment Sxx
SMyy_TH | 50 |   119 |VM      MR|degC            |Pot.Temp.   2nd Order Moment Syy
SMzz_TH | 50 |       |SM      MR|degC            |Pot.Temp.   2nd Order Moment Szz
SMxy_TH | 50 |       |SM      MR|degC            |Pot.Temp.   2nd Order Moment Sxy
SMxz_TH | 50 |   124 |UM      MR|degC            |Pot.Temp.   2nd Order Moment Sxz
SMyz_TH | 50 |   123 |VM      MR|degC            |Pot.Temp.   2nd Order Moment Syz
SM_v_TH | 50 |       |SM P    MR|(degC)^2        |Pot.Temp.   sub-grid variance
```

(continues on next page)

(continued from previous page)

```
ADVr_SLT| 50 |       |WM      LR|psu.m^3/s       |Vertical   Advective Flux of␣
↪Salinity
ADVx_SLT| 50 |   128 |UU      MR|psu.m^3/s       |Zonal      Advective Flux of␣
↪Salinity
ADVy_SLT| 50 |   127 |VV      MR|psu.m^3/s       |Meridional Advective Flux of␣
↪Salinity
DFrE_SLT| 50 |       |WM      LR|psu.m^3/s       |Vertical Diffusive Flux of Salinity␣
↪   (Explicit part)
DFxE_SLT| 50 |   131 |UU      MR|psu.m^3/s       |Zonal      Diffusive Flux of␣
↪Salinity
DFyE_SLT| 50 |   130 |VV      MR|psu.m^3/s       |Meridional Diffusive Flux of␣
↪Salinity
DFrI_SLT| 50 |       |WM      LR|psu.m^3/s       |Vertical Diffusive Flux of Salinity␣
↪   (Implicit part)
SALTFILL| 50 |       |SM      MR|psu.m^3/s       |Filling of Negative Values of␣
↪Salinity
SM_x_SLT| 50 |   135 |UM      MR|psu              |Salinity    1rst Order Moment Sx
SM_y_SLT| 50 |   134 |VM      MR|psu              |Salinity    1rst Order Moment Sy
SM_z_SLT| 50 |       |SM      MR|psu              |Salinity    1rst Order Moment Sz
SMxx_SLT| 50 |   138 |UM      MR|psu              |Salinity    2nd Order Moment Sxx
SMyy_SLT| 50 |   137 |VM      MR|psu              |Salinity    2nd Order Moment Syy
SMzz_SLT| 50 |       |SM      MR|psu              |Salinity    2nd Order Moment Szz
SMxy_SLT| 50 |       |SM      MR|psu              |Salinity    2nd Order Moment Sxy
SMxz_SLT| 50 |   142 |UM      MR|psu              |Salinity    2nd Order Moment Sxz
SMyz_SLT| 50 |   141 |VM      MR|psu              |Salinity    2nd Order Moment Syz
SM_v_SLT| 50 |       |SM P    MR|(psu)^2          |Salinity    sub-grid variance
VISCAHZ | 50 |       |SZ      MR|m^2/s            |Harmonic Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VISCA4Z | 50 |       |SZ      MR|m^4/s            |Biharmonic Visc Coefficient (m4/s)␣
↪(Zeta Pt)
VISCAHD | 50 |       |SM      MR|m^2/s            |Harmonic Viscosity Coefficient (m2/
↪s) (Div Pt)
VISCA4D | 50 |       |SM      MR|m^4/s            |Biharmonic Viscosity Coefficient␣
↪(m4/s) (Div Pt)
VISCAHW | 50 |       |WM      LR|m^2/s            |Harmonic Viscosity Coefficient (m2/
↪s) (W Pt)
VISCA4W | 50 |       |WM      LR|m^4/s            |Biharmonic Viscosity Coefficient␣
↪(m4/s) (W Pt)
VAHZMAX | 50 |       |SZ      MR|m^2/s            |CFL-MAX Harm Visc Coefficient (m2/
↪s) (Zeta Pt)
VA4ZMAX | 50 |       |SZ      MR|m^4/s            |CFL-MAX Biharm Visc Coefficient (m4/
↪s) (Zeta Pt)
VAHDMAX | 50 |       |SM      MR|m^2/s            |CFL-MAX Harm Visc Coefficient (m2/
↪s) (Div Pt)
VA4DMAX | 50 |       |SM      MR|m^4/s            |CFL-MAX Biharm Visc Coefficient (m4/
↪s) (Div Pt)
VAHZMIN | 50 |       |SZ      MR|m^2/s            |RE-MIN Harm Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VA4ZMIN | 50 |       |SZ      MR|m^4/s            |RE-MIN Biharm Visc Coefficient (m4/
↪s) (Zeta Pt)
VAHDMIN | 50 |       |SM      MR|m^2/s            |RE-MIN Harm Visc Coefficient (m2/s)␣
↪(Div Pt)
VA4DMIN | 50 |       |SM      MR|m^4/s            |RE-MIN Biharm Visc Coefficient (m4/
↪s) (Div Pt)
VAHZLTH | 50 |       |SZ      MR|m^2/s            |Leith Harm Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VA4ZLTH | 50 |       |SZ      MR|m^4/s            |Leith Biharm Visc Coefficient (m4/
↪s) (Zeta Pt)
```

(continues on next page)

(continued from previous page)

```
VAHDLTH | 50 |       |SM      MR|m^2/s          |Leith Harm Visc Coefficient (m2/s)␣
↪(Div Pt)
VA4DLTH | 50 |       |SM      MR|m^4/s          |Leith Biharm Visc Coefficient (m4/
↪s) (Div Pt)
VAHZLTHD| 50 |       |SZ      MR|m^2/s          |LeithD Harm Visc Coefficient (m2/s)␣
↪(Zeta Pt)
VA4ZLTHD| 50 |       |SZ      MR|m^4/s          |LeithD Biharm Visc Coefficient (m4/
↪s) (Zeta Pt)
VAHDLTHD| 50 |       |SM      MR|m^2/s          |LeithD Harm Visc Coefficient (m2/s)␣
↪(Div Pt)
VA4DLTHD| 50 |       |SM      MR|m^4/s          |LeithD Biharm Visc Coefficient (m4/
↪s) (Div Pt)
VAHZSMAG| 50 |       |SZ      MR|m^2/s          |Smagorinsky Harm Visc Coefficient␣
↪(m2/s) (Zeta Pt)
VA4ZSMAG| 50 |       |SZ      MR|m^4/s          |Smagorinsky Biharm Visc Coeff. (m4/
↪s) (Zeta Pt)
VAHDSMAG| 50 |       |SM      MR|m^2/s          |Smagorinsky Harm Visc Coefficient␣
↪(m2/s) (Div Pt)
VA4DSMAG| 50 |       |SM      MR|m^4/s          |Smagorinsky Biharm Visc Coeff. (m4/
↪s) (Div Pt)
momKE   | 50 |       |SMR     MR|m^2/s^2        |Kinetic Energy (in momentum Eq.)
momHDiv | 50 |       |SMR     MR|s^-1           |Horizontal Divergence (in momentum␣
↪Eq.)
momVort3| 50 |       |SZR     MR|s^-1           |3rd component (vertical) of␣
↪Vorticity
Strain  | 50 |       |SZR     MR|s^-1           |Horizontal Strain of Horizontal␣
↪Velocities
Tension | 50 |       |SMR     MR|s^-1           |Horizontal Tension of Horizontal␣
↪Velocities
UBotDrag| 50 |   176 |UUR     MR|m/s^2          |U momentum tendency from Bottom Drag
VBotDrag| 50 |   175 |VVR     MR|m/s^2          |V momentum tendency from Bottom Drag
USidDrag| 50 |   178 |UUR     MR|m/s^2          |U momentum tendency from Side Drag
VSidDrag| 50 |   177 |VVR     MR|m/s^2          |V momentum tendency from Side Drag
Um_Diss | 50 |   180 |UUR     MR|m/s^2          |U momentum tendency from Dissipation
Vm_Diss | 50 |   179 |VVR     MR|m/s^2          |V momentum tendency from Dissipation
Um_Advec| 50 |   182 |UUR     MR|m/s^2          |U momentum tendency from Advection␣
↪terms
Vm_Advec| 50 |   181 |VVR     MR|m/s^2          |V momentum tendency from Advection␣
↪terms
Um_Cori | 50 |   184 |UUR     MR|m/s^2          |U momentum tendency from Coriolis␣
↪term
Vm_Cori | 50 |   183 |VVR     MR|m/s^2          |V momentum tendency from Coriolis␣
↪term
Um_dPHdx| 50 |   186 |UUR     MR|m/s^2          |U momentum tendency from␣
↪Hydrostatic Pressure grad
Vm_dPHdy| 50 |   185 |VVR     MR|m/s^2          |V momentum tendency from␣
↪Hydrostatic Pressure grad
Um_Ext  | 50 |   188 |UUR     MR|m/s^2          |U momentum tendency from external␣
↪forcing
Vm_Ext  | 50 |   187 |VVR     MR|m/s^2          |V momentum tendency from external␣
↪forcing
Um_AdvZ3| 50 |   190 |UUR     MR|m/s^2          |U momentum tendency from Vorticity␣
↪Advection
Vm_AdvZ3| 50 |   189 |VVR     MR|m/s^2          |V momentum tendency from Vorticity␣
↪Advection
Um_AdvRe| 50 |   192 |UUR     MR|m/s^2          |U momentum tendency from vertical␣
↪Advection (Explicit part)
```

(continues on next page)

(continued from previous page)

```
Vm_AdvRe| 50 |   191 |VVR     MR|m/s^2          |V momentum tendency from vertical␣
↪Advection (Explicit part)
VISrI_Um| 50 |       |WU      LR|m^4/s^2        |Vertical   Viscous Flux of U␣
↪momentum (Implicit part)
VISrI_Vm| 50 |       |WV      LR|m^4/s^2        |Vertical   Viscous Flux of V␣
↪momentum (Implicit part)
```

9.1.4.5 MITgcm packages: available diagnostics lists

For a list of the diagnostic fields available in the different MITgcm packages, follow the link to the available diagnostics listing in the manual section describing the package:

- pkg/aim_v23: *available diagnostics*
- pkg/exf: *available diagnostics*
- pkg/gchem: *available diagnostics*
- pkg/generic_advdiff: *available diagnostics*
- pkg/gridalt: *available diagnostics*
- pkg/gmredi: *available diagnostics*
- pkg/fizhi: *available diagnostics*
- pkg/kpp: *available diagnostics*
- pkg/land: *available diagnostics*
- pkg/mom_common: *available diagnostics*
- pkg/obcs: *available diagnostics*
- pkg/thsice: *available diagnostics*
- pkg/seaice: *available diagnostics*
- pkg/shap_filt: *available diagnostics*
- pkg/ptracers: *available diagnostics*

9.2 Fortran Native I/O: pkg/mdsio and pkg/rw

9.2.1 pkg/mdsio

9.2.1.1 Introduction

pkg/mdsio contains a group of Fortran routines intended as a general interface for reading and writing direct-access ("binary") Fortran files. pkg/mdsio routines are used by pkg/rw.

9.2.1.2 Using pkg/mdsio

pkg/mdsio is geared toward the reading and writing of floating point (Fortran `REAL*4` or `REAL*8`) arrays. It assumes that the in-memory layout of all arrays follows the per-tile MITgcm convention

```
C     Example of a "2D" array
      _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)

C     Example of a "3D" array
      _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,1:Nr,nSx,nSy)
```

where the first two dimensions are spatial or "horizontal" indicies that include a "halo" or exchange region (please see Section 6 and Section 8.2.5 which describe domain decomposition), and the remaining indicies (`Nr`, `nSx`, and `nSx`) are often present but may or may not be necessary for a specific variable..

In order to write output, pkg/mdsio is called with a function such as:

```
CALL MDSWRITEFIELD(fn,prec,lgf,typ,Nr,arr,irec,myIter,myThid)
```

where:

`fn` is a `CHARACTER` string containing a file "base" name which will then be used to create file names that contain tile and/or model iteration indicies

`prec` is an integer that contains one of two globally defined values (`precFloat64` or `precFloat32`)

`lgf` is a `LOGICAL` that typically contains the globally defined `globalFile` option which specifies the creation of globally (spatially) concatenated files

`typ` is a `CHARACTER` string that specifies the type of the variable being written (`'RL'` or `'RS'`)

`Nr` is an integer that specifies the number of vertical levels within the variable being written

`arr` is the variable (array) to be written

`irec` is the starting record within the output file that will contain the array

`myIter,myThid` are integers containing, respectively, the current model iteration count and the unique thread ID for the current context of execution

As one can see from the above (generic) example, enough information is made available (through both the argument list and through common blocks) for pkg/mdsio to perform the following tasks:

1. open either a per-tile file such as:

   `uVel.0000302400.003.001.data`

   or a "global" file such as

   `uVel.0000302400.data`

2. byte-swap (as necessary) the input array and write its contents (minus any halo information) to the binary file – or to the correct location within the binary file if the `globalfile` option is used, and

3. create an ASCII–text metadata file (same name as the binary but with a `.meta` extension) describing the binary file contents (often, for later use with the MATLAB rdmds() utility).

Reading output with pkg/mdsio is very similar to writing it. A typical function call is

```
CALL MDSREADFIELD(fn,prec,typ,Nr,arr,irec,myThid)
```

where variables are exactly the same as the `MDSWRITEFIELD` example provided above. It is important to note that the `lgf` argument is missing from the `MDSREADFIELD` function. By default, pkg/mdsio will first try to read from an appropriately named global file and, failing that, will try to read from a per-tile file.

9.2.1.3 Important considerations

When using pkg/mdsio, one should be aware of the following package features and limitations:

- **Byte-swapping:** For the most part, byte-swapping is gracefully handled. All files intended for reading/writing by pkg/mdsio should contain big-endian (sometimes called "network byte order") data. By handling byte-swapping within the model, MITgcm output is more easily ported between different machines, architectures, compilers, etc. Byteswapping can be turned on/off at compile time within pkg/mdsio' using the `_BYTESWAPIO` CPP macro which is usually set within a genmake2 options file or `optfile` (see Section 3.5.2.2). Additionally, some compilers may have byte-swap options that are speedier or more convenient to use.
- **Data types:** Data types are currently limited to single– or double–precision floating point values. These values can be converted, on-the-fly, from one to the other so that any combination of either single– or double–precision variables can be read from or written to files containing either single– or double–precision data.
- **Array sizes:** Array sizes are limited; pkg/mdsio is very much geared towards the reading/writing of per-tile (that is, domain-decomposed and halo-ed) arrays. Data that cannot be made to "fit" within these assumed sizes can be challenging to read or write with pkg/mdsio.
- **Tiling:** Tiling or domain decomposition is automatically handled by pkg/mdsio for logically rectangular grid topologies (e.g., lat-lon grids) and "standard" cubed sphere topologies. More complicated topologies will probably not be supported. pkg/mdsio can, without any coding changes, read and write to/from files that were run on the same global grid but with different tiling (grid decomposition) schemes. For example, pkg/mdsio can use and/or create identical input/output files for a "C32" cube when the model is run with either 6, 12, or 24 tiles (corresponding to 1, 2 or 4 tiles per cubed sphere face). This is one of the primary advantages that the pkg/mdsio package has over pkg/mnc.
- **Single-CPU I/O:** This option can be specified with the flag `useSingleCpuIO = .TRUE.` in the `PARM01` namelist within the main `data` file. Single–CPU I/O mode is appropriate for computers (e.g., some SGI systems) where it can either speed overall I/O or solve problems where the operating system or file systems cannot correctly handle multiple threads or MPI processes simultaneously writing to the same file.
- **Meta-data:** Meta-data is written by MITgcm on a per-file basis using a second file with a `.meta` extension as described above. MITgcm itself does not read the `*.meta` files, they are there primarly for convenience during post-processing. One should be careful not to delete the meta-data files when using MATLAB post-processing scripts such as rdmds() since it relies upon them.
- **Numerous files:** If one is not careful (e.g., dumping many variables every time step over a long integration), pkg/mdsio will write copious amounts of files. The creation of both a binary (`*.data`) and ASCII text meta-data (`*.meta`) file for each output type step exacerbates the issue. Some operating systems do not gracefully handle large numbers (e.g., many thousands to millions) of files within one directory. So care should be taken to split output into smaller groups using subdirectories.
- **Overwriting output:** Overwriting of output is the **default behavior** for pkg/mdsio. If a model tries to write to a file name that already exists, the older file **will be deleted**. For this reason, MITgcm users should be careful to move output that they wish to keep into, for instance, subdirectories before performing subsequent runs that may over–lap in time or otherwise produce files with identical names (e.g., Monte-Carlo simulations).
- **No "halo" information:** "Halo" information is neither written nor read by pkg/mdsio. Along the horizontal dimensions, all variables are written in an `sNx`–by–`sNy` fashion. So, although variables (arrays) may be defined at different locations on Arakawa grids [U (right/left horizontal edges), V (top/bottom horizontal edges), M (mass or cell center), or Z (vorticity or cell corner) points], they are all written using only interior (`1:sNx` and `1:sNy`) values. For quantities defined at U, V, and M points, writing `1:sNx` and `1:sNy` for every tile is sufficient to ensure that all values are written globally for some grids (e.g., cubed sphere, re-entrant channels, and doubly-periodic rectangular regions). For Z points, failing to write values at the `sNx+1` and `sNy+1` locations means that, for some tile topologies, not all values are written. For instance, with a cubed sphere topology at least two corner values are "lost" (fail to be written for any tile) if the `sNx+1` and `sNy+1` values are ignored. If this is an issue, we recommend switching to pkg/mnc, which writes the `sNx+1` and `sNy+1` grid values for

the U, V, and Z locations. Also, pkg/mnc is capable of reading and/or writing entire halo regions and more complicated array shapes which can be helpful when debugging – features that do not exist within pkg/mdsio.

CPP Flag Name	Default	Description
SAFE_IO	#undef	if defined, stops the model from overwriting its own files
ALLOW_WHIO	#undef	I/O will include tile halos in the files

9.2.2 pkg/rw basic binary I/O utilities

pkg/rw provides a very rudimentary binary I/O capability for quickly writing *single record* direct-access Fortran binary files. It is primarily used for writing diagnostic output.

9.2.2.1 Introduction

pkg/rw is an interface to the more general pkg/mdsio package. pkg/rw can be used to write or read direct-access Fortran binary files for 2-D XY and 3-D XYZ arrays. The arrays are assumed to have been declared according to the standard MITgcm 2-D or 3-D floating point array type:

```
C     Example of declaring a standard two dimensional "long"
C     floating point type array (the _RL macro is usually
C     mapped to 64-bit floats in most configurations)
      _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
```

Each call to a pkg/rw read or write routine will read (or write) to the first record of a file. To write direct access Fortran files with multiple records use the higher-level routines in pkg/mdsio rather than pkg/rw routines. To write self-describing files that contain embedded information describing the variables being written and the spatial and temporal locations of those variables use the pkg/mnc instead (see Section 9.3) which produces netCDF format output.

CPP Flag Name	Default	Description
RW_SAFE_MFLDS	#define	use READ_MFLDS in "safe" mode (set/check/unset for each file to read); involves more thread synchronization which could slow down multi-threaded run
RW_DISABLE_SMALL_OVERLAP	#undef	disable writing of small-overlap size array (to reduce memory size since those S/R do a local copy to 3-D full-size overlap array)

9.3 NetCDF I/O: pkg/mnc

Package pkg/mnc is a set of convenience routines written to expedite the process of creating, appending, and reading netCDF files. NetCDF is an increasingly popular self-describing file format intended primarily for scientific data sets. An extensive collection of netCDF documentation, including a user's guide, tutorial, FAQ, support archive and other information can be obtained from UCAR's web site http://www.unidata.ucar.edu/software/netcdf.

Since it is a "wrapper" for netCDF, pkg/mnc depends upon the Fortran-77 interface included with the standard NetCDF v3.x library which is often called `libnetcdf.a`. Please contact your local systems administrators or email mitgcm-support@mitgcm.org for help building and installing netCDF for your particular platform.

Every effort has been made to allow pkg/mnc and pkg/mdsio (see Section 9.2) to peacefully co-exist. In may cases, the model can read one format and write to the other. This side-by-side functionality can be used to, for instance, help convert pickup files or other data sets between the two formats.

9.3.1 Using pkg/mnc

9.3.1.1 pkg/mnc configuration:

As with all MITgcm packages, pkg/mnc can be turned on or off at compile time using the `packages.conf` file or the genmake2 `-enable=mnc` or `-disable=mnc` switches.

While pkg/mnc is likely to work "as is", there are a few compile–time constants that may need to be increased for simulations that employ large numbers of tiles within each process. Note that the important quantity is the maximum number of tiles **per process**. Since MPI configurations tend to distribute large numbers of tiles over relatively large numbers of MPI processes, these constants will rarely need to be increased.

If pkg/mnc runs out of space within its "lookup" tables during a simulation, then it will provide an error message along with a recommendation of which parameter to increase. The parameters are all located within MNC_COMMON.h and the ones that may need to be increased are:

Name	Default	Description
MNC_MAX_ID	1000	IDs for various low-level entities
MNC_MAX_INFO	400	IDs (mostly for object sizes)
MNC_CW_MAX_I	150	IDs for the "wrapper" layer

In those rare cases where pkg/mnc "out-of-memory" error messages are encountered, it is a good idea to increase the too-small parameter by a factor of 2–10 in order to avoid wasting time on an iterative compile–test sequence.

9.3.1.2 pkg/mnc Inputs:

Like most MITgcm packages, all of pkg/mnc can be turned on/off at runtime using a single flag in `data.pkg`:

Name	Type	Default	Description
useMNC	L	.FALSE.	overall MNC ON/OFF switch

One important MNC–related flag is present in the main `data` namelist file in the `PARM03` section:

Name	Type	Default	Description
outputTypesInclusive	L	.FALSE.	use all available output "types"

which specifies that turning on pkg/mnc for a particular type of output should not simultaneously turn off the default output method as it normally does. Usually, this option is only used for debugging purposes since it is inefficient to write output types using both pkg/mnc and pkg/mdsio or ASCII output. This option can also be helpful when transitioning from pkg/mdsio to pkg/mnc since the output can be readily compared.

For run-time configuration, most of the pkg/mnc–related model parameters are contained within a Fortran namelist file called `data.mnc`. The available parameters currently include:

Name	Type	Default	Description
mnc_use_outdir	L	.FALSE.	create a directory for output
mnc_outdir_str	S	'mnc_'	output directory name
mnc_outdir_date	L	.FALSE.	embed date in the outdir name
mnc_outdir_num	L	.TRUE.	optional
pickup_write_mnc	L	.TRUE.	use MNC to write pickup files
pickup_read_mnc	L	.TRUE.	use MNC to read pickup file
mnc_use_indir	L	.FALSE.	use a directory (path) for input
mnc_indir_str	S	' '	input directory (or path) name
snapshot_mnc	L	.TRUE.	write snapshot output w/MNC
monitor_mnc	L	.TRUE.	write pkg/monitor output w/MNC
timeave_mnc	L	.TRUE.	write pkg/timeave' output w/MNC
autodiff_mnc	L	.TRUE.	write pkg/autodiff' output w/MNC
mnc_max_fsize	R	2.1e+09	max allowable file size (<2GB)
mnc_filefreq	R	-1	frequency of new file creation (seconds)
readgrid_mnc	L	.FALSE.	read grid quantities using MNC
mnc_echo_gvtypes	L	.FALSE.	list pre-defined "types" (debug)

Unlike the older pkg/mdsio method, pkg/mnc has the ability to create or use existing output directories. If either mnc_outdir_date or mnc_outdir_num is `.TRUE.`, then pkg/mnc will try to create directories on a *per process* basis for its output. This means that a single directory will be created for a non-MPI run and multiple directories (one per MPI process) will be created for an MPI run. This approach was chosen since it works safely on both shared global file systems (such as NFS and AFS) and on local (per-compute-node) file systems. And if both mnc_outdir_date and mnc_outdir_num are `.FALSE.`, then the pkg/mnc package will assume that the directory specified in mnc_outdir_str already exists and will use it. This allows the user to create and specify directories outside of the model.

For input, pkg/mnc can use a single global input directory. This is a just convenience that allows pkg/mnc to gather all of its input files from a path other than the current working directory. As with pkg/mdsio, the default is to use the current working directory.

The flags snapshot_mnc, monitor_mnc, timeave_mnc, and autodiff_mnc allow the user to turn on pkg/mnc for particular "types" of output. If a type is selected, then pkg/mnc will be used for all output that matches that type. This applies to output from the main model and from all of the optional MITgcm packages. Mostly, the names used here correspond to the names used for the output frequencies in the main `data` namelist file.

The mnc_max_fsize parameter is a convenience added to help users work around common file size limitations. On many computer systems, either the operating system, the file system(s), and/or the netCDF libraries are unable to handle files greater than two or four gigabytes in size. pkg/mnc is able to work within this limitation by creating new files which grow along the netCDF "unlimited" (usually, time) dimension. The default value for this parameter is just slightly less than 2GB which is safe on virtually all operating systems. Essentially, this feature is a way to intelligently and automatically split files output along the unlimited dimension. On systems that support large file sizes, these splits can be readily concatenated (that is, un-done) using tools such as the NetCDF Operators (with ncrcat) which is available at http://nco.sourceforge.net.

Another way users can force the splitting of pkg/mnc files along the time dimension is the mnc_filefreq option. With it, files that contain variables with a temporal dimension can be split at regular intervals based solely upon the model time (specified in seconds). For some problems, this can be much more convenient than splitting based upon file size.

Additional pkg/mnc–related parameters may be contained within each package. Please see the individual packages for descriptions of their use of pkg/mnc.

9.3.1.3 pkg/mnc output:

Depending upon the flags used, pkg/mnc will produce zero or more directories containing one or more netCDF files as output. These files are either mostly or entirely compliant with the netCDF "CF" convention (v1.0) and any conformance issues will be fixed over time. The patterns used for file names are:

- «BASENAME».«tileNum».nc
- «BASENAME».«nIter».«faceNum».nc
- «BASENAME».«nIter».«tileNum».nc

and examples are:

- grid.t001.nc, grid.t002.nc
- input.0000072000.f001.nc
- state.0000000000.t001.nc, surfDiag.0000036000.t001.nc

where «BASENAME» is the name selected to represent a set of variables written together, «nIter» is the current iteration number as specified in the main `data` namelist input file and written in a zero-filled 10-digit format, «tileNum» is a three-or-more-digit zero-filled and `t`–prefixed tile number, «faceNum» is a three-or-more-digit zero-filled and `f`–prefixed face number, and `.nc` is the file suffix specified by the current netCDF "CF" conventions.

Some example «BASENAME» values are:

grid contains the variables that describe the various grid constants related to locations, lengths, areas, etc.

state contains the variables output at the dumpFreq time frequency

pickup.ckptA, pickup.ckptB are the "rolling" checkpoint files

tave contains the time-averaged quantities from the main model

All pkg/mnc output is currently done in a "file-per-tile" fashion since most NetCDF v3.x implementations cannot write safely within MPI or multi-threaded environments. This tiling is done in a global fashion and the tile numbers are appended to the base names as described above. Some scripts to manipulate pkg/mnc output are available at utils/matlab which includes a spatial "assembly" script mnc_assembly.m.

More general manipulations can be performed on netCDF files with the NetCDF Operators ("NCO") at http://nco.sourceforge.net or with the Climate Data Operators ("CDO") at https://code.mpimet.mpg.de/projects/cdo.

Unlike the older pkg/mdsio routines, pkg/mnc reads and writes variables on different "grids" depending upon their location in the Arakawa C–grid. The following table provides examples:

Name	C–grid location	# in X	# in Y
Temperature	mass	sNx	sNy
Salinity	mass	sNx	sNy
U velocity	U	sNx+1	sNy
V velocity	V	sNx	sNy+1
Vorticity	vorticity	sNx+1	sNy+1

and the intent is two–fold:

1. For some grid topologies it is impossible to output all quantities using only `sNx,sNy` arrays for every tile. Two examples of this failure are the missing corners problem for vorticity values on the cubed sphere and the velocity edge values for some open–boundary domains.
2. Writing quantities located on velocity or vorticity points with the above scheme introduces a very small data redundancy. However, any slight inconvenience is easily offset by the ease with which one can, on every

individual tile, interpolate these values to mass points without having to perform an "exchange" (or "halo-filling") operation to collect the values from neighboring tiles. This makes the most common post–processing operations much easier to implement.

9.3.2 pkg/mnc Troubleshooting

9.3.2.1 Build troubleshooting:

In order to build MITgcm with pkg/mnc enabled, the NetCDF v3.x Fortran-77 (not Fortran-90) library must be available. This library is composed of a single header file (called `netcdf.inc`) and a single library file (usually called `libnetcdf.a`) and it must be built with the same compiler with compatible compiler options as the one used to build MITgcm (in other words, while one does not have to build `libnetcdf.a` with the same exact set of compiler options as MITgcm, one must avoid using some specific different compiler options which are incompatible, i.e., causing a compile-time or run-time error).

For more details concerning the netCDF build and install process, please visit the Getting and Building NetCDF guide which includes an extensive list of known–good netCDF configurations for various platforms.

9.3.2.2 Runtime troubleshooting:

Please be aware of the following:

- As a safety feature, the pkg/mnc does not, by default, allow pre-existing files to be appended to or overwritten. This is in contrast to the older pkg/mdsio which will, without any warning, overwrite existing files. If MITgcm aborts with an error message about the inability to open or write to a netCDF file, please check **first** whether you are attempting to overwrite files from a previous run.
- The constraints placed upon the "unlimited" (or "record") dimension inherent with NetCDF v3.x make it very inefficient to put variables written at potentially different intervals within the same file. For this reason, pkg/mnc output is split into groups of files which attempt to reflect the nature of their content.
- On many systems, netCDF has practical file size limits on the order of 2–4GB (the maximium memory addressable with 32bit pointers or pointer differences) due to a lack of operating system, compiler, and/or library support. The latest revisions of NetCDF v3.x have large file support and, on some operating systems, file sizes are only limited by available disk space.
- There is an 80 character limit to the total length of all file names. This limit includes the directory (or path) since paths and file names are internally appended. Generally, file names will not exceed the limit and paths can usually be shortened using, for example, soft links.
- pkg/mnc does not (yet) provide a mechanism for reading information from a single "global" file as can be done with pkg/mdsio. This is in progress.

9.3.3 pkg/mnc Internals

pkg/mnc is a two-level convenience library (or "wrapper") for most of the netCDF Fortran API. Its purpose is to streamline the user interface to netCDF by maintaining internal relations (look-up tables) keyed with strings (or names) and entities such as netCDF files, variables, and attributes.

The two levels of pkg/mnc are:

Upper level

The upper level contains information about two kinds of associations:

grid type is lookup table indexed with a grid type name. Each grid type name is associated with a number of dimensions, the dimension sizes (one of which may be unlimited), and starting and ending index arrays. The intent is to store all the necessary size and shape information for the Fortran arrays containing MITgcm–style "tile" variables (i.e., a central region surrounded by a variably-sized "halo" or exchange region as shown in Figure 6.5 and Figure 6.6).

variable type is a lookup table indexed by a variable type name. For each name, the table contains a reference to a grid type for the variable and the names and values of various attributes.

Within the upper level, these associations are not permanently tied to any particular netCDF file. This allows the information to be re-used over multiple file reads and writes.

Lower level

In the lower (or internal) level, associations are stored for netCDF files and many of the entities that they contain including dimensions, variables, and global attributes. All associations are on a per-file basis. Thus, each entity is tied to a unique netCDF file and will be created or destroyed when files are, respectively, opened or closed.

9.3.3.1 pkg/mnc grid–tTypes and variable–types:

As a convenience for users, pkg/mnc includes numerous routines to aid in the writing of data to netCDF format. Probably the biggest convenience is the use of pre-defined "grid types" and "variable types". These "types" are simply look-up tables that store dimensions, indicies, attributes, and other information that can all be retrieved using a single character string.

The "grid types" are a way of mapping variables within MITgcm to netCDF arrays. Within MITgcm, most spatial variables are defined using 2–D or 3–D arrays with "overlap" regions (see Figure 6.5, a possible vertical index, and Figure 6.6) and tile indicies such as the following "U" velocity:

```
_RL  uVel (1-OLx:sNx+OLx,1-OLy:sNy+OLy,Nr,nSx,nSy)
```

as defined in DYNVARS.h.

The grid type is a character string that encodes the presence and types associated with the four possible dimensions. The character string follows the format:

```
«H0»_«H1»_«H2»__«V»__«T»
```

(note the double underscore between «H2» and «V», and «V» and «T») where the terms «H0», «H1», «H2», «V», «T» can be almost any combination of the following:

Horizontal			Vertical	Time
H0: location	H1: dimensions	H2: halo	V: location	T: level
–	xy	Hn	–	–
U	x	Hy	i	t
V	y		c	
Cen				
Cor				

A example list of all pre-defined combinations is contained in the file pkg/mnc/pre-defined_grids.txt.

The variable type is an association between a variable type name and the following items:

Item	Purpose
grid type	defines the in-memory arrangement
bi,bj dimensions	tiling indices, if present

and is used by the `mnc_cw__[R|W]` subroutines for reading and writing variables.

9.3.3.2 Using pkg/mnc: examples

Writing variables to netCDF files can be accomplished in as few as two function calls. The first function call defines a variable type, associates it with a name (character string), and provides additional information about the indicies for the tile (bi,bj) dimensions. The second function call will write the data at, if necessary, the current time level within the model.

Examples of the initialization calls can be found in the file model/src/ini_model_io.F where these function calls:

```
C     Create MNC definitions for DYNVARS.h variables
      CALL MNC_CW_ADD_VNAME('iter', '-_-_--__-__t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('iter',1,
     &     'long_name','iteration_count', myThid)

      CALL MNC_CW_ADD_VNAME('model_time', '-_-_--__-__t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,
     &     'long_name','Model Time', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,'units','s', myThid)

      CALL MNC_CW_ADD_VNAME('U', 'U_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,'units','m/s', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,
     &     'coordinates','XU YU RC iter', myThid)

      CALL MNC_CW_ADD_VNAME('T', 'Cen_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'units','degC', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'long_name',
     &     'potential_temperature', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,
     &     'coordinates','XC YC RC iter', myThid)
```

initialize four `VNAME`s and add one or more netCDF attributes to each.

The four variables defined above are subsequently written at specific time steps within model/src/write_state.F using the function calls:

```
C       Write dynvars using the MNC package
        CALL MNC_CW_SET_UDIM('state', -1, myThid)
        CALL MNC_CW_I_W('I','state',0,0,'iter', myIter, myThid)
        CALL MNC_CW_SET_UDIM('state', 0, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'model_time',myTime, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'U', uVel, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'T', theta, myThid)
```

While it is easiest to write variables within typical 2-D and 3-D fields where all data is known at a given time, it is also possible to write fields where only a portion (e.g., a "slab" or "slice") is known at a given instant. An example is provided within pkg/mom_vecinv/mom_vecinv.F where an offset vector is used:

```
    IF (useMNC .AND. snapshot_mnc) THEN
      CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fV', uCf,
&          offsets, myThid)
      CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fU', vCf,
&          offsets, myThid)
    ENDIF
```

to write a 3-D field one depth slice at a time.

Each element in the offset vector corresponds (in order) to the dimensions of the "full" (or virtual) array and specifies which are known at the time of the call. A zero within the offset array means that all values along that dimension are available while a positive integer means that only values along that index of the dimension are available. In all cases, the matrix passed is assumed to start (that is, have an in-memory structure) coinciding with the start of the specified slice. Thus, using this offset array mechanism, a slice can be written along any single dimension or combinations of dimensions.

9.4 Monitor: Simulation State Monitoring Toolkit

9.4.1 Introduction

pkg/monitor is primarily intended as a convenient method for calculating and writing the following statistics:

- minimum
- maximum
- mean
- standard deviation

for spatially distributed fields. By default, pkg/monitor output is sent to the "standard output" channel where it appears as ASCII text containing a `%MON` string such as this example:

```
(PID.TID 0000.0001) %MON time_tsnumber      =                     3
(PID.TID 0000.0001) %MON time_secondsf      =   3.6000000000000E+03
(PID.TID 0000.0001) %MON dynstat_eta_max    =   1.0025466645951E-03
(PID.TID 0000.0001) %MON dynstat_eta_min    =  -1.0008899950901E-03
(PID.TID 0000.0001) %MON dynstat_eta_mean   =   2.1037438449350E-14
(PID.TID 0000.0001) %MON dynstat_eta_sd     =   5.0985228723396E-04
(PID.TID 0000.0001) %MON dynstat_eta_del2   =   3.5216706549525E-07
(PID.TID 0000.0001) %MON dynstat_uvel_max   =   3.7594045977254E-05
(PID.TID 0000.0001) %MON dynstat_uvel_min   =  -2.8264287531564E-05
(PID.TID 0000.0001) %MON dynstat_uvel_mean  =   9.1369201945671E-06
(PID.TID 0000.0001) %MON dynstat_uvel_sd    =   1.6868439193567E-05
(PID.TID 0000.0001) %MON dynstat_uvel_del2  =   8.4315445301916E-08
```

pkg/monitor text can be readily parsed by the `testreport` script to determine, somewhat crudely but quickly, how similar the output from two experiments are when run on different platforms or before/after code changes.

pkg/monitor output can also be useful for quickly diagnosing practical problems such as CFL limitations, model progress (through iteration counts), and behavior within some packages that use it.

9.4.2 Using pkg/monitor

As with most packages, pkg/monitor can be turned on or off at compile and/or run times using the `packages.conf` and `data.pkg` files.

The monitor output can be sent to the standard output channel, to an pkg/mnc–generated file, or to both simultaneously. For pkg/mnc output, the flag `monitor_mnc=.TRUE.` should be set within the `data.mnc` file. For output to both ASCII and pkg/mnc, the flag `outputTypesInclusive=.TRUE.` should be set within the `PARM03` section of the main `data` file. It should be noted that the `outputTypesInclusive` flag will make **ALL** kinds of output (that is, everything written by pkg/mdsio, pkg/mnc, and pkg/monitor) simultaneously active so it should be used only with caution -– and perhaps only for debugging purposes.

CPP Flag Name	Default	Description
MONITOR_TEST_HFACZ	#undef	disable use of hFacZ

9.5 Grid Generation

The horizontal discretizations within MITgcm have been written to work with many different grid types including:

- cartesian coordinates
- spherical polar ("latitude-longitude") coordinates
- general curvilinear orthogonal coordinates

The last of these, especially when combined with the domain decomposition capabilities of MITgcm, allows a great degree of grid flexibility. To date, general curvilinear orthogonal coordinates have been used extensively in conjunction with so-called "cubed sphere" grids. However, it is important to observe that cubed sphere arrangements are only one example of what is possible with domain-decomposed logically rectangular regions each containing curvilinear orthogonal coordinate systems. Much more sophisticated domains can be imagined and constructed.

In order to explore the possibilities of domain-decomposed curvilinear orthogonal coordinate systems, a suite of grid generation software called "SPGrid" (for SPherical Gridding) has been developed. SPGrid is a relatively new facility and papers detailing its algorithms are in preparation. Although SPGrid is new and rapidly developing, it has already demonstrated the ability to generate some useful and interesting grids.

This section provides a very brief introduction to SPGrid and shows some early results. For further information, please contact the MITgcm support list MITgcm-support@mitgcm.org.

9.5.1 Using SPGrid

The SPGrid software is not a single program. Rather, it is a collection of C++ code and MATLAB scripts that can be used as a framework or library for grid generation and manipulation. Currently, grid creation is accomplished by either directly running MATLAB scripts or by writing a C++ "driver" program. The MATLAB scripts are suitable for grids composed of a single "face" (that is, a single logically rectangular region on the surface of a sphere). The C++ driver programs are appropriate for grids composed of multiple connected logically rectangular patches. Each driver program is written to specify the shape and connectivity of tiles and the preferred grid density (that is, the number of grid cells in each logical direction) and edge locations of the cells where they meet the edges of each face. The driver programs pass this information to the SPGrid library, which generates the actual grid and produces the output files that describe it.

Currently, driver programs are available for a few examples including cubes, "lat-lon caps" (cube topologies that have conformal caps at the poles and are exactly lat-lon channels for the remainder of the domain), and some simple "embedded" regions that are meant to be used within typical cubes or traditional lat-lon grids.

To create new grids, one may start with an existing driver program and modify it to describe a domain that has a different arrangement. The number, location, size, and connectivity of grid "faces" (the name used for the logically rectangular regions) can be readily changed. Further, the number of grid cells within faces and the location of the grid cells at the face edges can also be specified.

9.5.1.1 SPGrid requirements

The following programs and libraries are required to build and/or run the SPGrid suite:

- MATLAB is a run-time requirement since many of the generation algorithms have been written as MATLAB scripts.

- The Geometric Tools Engine (a C++ library) is needed for the main "driver" code.
- The netCDF library is needed for file I/O.
- The Boost serialization library is also used for I/O:
- a typical Linux/Unix build environment including the make utility (preferably GNU Make) and a C++ compiler (SPGrid was developed with g++ v4.x).

9.5.1.2 Obtaining SPGrid

The latest version can be obtained from:

9.5.1.3 Building SPGrid

The procedure for building is similar to many open source projects:

```
tar -xf spgrid-0.9.4.tar.gz
cd spgrid-0.9.4
export CPPFLAGS="-I/usr/include/netcdf-3"
export LDFLAGS="-L/usr/lib/netcdf-3"
./configure
make
```

where the `CPPFLAGS` and `LDFLAGS` environment variables can be edited to reflect the locations of all the necessary dependencies. SPGrid is known to work on Fedora Core Linux (versions 4 and 5) and is likely to work on most any Linux distribution that provides the needed dependencies.

9.5.1.4 Running SPGrid

Within the `src` sub-directory, various example driver programs exist. These examples describe small, simple domains and can generate the input files (formatted as either binary `*.mitgrid` or netCDF) used by MITgcm.

One such example is called `SpF_test_cube_cap` and it can be run with the following sequence of commands:

```
cd spgrid-0.9.4/src
make SpF_test_cube_cap
mkdir SpF_test_cube_cap.d
( cd SpF_test_cube_cap.d && ln -s ../../scripts/*.m . )
./SpF_test_cube_cap
```

which should create a series of output files:

```
SpF_test_cube_cap.d/grid_*.mitgrid
SpF_test_cube_cap.d/grid_*.nc
SpF_test_cube_cap.d/std_topology.nc
```

where the `grid_.mitgrid` and `grid_.nc` files contain the grid information in binary and netCDF formats and the `std_topology.nc` file contains the information describing the connectivity (both edge–edge and corner–corner contacts) between all the faces.

9.5.2 Example Grids

The following grids are various examples created with SPGrid.

9.6 Pre– and Post–Processing Scripts and Utilities

There are numerous tools for pre-processing data, converting model output and analysis written in MATLAB, Fortran (f77 and f90) and perl. As yet they remain undocumented although many are self-documenting (MATLAB routines have "help" written into them).

Here we'll summarize what is available but this is an ever growing resource so this may not cover everything that is out there:

9.6.1 Utilities Supplied With the Model

We supply some basic scripts with the model to facilitate conversion or reading of data into analysis software.

9.6.1.1 utils/scripts

In the directory utils/scripts, joinds and joinmds are perl scripts used to joining multi-part files created by MITgcm. Use joinmds. You will only need joinds if you are working with output older than two years (prior to c23).

9.6.1.2 utils/matlab

In the directory utils/matlab you will find several MATLAB scripts (`.m` files). The principle script is rdmds.m, used for reading the multi-part model output files into MATLAB . Place the scripts in your MATLAB path or change the path appropriately, then at the MATLAB prompt type:

```
>> help rdmds
```

to get help on how to use rdmds.

Another useful script scans the terminal output file for pkg/monitor information.

Most other scripts are for working in the curvilinear coordinate systems, and as yet are unpublished and undocumented.

9.6.1.3 pkg/mnc utils

The following scripts and utilities have been written to help manipulate netCDF files:

Tile Assembly: A MATLAB script mnc_assembly.m is available for spatially "assembling" pkg/mnc output. A convenience wrapper script called gluemnc.m is also provided. Please use the MATLAB help facility for more information.

gmt: As MITgcm evolves to handle more complicated domains and topologies, a suite of matlab tools is being written to more gracefully handle the model files. This suite is called "gmt" which refers to "generalized model topology" pre-/post-processing. Currently, this directory contains a MATLAB script gmt/rdnctiles.m that is able to read netCDF files for any domain. Additional scripts are being created that will work with these fields on a per-tile basis.

9.6.2 Pre-Processing Software

There is a suite of pre-processing software for interpolating bathymetry and forcing data, written by Adcroft and Biastoch. At some point, these will be made available for download. If you are in need of such software, contact one of them.

9.7 Potential Vorticity Matlab Toolbox

Author: Guillaume Maze

9.7.1 Introduction

This section of the documentation describes a MATLAB package that aims to provide useful routines to compute vorticity fields (relative, potential and planetary) and its related components. This is an offline computation. It was developed to be used in mode water studies, so that it comes with other related routines, in particular ones computing surface vertical potential vorticity fluxes.

9.7.2 Equations

9.7.2.1 Potential vorticity

The package computes the three components of the relative vorticity defined by:

$$\omega = \nabla \times \mathbf{U} = \begin{pmatrix} \omega_x \\ \omega_y \\ \zeta \end{pmatrix} \simeq \begin{pmatrix} -\frac{\partial v}{\partial z} \\ -\frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \end{pmatrix} \tag{9.1}$$

where we omitted the vertical velocity component (as done throughout the package).

The package then computes the potential vorticity as:

$$\begin{aligned} Q &= -\frac{1}{\rho}\omega \cdot \nabla\sigma_\theta \\ &= -\frac{1}{\rho}\left(\omega_x \frac{\partial \sigma_\theta}{\partial x} + \omega_y \frac{\partial \sigma_\theta}{\partial y} + (f+\zeta)\frac{\partial \sigma_\theta}{\partial z}\right) \end{aligned} \tag{9.2}$$

where ρ is the density, σ_θ is the potential density (both eventually computed by the package) and f is the Coriolis parameter.

The package is also able to compute the simpler planetary vorticity as:

$$Q_{spl} = -\frac{f}{\rho}\frac{\sigma_\theta}{\partial z} \tag{9.3}$$

9.7.2.2 Surface vertical potential vorticity fluxes

These quantities are useful in mode water studies because of the impermeability theorem which states that for a given potential density layer (embedding a mode water), the integrated PV only changes through surface input/output.

Vertical PV fluxes due to frictional and diabatic processes are given by:

$$J_z^B = -\frac{f}{h}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right) \tag{9.4}$$

$$J_z^F = \frac{1}{\rho \delta_e}\vec{k} \times \tau \cdot \nabla\sigma_m \tag{9.5}$$

These components can be computed with the package. Details on the variables definition and the way these fluxes are derived can be found in Section 9.7.5.

We now give some simple explanations about these fluxes and how they can reduce the PV value of an oceanic potential density layer.

Diabatic process

Let's take the PV flux due to surface buoyancy forcing from (9.4) and simplify it as:

$$J_z^B \simeq -\frac{\alpha f}{hC_w} Q_{net}$$

When the net surface heat flux Q_{net} is upward, i.e., negative and cooling the ocean (buoyancy loss), surface density will increase, triggering mixing which reduces the stratification and then the PV.

$$\begin{aligned} Q_{net} < 0 \qquad & \text{(upward, cooling)} \\ J_z^B > 0 \qquad & \text{(upward)} \\ -\rho^{-1}\nabla \cdot J_z^B < 0 \qquad & \text{(PV flux divergence)} \\ PV \searrow \qquad & \text{where } Q_{net} < 0 \end{aligned}$$

Frictional process: "Down-front" wind-stress

Now let's take the PV flux due to the "wind-driven buoyancy flux" from (9.5) and simplify it as:

$$\begin{aligned} J_z^F &= \frac{1}{\rho\delta_e}\left(\tau_x \frac{\partial \sigma}{\partial y} - \tau_y \frac{\partial \sigma}{\partial x}\right) \\ &\simeq \frac{1}{\rho\delta_e}\tau_x \frac{\partial \sigma}{\partial y} \end{aligned}$$

When the wind is blowing from the east above the Gulf Stream (a region of high meridional density gradient), it induces an advection of dense water from the northern side of the GS to the southern side through Ekman currents. Then, it induces a "wind-driven" buoyancy lost and mixing which reduces the stratification and the PV.

$$\begin{aligned} \vec{k} \times \tau \cdot \nabla\sigma > 0 \qquad & \text{("Down-front" wind)} \\ J_z^F > 0 \qquad & \text{(upward)} \\ -\rho^{-1}\nabla \cdot J_z^F < 0 \qquad & \text{(PV flux divergence)} \\ PV \searrow \qquad & \text{where } \vec{k} \times \tau \cdot \nabla\sigma > 0 \end{aligned}$$

Diabatic versus frictional processes

A recent debate in the community arose about the relative role of these processes. Taking the ratio of (9.4) and (9.5) leads to:

$$\begin{aligned} \frac{J_z^F}{J_Z^B} &= \frac{\frac{1}{\rho\delta_e}\vec{k} \times \tau \cdot \nabla\sigma}{-\frac{f}{h}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right)} \\ &\simeq \frac{Q_{Ek}/\delta_e}{Q_{net}/h} \end{aligned}$$

where appears the lateral heat flux induced by Ekman currents:

$$\begin{aligned} Q_{Ek} &= -\frac{C_w}{\alpha\rho f}\vec{k} \times \tau \cdot \nabla\sigma \\ &= \frac{C_w}{\alpha}\delta_e u_{\vec{Ek}} \cdot \nabla\sigma \end{aligned}$$

which can be computed with the package. In the aim of comparing both processes, it will be useful to plot surface net and lateral Ekman-induced heat fluxes together with PV fluxes.

9.7.3 Key routines

- **A_compute_potential_density.m**: Compute the potential density field. Requires the potential temperature and salinity (either total or anomalous) and produces one output file with the potential density field (file prefix is `SIGMATHETA`). The routine uses utils/matlab/densjmd95.m, a Matlab counterpart of the MITgcm built-in function to compute the density.
- **B_compute_relative_vorticity.m**: Compute the three components of the relative vorticity defined in (9.1). Requires the two horizontal velocity components and produces three output files with the three components (files prefix are `OMEGAX`, `OMEGAY` and `ZETA`).
- **C_compute_potential_vorticity.m**: Compute the potential vorticity without the negative ratio by the density. Two options are possible in order to compute either the full component (term into parenthesis in (9.2) or the planetary component ($f\partial_z\sigma_\theta$ in (9.3)). Requires the relative vorticity components and the potential density, and produces one output file with the potential vorticity (file prefix is `PV` for the full term and `splPV` for the planetary component).
- **D_compute_potential_vorticity.m**: Load the field computed with and divide it by $-\rho$ to obtain the correct potential vorticity. Require the density field and after loading, overwrite the file with prefix `PV` or `splPV`.
- **compute_density.m**: Compute the density ρ from the potential temperature and the salinity fields.
- **compute_JFz.m**: Compute the surface vertical PV flux due to frictional processes. Requires the wind stress components, density, potential density and Ekman layer depth (all of them, except the wind stress, may be computed with the package), and produces one output file with the PV flux J_z^F (see (9.5) and with `JFz` as a prefix.
- **compute_JBz.m**: Compute the surface vertical PV flux due to diabatic processes as:

$$J_z^B = -\frac{f}{h}\frac{\alpha Q_{net}}{C_w}$$

  which is a simplified version of the full expression given in (9.4). Requires the net surface heat flux and the mixed layer depth (of which an estimation can be computed with the package), and produces one output file with the PV flux J_z^B and with JBz as a prefix.
- **compute_QEk.m**: Compute the horizontal heat flux due to Ekman currents from the PV flux induced by frictional forces as:

$$Q_{Ek} = -\frac{C_w\delta_e}{\alpha f}J_z^F$$

  Requires the PV flux due to frictional forces and the Ekman layer depth, and produces one output with the heat flux and with QEk as a prefix.
- **eg_main_getPV**: A complete example of how to set up a master routine able to compute everything from the package.

9.7.4 Technical details

9.7.4.1 File name

A file name is formed by three parameters which need to be set up as global variables in MATLAB before running any routines. They are:

- the prefix, i.e., the variable name (`netcdf_UVEL` for example). This parameter is specified in the help section of all diagnostic routines.
- `netcdf_domain`: the geographical domain.

- `netcdf_suff`: the netcdf extension (nc or cdf for example).

Then, for example, if the calling MATLAB routine had set up:

```
global netcdf_THETA netcdf_SALTanom netcdf_domain netcdf_suff
netcdf_THETA    = 'THETA';
netcdf_SALTanom = 'SALT';
netcdf_domain   = 'north_atlantic';
netcdf_suff     = 'nc';
```

the routine A_compute_potential_density.m to compute the potential density field, will look for the files:

```
THETA.north_atlantic.nc
SALT.north_atlantic.nc
```

and the output file will automatically be: `SIGMATHETA.north_atlantic.nc`.

Otherwise indicated, output file prefix cannot be changed.

9.7.4.2 Path to file

All diagnostic routines look for input files in a subdirectory (relative to the MATLAB routine directory) called `./netcdf-files`, which in turn is supposed to contain subdirectories for each set of fields. For example, computing the potential density for the timestep 12H00 02/03/2005 will require a subdirectory with the potential temperature and salinity files like:

```
./netcdf-files/200501031200/THETA.north_atlantic.nc
./netcdf-files/200501031200/SALT.north_atlantic.nc
```

The output file `SIGMATHETA.north\_atlantic.nc` will be created in `./netcdf-files/200501031200/`. All diagnostic routines take as argument the name of the timestep subdirectory into `./netcdf-files`.

9.7.4.3 Grids

With MITgcm numerical outputs, velocity and tracer fields may not be defined on the same grid. Usually, `UVEL` and `VVEL` are defined on a C-grid but when interpolated from a cube-sphere simulation they are defined on a A-grid. When it is needed, routines allow to set up a global variable which define the grid to use.

9.7.5 Notes on the flux form of the PV equation and vertical PV fluxes

9.7.5.1 Flux form of the PV equation

The conservative flux form of the potential vorticity equation is:

$$\frac{\partial \rho Q}{\partial t} + \nabla \cdot \vec{J} = \quad 0 \tag{9.6}$$

where the potential vorticity Q is given by (9.2).

The generalized flux vector of potential vorticity is:

$$\vec{J} = \quad \rho Q \vec{u} + \vec{N_Q}$$

which allows to rewrite (9.6) as:

$$\frac{DQ}{dt} = -\frac{1}{\rho}\nabla \cdot \vec{N_Q} \tag{9.7}$$

where the non-advective PV flux $\vec{N_Q}$ is given by:

$$\vec{N_Q} = -\frac{\rho_0}{g} B\vec{\omega_a} + \vec{F} \times \nabla\sigma_\theta \tag{9.8}$$

Its first component is linked to the buoyancy forcing:

$$B = -\frac{g}{\rho_o}\frac{D\sigma_\theta}{dt}$$

and the second one to the non-conservative body forces per unit mass:

$$\vec{F} = \frac{D\vec{u}}{dt} + 2\Omega \times \vec{u} + \nabla p$$

Note that introducing B into (9.8) yields:

$$\vec{N_Q} = \omega_a \frac{D\sigma_\theta}{dt} + \vec{F} \times \nabla\sigma_\theta$$

9.7.5.2 Determining the PV flux at the ocean's surface

In the context of mode water study, we are particularly interested in how the PV may be reduced by surface PV fluxes because a mode water is characterized by a low PV value. Considering the volume limited by two $iso - \sigma_\theta$, PV flux is limited to surface processes and then vertical component of $\vec{N_Q}$. It is supposed that B and $\vec{F}$ will only be non-zero in the mixed layer (of depth h and variable density σ_m) exposed to mechanical forcing by the wind and buoyancy fluxes through the ocean's surface.

Given the assumption of a mechanical forcing confined to a thin surface Ekman layer (of depth δ_e, eventually computed by the package) and of hydrostatic and geostrophic balances, we can write:

$$\begin{aligned} \vec{u_g} &= \frac{1}{\rho f}\vec{k} \times \nabla p \\ \frac{\partial p_m}{\partial z} &= -\sigma_m g \\ \frac{\partial \sigma_m}{\partial t} + \vec{u}_m \cdot \nabla\sigma_m &= -\frac{\rho_0}{g} B \end{aligned} \tag{9.9}$$

where:

$$\vec{u}_m = \vec{u}_g + \vec{u}_{Ek} + o(R_o) \tag{9.10}$$

is the full velocity field composed of the geostrophic current $\vec{u}_g$ and the Ekman drift:

$$\vec{u}_{Ek} = -\frac{1}{\rho f}\vec{k} \times \frac{\partial \tau}{\partial z} \tag{9.11}$$

(where τ is the wind stress) and last by other ageostrophic components of $o(R_o)$ which are neglected.

Partitioning the buoyancy forcing as:

$$B = B_g + B_{Ek} \tag{9.12}$$

and using (9.10) and (9.11), (9.9) becomes:

$$\frac{\partial \sigma_m}{\partial t} + \vec{u}_g \cdot \nabla \sigma_m = \quad -\frac{\rho_0}{g} B_g$$

revealing the "wind-driven buoyancy forcing":

$$B_{Ek} = \quad \frac{g}{\rho_0} \frac{1}{\rho f} \left(\vec{k} \times \frac{\partial \tau}{\partial z} \right) \cdot \nabla \sigma_m$$

Note that since:

$$\frac{\partial B_g}{\partial z} = \quad \frac{\partial}{\partial z} \left(-\frac{g}{\rho_0} \vec{u_g} \cdot \nabla \sigma_m \right) = -\frac{g}{\rho_0} \frac{\partial \vec{u_g}}{\partial z} \cdot \nabla \sigma_m = 0$$

B_g must be uniform throughout the depth of the mixed layer and then being related to the surface buoyancy flux by integrating (9.12) through the mixed layer:

$$\int_{-h}^{0} B \, dz = \, hB_g + \int_{-h}^{0} B_{Ek} \, dz \, = \quad \mathcal{B}_{in} \tag{9.13}$$

where $\mathcal{B}_{in}$ is the vertically integrated surface buoyancy (in)flux:

$$\mathcal{B}_{in} = \quad \frac{g}{\rho_o} \left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net} \right) \tag{9.14}$$

with $\alpha \simeq 2.5 \times 10^{-4} \, \mathrm{K}^{-1}$ the thermal expansion coefficient (computed by the package otherwise), $C_w = 4187 \, \mathrm{J \, kg^{-1} K^{-1}}$ the specific heat of seawater, Q_{net} [W/m^{-2}] the net heat surface flux (positive downward, warming the ocean), β [psu^{-1}] the saline contraction coefficient, and $S_{net} = S * (E - P)$ [psu m s^{-1}] the net freshwater surface flux with S [psu] the surface salinity and $(E - P)$ [m/s] the fresh water flux.

Introducing the body force in the Ekman layer:

$$F_z = \quad \frac{1}{\rho} \frac{\partial \tau}{\partial z}$$

the vertical component of (9.8) is:

$$\begin{aligned}
\vec{N_{Q}}_z &= -\frac{\rho_0}{g}(B_g + B_{Ek})\omega_z + \frac{1}{\rho}\left(\frac{\partial \tau}{\partial z} \times \nabla \sigma_\theta\right) \cdot \vec{k} \\
&= -\frac{\rho_0}{g} B_g \omega_z - \frac{\rho_0}{g}\left(\frac{g}{\rho_0}\frac{1}{\rho f}\vec{k} \times \frac{\partial \tau}{\partial z} \cdot \nabla \sigma_m\right)\omega_z + \frac{1}{\rho}\left(\frac{\partial \tau}{\partial z} \times \nabla \sigma_\theta\right) \cdot \vec{k} \\
&= -\frac{\rho_0}{g} B_g \omega_z + \left(1 - \frac{\omega_z}{f}\right)\left(\frac{1}{\rho}\frac{\partial \tau}{\partial z} \times \nabla \sigma_\theta\right) \cdot \vec{k}
\end{aligned}$$

and given the assumption that $\omega_z \simeq f$, the second term vanishes and we obtain:

$$\vec{N_{Q}}_z = \quad -\frac{\rho_0}{g} f B_g \tag{9.15}$$

Note that the wind-stress forcing does not appear explicitly here but is implicit in B_g through (9.13): the buoyancy forcing B_g is determined by the difference between the integrated surface buoyancy flux $\mathcal{B}_{in}$ and the integrated "wind-driven buoyancy forcing":

$$\begin{aligned}
B_g &= \frac{1}{h}\left(\mathcal{B}_{in} - \int_{-h}^{0} B_{Ek} dz\right) \\
&= \frac{1}{h}\frac{g}{\rho_0}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right) - \frac{1}{h}\int_{-h}^{0} \frac{g}{\rho_0}\frac{1}{\rho f}\vec{k} \times \frac{\partial \tau}{\partial z} \cdot \nabla \sigma_m dz \\
&= \frac{1}{h}\frac{g}{\rho_0}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right) - \frac{g}{\rho_0}\frac{1}{\rho f \delta_e}\vec{k} \times \tau \cdot \nabla \sigma_m
\end{aligned}$$

Finally, from (9.8), the vertical surface flux of PV may be written as:

$$\vec{N_{Q}}_z = J_z^B + J_z^F$$
$$J_z^B = -\frac{f}{h}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right)$$
$$J_z^F = \frac{1}{\rho \delta_e}\vec{k} \times \tau \cdot \nabla \sigma_m$$

9.8 pkg/flt – Simulation of float / parcel displacements

9.8.1 Introduction

This section describes the pkg/flt package and is largely based on the original documentation provided by *Arne Biastoch* and *Alistair Adcroft* circa 2001. pkg/flt computes float trajectories and simulates the behavior of profiling floats during a model run. Profiling floats (e.g.) Argo) typically drift at depth and go back to the surface at pre-defined time intervals. However, pkg/flt can also simulate observing devices such as non-profiling floats or surface drifters.

The package's core functionalities are operated by the *flt_main* call in *forward_step* (see below for details). Checkpointing is supported via *flt_write_pickup* called in *packages_write_pickup*.

Time-stepping of float locations is based on a second- or fourth-order Runga-Kutta scheme (Press et al., 1992, Numerical Recipes). Velocities and positions are interpolated between grid points to the simulated device location, and various types of noise can be added the simulated displacements. Spatial interpolation is bilinear close to boundaries and otherwise a polynomial interpolation. Float positions are expressed in local grid index space.

9.8.2 Compile-time options in *FLT_OPTIONS.h*

CPP Flag Name	Default	Description
ALLOW_3D_FLT	#define	allow three-dimensional float displacements
USE_FLT_ALT_NOISE	#define	use alternative method of adding random noise
ALLOW_FLT_3D_NOISE	#define	add noise also to the vertical velocity of 3D floats
FLT_SECOND_ORDER_RUNGE_KUTTA	#undef	revert to old second-order Runge-Kutta
FLT_WITHOUT_X_PERIODICITY	#undef	prevent floats to re-enter the opposite side of a periodic domain
FLT_WITHOUT_Y_PERIODICITY	#undef	prevent floats to re-enter the opposite side of a periodic domain
DEVEL_FLT_EXCH2	#undef	allow experimentation with pkg/flt + exch2 despite incomplete implementation

9.8.3 Compile-time parameters in *FLT_SIZE.h* include:

parameter (*max_npart_tile* = 300) is the maximum number of floats per tile. Should be smaller than the total number of floats when running on a parallel environment but as small as possible to avoid too large arrays. The model will stop if the number of floats per tile exceeds max_npart_tile at any time.

parameter (*max_npart_exch* = 50) is the maximum number of floats per tile that can be exchanged with other tiles to one side (there are 4 arrays) in one timestep. Should be generally small because only few floats leave the tile exactly at the same time.

9.8.4 Run-time options in *data.flt* include:

flt_int_traj is the time interval in seconds to sample float position and dynamic variables (T,S,U,V,Eta). To capture the whole profile cycle of a PALACE float this has to be at least as small as the shortest surface time

flt_int_prof is the time interval in seconds to sample a whole profile of T,S,U,V (as well as positions and Eta). This has to chosen at least as small as the shortest profiling interval.

flt_noise If *FLT_NOISE* is defined then this is the amplitude that is added to the advection velocity by the random number generator.

flt_file is the base filename of the float positions without tile information and ending (e.g. *float_pos*)

flt_selectTrajOutp selects variables to output following float trajectories (=0 : none ; =1 : position only ; =2 : +p,u,v,t,s)

flt_selectProfOutp‘ selects variables to output when floats profile (=0 : none ; =1 : position only ; =2 : +p,u,v,t,s)

flt_deltaT is equal to *deltaTClock* by default

FLT_Iter0 is the time step when floats are initialized

mapIniPos2Index converts float initial positions to local, fractional indices (*.TRUE.* by default)

Notes: flt_int_prof is the time between getting profiles, not the the return cycle of the float to the surface. The latter can be specified individually for every float. Because the mechanism for returning to the surface is called in the profiling routine flt_int_prof has to be the minimum of all iup(max_npart). The subsampling of profiles can be done later in the analysis.

Notes: All profiling intervals have to be an integer multiple of *flt_int_prof*. The profile is always taken over the whole water column. For example, let's assume that one wants a first set of floats with 5 days profiling interval and 24 hours surface time, and another one with 10 days profiling interval and 12 hours surface time. To capture all of the floats motions, one then would have to set *flt_int_traj=43200* and *flt_int_prof=432000*.

9.8.5 Input Files

If *nIter0.EQ.FLT_Iter0* then *flt_init_varia* first looks for a global file (e.g. *float_pos.data*). If that file does not exists then *flt_init_varia* looks for local files (e.g. *float_pos.001.001.data*, etc.) or for local pickup files that have been generated during a previous model run (e.g. *pickup_flt.ckptA.001.001.data*, etc.).

The first line of these input file provides:

- the number of floats on that tile in the first record
- the total number of floats in the sixth record

Notes: when using a global file at first-time initialization both fields should be the same.

Afterwards the input files contain one 9-element double-precision record for each float:

```
npart   A unique float identifier (1,2,3,...)
tstart  start date of integration of float (in s)
  - If tstart=-1 floats are integrated right from the beginning
xpart   x position of float (in units of XC)
ypart   y position of float (in units of YC)
kpart   actual vertical level of float
kfloat  target level of float
       - should be the same as kpart at the beginning
iup     flag if the float
  - should profile   ( >  0 = return cycle (in s) to surface)
  - remain at depth  ( =  0 )
```

(continues on next page)

(continued from previous page)

```
  - is a 3D float    ( = -1 ).
  - should be advected WITHOUT additional noise ( = -2 ).
    (This implies that the float is non-profiling)
  - is a mooring     ( = -3 ), i.e. the float is not advected
itop    time of float the surface (in s)
tend    end date of integration of float (in s)
  - If tend=-1 floats are integrated till the end of the integration
```

Notes: an example how to write a float file (*write_float.F*) is included in the verification experiment documented below.

9.8.6 Output Files

The output consists of 3 sets of local files:

- *pickup_flt** : last positions of floats that can be used for restart
- *float_trajectories** : trajectories of floats and actual values at depth
- *float_profiles** : profiles throughout the whole water column

9.8.7 Verification Experiment

The verification experiment is based on *exp4* (flow over a Gaussian in a channel). The two main difference is that an additional wind forcing was introduced to speed up the currents.

A few utilities are included that were supposedly used to prepare input for *pkg/flt* and / or visualize its output:

```
extra/cvfloat.F90
extra/cvprofiles.F
extra/write_float.F
input/convert_ini.m
input/read_flt_traj.m
```

9.8.8 Algorithm details

A summary of what *flt_main.F* currently does is as follows:

```
CALL FLT_RUNGA4
  CALL FLT_TRILINEAR
  or CALL FLT_BILINEAR
or CALL FLT_RUNGA2
  CALL FLT_TRILINEAR
  or CALL FLT_BILINEAR
CALL FLT_EXCH2
  CALL EXCH2_SEND_PUT_VEC_RL
  CALL EXCH2_RECV_GET_VEC_RL
or CALL FLT_EXCHG
  CALL EXCH_SEND_PUT_VEC_X_RL
  CALL EXCH_RECV_GET_VEC_X_RL
  CALL EXCH_SEND_PUT_VEC_Y_RL
  CALL EXCH_RECV_GET_VEC_Y_RL
CALL FLT_UP
```

(continues on next page)

(continued from previous page)

```
CALL FLT_DOWN
CALL FLT_TRAJ
```

A summary of included fortran files is provided inside *flt_main.F*:

```
Main Routines:
C
C     o flt_main       - Integrates the floats forward and stores
C                        positions and vertical profiles at specific
C                        time intervals.
C     o flt_readparms  - Read parameter file
C     o flt_init_fixed - Initialise fixed
C     o flt_init_varia - Initialise the floats
C     o flt_restart    - Writes restart data to file (=> renamed: flt_write_pickup)
C
C     Second Level Subroutines:
C
C     o flt_runga2     - Second order Runga-Kutta inetgration (default)
C     o flt_exchg      - Does a new distribution of floats over tiles
C                        after every integration step.
C     o flt_up         - moves float to the surface (if flag is set)
C                        and stores profiles to file
C     o flt_down       - moves float to its target depth (if flag is set)
C     o flt_traj       - stores positions and data to file
C     o flt_interp_linear  - contains blinear interpolation scheme
C     o flt_mapping        - contains mapping functions & subroutine
C     o flt_mdsreadvector  - modified mdsreadvector to read files
```

CHAPTER 10

Ocean State Estimation Packages

This chapter describes packages that have been introduced for ocean state estimation purposes and in relation with automatic differentiation (see *Automatic Differentiation*). Various examples in this chapter rely on two model configurations that can be setup as explained in *Test Cases For Estimation Package Capabilities*

10.1 ECCO: model-data comparisons using gridded data sets

Author: Gael Forget

The functionalities implemented in `pkg/ecco` are: (1) output time-averaged model fields to compare with gridded data sets; (2) compute normalized model-data distances (i.e., cost functions); (3) compute averages and transports (i.e., integrals). The former is achieved as the model runs forwards in time whereas the others occur after time-integration has completed. Following [FCH+15] the total cost function is formulated generically as

$$\mathcal{J}(\vec{u}) = \sum_i \alpha_i \left(\vec{d}_i^T R_i^{-1} \vec{d}_i \right) + \sum_j \beta_j \vec{u}^T \vec{u} \tag{10.1}$$

$$\vec{d}_i = \mathcal{P}(\vec{m}_i - \vec{o}_i) \tag{10.2}$$

$$\vec{m}_i = \mathcal{S}\mathcal{D}\mathcal{M}(\vec{v}) \tag{10.3}$$

$$\vec{v} = \mathcal{Q}(\vec{u}) \tag{10.4}$$

$$\vec{u} = \mathcal{R}(\vec{u}') \tag{10.5}$$

using symbols defined in Table 10.1. Per Equation (10.3) model counterparts ($\vec{m}_i$) to observational data ($\vec{o}_i$) derive from adjustable model parameters ($\vec{v}$) through model dynamics integration ($\mathcal{M}$), diagnostic calculations ($\mathcal{D}$), and averaging in space and time ($\mathcal{S}$). Alternatively $\mathcal{S}$ stands for subsampling in space and time in the context of Section 10.2 (*PROFILES: model-data comparisons at observed locations*). Plain model-data misfits ($\vec{m}_i - \vec{o}_i$) can be penalized directly in Eq. (10.1) but penalized misfits ($\vec{d}_i$) more generally derive from $\vec{m}_i - \vec{o}_i$ through the generic $\mathcal{P}$ post-processor (Eq. (10.2)). Eqs. (10.4)-(10.5) pertain to model control parameter adjustment capabilities described in Section 10.3 (*CTRL: Model Parameter Adjustment Capability*).

Table 10.1: Symbol used in formulating generic cost functions.

symbol	definition
$\vec{u}$	vector of nondimensional control variables
$\vec{v}$	vector of dimensional control variables
α_i, β_j	misfit and control cost function multipliers (1 by default)
R_i	data error covariance matrix (R_i^{-1} are weights)
$\vec{d_i}$	a set of model-data differences
$\vec{o_i}$	observational data vector
$\vec{m}_i$	model counterpart to $\vec{o_i}$
$\mathcal{P}$	post-processing operator (e.g., a smoother)
$\mathcal{M}$	forward model dynamics operator
$\mathcal{D}$	diagnostic computation operator
$\mathcal{S}$	averaging/subsampling operator
$\mathcal{Q}$	Pre-processing operator
$\mathcal{R}$	Pre-conditioning operator

10.1.1 Generic Cost Function

The parameters available for configuring generic cost function terms in `data.ecco` are given in Table 10.2 and examples of possible specifications are available in:

- MITgcm_contrib/verification_other/global_oce_cs32/input/data.ecco
- MITgcm_contrib/verification_other/global_oce_cs32/input_ad.sens/data.ecco
- MITgcm_contrib/gael/verification/global_oce_llc90/input.ecco_v4/data.ecco

The gridded observation file name is specified by `gencost_datafile`. Observational time series may be provided as on big file or split into yearly files finishing in '_1992', '_1993', etc. The corresponding $\vec{m}_i$ physical variable is specified via the `gencost_barfile` root (see Table 10.3). A file named as specified by `gencost_barfile` gets created where averaged fields are written progressively as the model steps forward in time. After the final time step this file is re-read by `cost_generic.F` to compute the corresponding cost function term. If `gencost_outputlevel` = 1 and `gencost_name`='foo' then `cost_generic.F` outputs model-data misfit fields (i.e., $\vec{d_i}$) to a file named 'misfit_foo.data' for offline analysis and visualization.

In the current implementation, model-data error covariance matrices R_i omit non-diagonal terms. Specifying R_i thus boils down to providing uncertainty fields (σ_i such that $R_i = \sigma_i^2$) in a file specified via `gencost_errfile`. By default σ_i is assumed to be time-invariant but a σ_i time series of the same length as the $\vec{o_i}$ time series can be provided using the `variaweight` option (Table 10.4). By default cost functions are quadratic but $\vec{d_i}^T R_i^{-1} \vec{d_i}$ can be replaced with $R_i^{-1/2} \vec{d_i}$ using the `nosumsq` option (Table 10.4).

In principle, any averaging frequency should be possible, but only 'day', 'month', 'step', and 'const' are implemented for `gencost_avgperiod`. If two different averaging frequencies are needed for a variable used in multiple cost function terms (e.g., daily and monthly) then an extension starting with '_' should be added to `gencost_barfile` (such as '_day' and '_mon').[1] If two cost function terms use the same variable and frequency, however, then using a common `gencost_barfile` saves disk space.

Climatologies of $\vec{m}_i$ can be formed from the time series of model averages in order to compare with climatologies of $\vec{o_i}$ by activating the 'clim' option via `gencost_preproc` and setting the corresponding `gencost_preproc_i` integer parameter to the number of records (i.e., a # of months, days, or time steps) per climatological cycle. The generic post-processor ($\mathcal{P}$ in Eq. (10.2)) also allows model-data misfits to be, for example, smoothed in space by setting `gencost_posproc` to 'smooth' and specifying the smoother parameters via `gencost_posproc_c` and

[1] ecco_check may be missing a test for conflicting names...

gencost_posproc_i (see Table 10.4). Other options associated with the computation of Eq. (10.1) are summarized in Table 10.4 and further discussed below. Multiple gencost_preproc / gencost_posproc options may be specified per cost term.

In general the specification of gencost_name is optional, has no impact on the end-result, and only serves to distinguish between cost function terms amongst the model output (STDOUT.0000, STDERR.0000, costfunction000, misfit*.data). Exceptions listed in Table 10.6 however activate alternative cost function codes (in place of cost_generic.F) described in Section 10.1.3. In this section and in Table 10.3 (unlike in other parts of the manual) 'zonal' / 'meridional' are to be taken literally and these components are centered (i.e., not at the staggered model velocity points). Preparing gridded velocity data sets for use in cost functions thus boils down to interpolating them to XC / YC.

Table 10.2: Run-time parameters used in formulating generic cost functions and defined via *ecco_gencost_nml'* namelist in data.ecco. All parameters are vectors of length NGENCOST (the # of available cost terms) except for gencost_proc* are arrays of size NGENPPROC×NGENCOST (10 × 20 by default; can be changed in ecco.h at compile time). In addition, the gencost_is3d internal parameter is reset to true on the fly in all 3D cases in Table 10.3.

parameter	type	function
gencost_name	character(*)	Name of cost term
gencost_barfile	character(*)	File to receive model counterpart $\vec{m}_i$ (See Table 10.3)
gencost_datafile	character(*)	File containing observational data $\vec{o}_i$
gencost_avgperiod	character(5)	Averaging period for $\vec{o}_i$ and $\vec{m}_i$ (see text)
gencost_outputlevel	integer	Greater than 0 will output misfit fields
gencost_errfile	character(*)	Uncertainty field name (not used in Section 10.1.2)
gencost_mask	character(*)	Mask file name root (used only in Section 10.1.2)
mult_gencost	real	Multiplier α_i (default: 1)
gencost_preproc	character(*)	Preprocessor names
gencost_preproc_c	character(*)	Preprocessor character arguments
gencost_preproc_i	integer(*)	Preprocessor integer arguments
gencost_preproc_r	real(*)	Preprocessor real arguments
gencost_posproc	character(*)	Post-processor names
gencost_posproc_c	character(*)	Post-processor character arguments
gencost_posproc_i	integer(*)	Post-processor integer arguments
gencost_posproc_r	real(*)	Post-processor real arguments
gencost_spmin	real	Data less than this value will be omitted
gencost_spmax	real	Data greater than this value will be omitted
gencost_spzero	real	Data points equal to this value will be omitted
gencost_startdate1	integer	Start date of observations (YYYMMDD)
gencost_startdate2	integer	Start date of observations (HHMMSS)
gencost_is3d	logical	Needs to be true for 3D fields
gencost_enddate1	integer	Not fully implemented (used only in Section 10.1.3)
gencost_enddate2	integer	Not fully implemented (used only in Section 10.1.3)

Table 10.3: Implemented gencost_barfile options (as of checkpoint 65z) that can be used via cost_generic.F (Section 10.1.1). An extension starting with '_' can be appended at the end of the variable name to distinguish between separate cost function terms. Note: the 'm_eta' formula depends on the ATMOSPHERIC_LOADING and ALLOW_PSBAR_STERIC compile time options and 'useRealFreshWaterFlux' run time parameter.

variable name	description	remarks
m_eta	sea surface height	free surface + ice + global steric correction
m_sst	sea surface temperature	first level potential temperature
m_sss	sea surface salinity	first level salinity
m_bp	bottom pressure	phiHydLow
m_siarea	sea-ice area	from pkg/seaice
m_siheff	sea-ice effective thickness	from pkg/seaice
m_sihsnow	snow effective thickness	from pkg/seaice
m_theta	potential temperature	three-dimensional
m_salt	salinity	three-dimensional
m_UE	zonal velocity	three-dimensional
m_VN	meridional velocity	three-dimensional
m_ustress	zonal wind stress	from pkg/exf
m_vstress	meridional wind stress	from pkg/exf
m_uwind	zonal wind	from pkg/exf
m_vwind	meridional wind	from pkg/exf
m_atemp	atmospheric temperature	from pkg/exf
m_aqh	atmospheric specific humidity	from pkg/exf
m_precip	precipitation	from pkg/exf
m_swdown	downward shortwave	from pkg/exf
m_lwdown	downward longwave	from pkg/exf
m_wspeed	wind speed	from pkg/exf
m_diffkr	vertical/diapycnal diffusivity	three-dimensional, constant
m_kapgm	GM diffusivity	three-dimensional, constant
m_kapredi	isopycnal diffusivity	three-dimensional, constant
m_geothermalflux	geothermal heat flux	constant
m_bottomdrag	bottom drag	constant

Table 10.4: gencost_preproc and gencost_posproc options implemented as of checkpoint 65z. Note: the distinction between gencost_preproc and gencost_posproc seems unclear and may be revisited in the future.

name	description	gencost_preproc_i , _r, or _c
gencost_preproc		
clim	Use climatological misfits	integer: no. of records per climatological cycle
mean	Use time mean of misfits	—
anom	Use anomalies from time mean	—
variaweight	Use time-varying weight W_i	—
nosumsq	Use linear misfits	—
factor	Multiply $\vec{m}_i$ by a scaling factor	real: the scaling factor
gencost_posproc		
smooth	Smooth misfits	character: smoothing scale file
		integer: smoother # of time steps

10.1.2 Generic Integral Function

The functionality described in this section is operated by `cost_gencost_boxmean.F`. It is primarily aimed at obtaining a mechanistic understanding of a chosen physical variable via adjoint sensitivity computations (see *Automatic Differentiation*) as done for example in [MGZ+99][HWP+11][FWL+15]. Thus the quadratic term in Eq. (10.1) ($\vec{d}_i^T R_i^{-1} \vec{d}_i$) is by default replaced with a d_i scalar[2] that derives from model fields through a generic integral formula (Eq. (10.3)). The specification of `gencost_barfile` again selects the physical variable type. Current valid options to use `cost_gencost_boxmean.F` are reported in Table 10.5. A suffix starting with `'_'` can again be appended to `gencost_barfile`.

The integral formula is defined by masks provided via binary files which names are specified via `gencost_mask`. There are two cases: (1) if `gencost_mask = 'foo_mask'` and `gencost_barfile` is of the 'm_boxmean*' type then the model will search for horizontal, vertical, and temporal mask files named `foo_maskC`, `foo_maskK`, and `foo_maskT`; (2) if instead `gencost_barfile` is of the 'm_horflux_' type then the model will search for `foo_maskW`, `foo_maskS`, `foo_maskK`, and `foo_maskT`.

The 'C' mask or the 'W' / 'S' masks are expected to be two-dimensional fields. The 'K' and 'T' masks (both optional; all 1 by default) are expected to be one-dimensional vectors. The 'K' vector length should match Nr. The 'T' vector length should match the # of records that the specification of `gencost_avgperiod` implies but there is no restriction on its values. In case #1 ('m_boxmean*') the 'C' and 'K' masks should consists of +1 and 0 values and a volume average will be computed accordingly. In case #2 ('m_horflux*') the 'W', 'S', and 'K' masks should consists of +1, -1, and 0 values and an integrated horizontal transport (or overturn) will be computed accordingly.

Table 10.5: Implemented `gencost_barfile` options (as of checkpoint 65z) that can be used via `cost_gencost_boxmean.F` (Section 10.1.2).

variable name	description	remarks
`m_boxmean_theta`	mean of theta over box	specify box
`m_boxmean_salt`	mean of salt over box	specify box
`m_boxmean_eta`	mean of SSH over box	specify box
`m_horflux_vol`	volume transport through section	specify transect

10.1.3 Custom Cost Functions

This section (very much a work in progress...) pertains to the special cases of `cost_gencost_bpv4.F`, `cost_gencost_seaicev4.F`, `cost_gencost_sshv4.F`, `cost_gencost_sstv4.F`, and `cost_gencost_transp.F`. The cost_gencost_transp.F function can be used to compute a transport of volume, heat, or salt through a specified section (non quadratic cost function). To this end one sets `gencost_name = 'transp*'`, where $\star$ is an optional suffix starting with `'_'`, and set `gencost_barfile` to one of `m_trVol`, `m_trHeat`, and `m_trSalt`.

Note: the functionality in `cost_gencost_transp.F` is not regularly tested. Users interested in computing volumetric transports through a section are recommended to use the `m_horflux_vol` capabilities described above as it is regularly tested. Users interested in computing heat and salt transport should note the following about `m_trHeat` and `m_trSalt`:

1. The associated advection scheme with transports may be inconsistent with the model unless `ENUM_CENTERED_2ND` is implemented
2. Bolus velocities are not included
3. Diffusion components are not included

[2] The quadratic option in fact does not yet exist in `cost_gencost_boxmean.F`...

Table 10.6: Pre-defined `gencost_name` special cases (as of checkpoint 65z; Section 10.1.3).

name	description	remarks
`sshv4-mdt`	sea surface height	mean dynamic topography (SSH - geod)
`sshv4-tp`	sea surface height	Along-Track Topex/Jason SLA (level 3)
`sshv4-ers`	sea surface height	Along-Track ERS/Envisat SLA (level 3)
`sshv4-gfo`	sea surface height	Along-Track GFO class SLA (level 3)
`sshv4-lsc`	sea surface height	Large-Scale SLA (from the above)
`sshv4-gmsl`	sea surface height	Global-Mean SLA (from the above)
`bpv4-grace`	bottom pressure	GRACE maps (level 4)
`sstv4-amsre`	sea surface temperature	Along-Swath SST (level 3)
`sstv4-amsre-lsc`	sea surface temperature	Large-Scale SST (from the above)
`si4-cons`	sea ice concentration	needs sea-ice adjoint (level 4)
`si4-deconc`	model sea ice deficiency	proxy penalty (from the above)
`si4-exconc`	model sea ice excess	proxy penalty (from the above)
`transp_trVol`	volume transport	specify masks (Section 10.1.2)
`transp_trHeat`	heat transport	specify masks (Section 10.1.2)
`transp_trSalt`	salt transport	specify masks (Section 10.1.2)

10.1.4 Key Routines

TBA... `ecco_readparms.F`, `ecco_check.F`, `ecco_summary.F`, ... `cost_generic.F`, `cost_gencost_boxmean.F`, `ecco_toolbox.F`, ... `ecco_phys.F`, `cost_gencost_customize.F`, `cost_averagesfields.F`, ...

10.1.5 Compile Options

TBA... ALLOW_GENCOST_CONTRIBUTION, ALLOW_GENCOST3D, ... ALLOW_PSBAR_STERIC, ALLOW_SHALLOW_ALTIMETRY, ALLOW_HIGHLAT_ALTIMETRY, ... ALLOW_PROFILES_CONTRIBUTION, ... ALLOW_ECCO_OLD_FC_PRINT, ... ECCO_CTRL_DEPRECATED, ... packages required for some functionalities: smooth, profiles, ctrl

10.2 PROFILES: model-data comparisons at observed locations

Author: Gael Forget

The purpose of pkg/profiles is to allow sampling of MITgcm runs according to a chosen pathway (after a ship or a drifter, along altimeter tracks, etc.), typically leading to easy model-data comparisons. Given input files that contain positions and dates, pkg/profiles will interpolate the model trajectory at the observed location. In particular, pkg/profiles can be used to do model-data comparison online and formulate a least-squares problem (ECCO application).

The pkg/profiles namelist is called data.profiles. In the example below, it includes two input netcdf file names (ARGOifremer_r8.nc and XBT_v5.nc) that should be linked to the run directory and *cost function* multipliers that only matter in the context of automatic differentiation (see *Automatic Differentiation*). The first index is a file number and the second index (in mult* only) is a variable number. By convention, the variable number is an integer ranging 1 to 6: temperature, salinity, zonal velocity, meridional velocity, sea surface height anomaly, and passive tracer.

The netcdf input file structure is illustrated in the case of XBT_v5.nc To create such files, one can use the MITprof matlab toolbox obtained from https://github.com/gaelforget/MITprof . At run time, each file is scanned to determine which

variables are included; these will be interpolated. The (final) output file structure is similar but with interpolated model values in prof_T etc., and it contains model mask variables (e.g. prof_Tmask). The very model output consists of one binary (or netcdf) file per processor. The final netcdf output is to be built from those using netcdf_ecco_recompose.m (offline).

When the k2 option is used (e.g. for cubed sphere runs), the input file is to be completed with interpolation grid points and coefficients computed offline using netcdf_ecco_GenericgridMain.m. Typically, you would first provide the standard namelist and files. After detecting that interpolation information is missing, the model will generate special grid files (profilesXCincl1PointOverlap* etc.) and then stop. You then want to run netcdf_ecco_GenericgridMain.m using the special grid files. *This operation could eventually be inlined.*

Example: data.profiles

```
#
# \*****************\*
# PROFILES cost function
# \*****************\*
&PROFILES_NML
#
profilesfiles(1)= 'ARGOifremer_r8',
mult_profiles(1,1) = 1.,
mult_profiles(1,2) = 1.,
profilesfiles(2)= 'XBT_v5',
mult_profiles(2,1) = 1.,
#
/
```

Example: XBT_v5.nc

```
netcdf XBT_v5 {
dimensions:
iPROF = 278026 ;
iDEPTH = 55 ;
lTXT = 30 ;
variables:
double depth(iDEPTH) ;
depth:units = "meters" ;
double prof_YYYYMMDD(iPROF) ;
prof_YYYYMMDD:missing_value = -9999. ;
prof_YYYYMMDD:long_name = "year (4 digits), month (2 digits), day (2 digits)" ;
double prof_HHMMSS(iPROF) ;
prof_HHMMSS:missing_value = -9999. ;
prof_HHMMSS:long_name = "hour (2 digits), minute (2 digits), second (2 digits)" ;
double prof_lon(iPROF) ;
prof_lon:units = "(degree E)" ;
prof_lon:missing_value = -9999. ;
double prof_lat(iPROF) ;
prof_lat:units = "(degree N)" ;
prof_lat:missing_value = -9999. ;
char prof_descr(iPROF, lTXT) ;
prof_descr:long_name = "profile description" ;
double prof_T(iPROF, iDEPTH) ;
prof_T:long_name = "potential temperature" ;
prof_T:units = "degree Celsius" ;
prof_T:missing_value = -9999. ;
double prof_Tweight(iPROF, iDEPTH) ;
prof_Tweight:long_name = "weights" ;
prof_Tweight:units = "(degree Celsius)-2" ;
```

(continues on next page)

(continued from previous page)

```
prof_Tweight:missing_value = -9999. ;
}
```

10.3 CTRL: Model Parameter Adjustment Capability

Author: Gael Forget

The parameters available for configuring generic cost terms in `data.ctrl` are given in Table 10.7.

Table 10.7: Parameters in `ctrl_nml_genarr` namelist in `data.ctrl`. The `*` can be replaced by `arr2d`, `arr3d`, or `tim2d` for time-invariant two and three dimensional controls and time-varying 2D controls, respectively. Parameters for `genarr2d`, `genarr3d`, and `gentime2d` are arrays of length `maxCtrlArr2D`, `maxCtrlArr3D`, and `maxCtrlTim2D`, respectively, with one entry per term in the cost function.

parameter	type	function
`xx_gen*_file`	character(*)	Control Name: prefix from Table 10.8 + suffix.
`xx_gen*_weight`	character(*)	Weights in the form of $\sigma_{\vec{u}_j}^{-2}$
`xx_gen*_bounds`	real(5)	Apply bounds
`xx_gen*_preproc`	character(*)	Control preprocessor(s) (see Table 10.9)
`xx_gen*_preproc_c`	character(*)	Preprocessor character arguments
`xx_gen*_preproc_i`	integer(*)	Preprocessor integer arguments
`xx_gen*_preproc_r`	real(*)	Preprocessor real arguments
`gen*Precond`	real	Preconditioning factor ($= 1$ by default)
`mult_gen*`	real	Cost function multiplier β_j ($= 1$ by default)
`xx_gentim2d_period`	real	Frequency of adjustments (in seconds)
`xx_gentim2d_startdate1`	integer	Adjustment start date
`xx_gentim2d_startdate2`	integer	Default: model start date
`xx_gentim2d_cumsum`	logical	Accumulate control adjustments
`xx_gentim2d_glosum`	logical	Global sum of adjustment (output is still 2D)

Table 10.8: Generic control prefixes implemented as of checkpoint 65z.

	name	description
2D, time-invariant controls	genarr2d	
	xx_etan	initial sea surface height
	xx_bottomdrag	bottom drag
	xx_geothermal	geothermal heat flux
3D, time-invariant controls	genarr3d	
	xx_theta	initial potential temperature
	xx_salt	initial salinity
	xx_kapgm	GM coefficient
	xx_kapredi	isopycnal diffusivity
	xx_diffkr	diapycnal diffusivity
2D, time-varying controls	gentim2D	
	xx_atemp	atmospheric temperature
	xx_aqh	atmospheric specific humidity
	xx_swdown	downward shortwave
	xx_lwdown	downward longwave
	xx_precip	precipitation
	xx_uwind	zonal wind
	xx_vwind	meridional wind
	xx_tauu	zonal wind stress
	xx_tauv	meridional wind stress
	xx_gen_precip	globally averaged precipitation?

Table 10.9: xx_gen????d_preproc options implemented as of checkpoint 65z. Notes: [a]: If noscaling is false, the control adjustment is scaled by one on the square root of the weight before being added to the base control variable; if noscaling is true, the control is multiplied by the weight in the cost function itself.

name	description	arguments
WC01	Correlation modeling	integer: operator type (default: 1)
smooth	Smoothing without normalization	integer: operator type (default: 1)
docycle	Average period replication	integer: cycle length
replicate	Alias for docycle	(units of xx_gentim2d_period)
rmcycle	Periodic average subtraction	integer: cycle length
variaweight	Use time-varying weight	—
noscaling:math: ^{a}	Do not scale with xx_gen*_weight	—
documul	Sets xx_gentim2d_cumsum	—
doglomean	Sets xx_gentim2d_glosum	—

The control problem is non-dimensional by default, as reflected in the omission of weights in control penalties [($\vec{u}_j^T \vec{u}_j$ in (10.1)]. Non-dimensional controls ($\vec{u}_j$) are scaled to physical units ($\vec{v}_j$) through multiplication by the respective uncertainty fields ($\sigma_{\vec{u}_j}$), as part of the generic preprocessor $\mathcal{Q}$ in (10.4). Besides the scaling of $\vec{u}_j$ to physical units, the preprocessor $\mathcal{Q}$ can include, for example, spatial correlation modeling (using an implementation of Weaver and Coutier, 2001) by setting xx_gen*_preproc = 'WC01'. Alternatively, setting xx_gen*_preproc = 'smooth' activates the smoothing part of WC01, but omits the normalization. Additionally, bounds for the controls can be specified by setting xx_gen*_bounds. In forward mode, adjustments to the i^{th} control are clipped so that they remain between xx_gen*_bounds(i,1) and xx_gen*_bounds(i,4). If xx_gen*_bounds(i,1) < xx_gen*_bounds(i+1,1) for $i = 1, 2, 3$, then the bounds will "emulate a local minimum;" otherwise, the bounds have no effect in adjoint mode.

For the case of time-varying controls, the frequency is specified by `xx_gentim2d_period`. The generic control package interprets special values of `xx_gentim2d_period` in the same way as the `exf` package: a value of -12 implies cycling monthly fields while a value of 0 means that the field is steady. Time varying weights can be provided by specifying the preprocessor `variaweight`, in which case the `xx_gentim2d_weight` file must contain as many records as the control parameter time series itself (approximately the run length divided by `xx_gentim2d_period`).

The parameter `mult_gen*` sets the multiplier for the corresponding cost function penalty [β_j in (10.1); $\beta_j = 1$ by default). The preconditioner, $\mathcal{R}$, does not directly appear in the estimation problem, but only serves to push the optimization process in a certain direction in control space; this operator is specified by `gen*Precond` ($= 1$ by default).

10.4 SMOOTH: Smoothing And Covariance Model

Author: Gael Forget

TO BE CONTINUED...

10.5 The line search optimisation algorithm

Author: Patrick Heimbach

10.5.1 General features

The line search algorithm is based on a quasi-Newton variable storage method which was implemented by [GLemarechal89].

TO BE CONTINUED...

10.5.2 The online vs. offline version

- **Online version**
  Every call to *simul* refers to an execution of the forward and adjoint model. Several iterations of optimization may thus be performed within a single run of the main program (lsopt_top). The following cases may occur:
  - cold start only (no optimization)
  - cold start, followed by one or several iterations of optimization
  - warm start from previous cold start with one or several iterations
  - warm start from previous warm start with one or several iterations
- **Offline version**
  Every call to simul refers to a read procedure which reads the result of a forward and adjoint run Therefore, only one call to simul is allowed, itmax = 0, for cold start itmax = 1, for warm start Also, at the end, **x(i+1)** needs to be computed and saved to be available for the offline model and adjoint run

In order to achieve minimum difference between the online and offline code **xdiff(i+1)** is stored to file at the end of an (offline) iteration, but recomputed identically at the beginning of the next iteration.

10.5.3 Number of iterations vs. number of simulations

- itmax: controls the max. number of iterations
- nfunc: controls the max. number of simulations within one iteration

10.5.3.1 Summary

From one iteration to the next the descent direction changes. Within one iteration more than one forward and adjoint run may be performed. The updated control used as input for these simulations uses the same descent direction, but different step sizes.

10.5.3.2 Description

From one iteration to the next the descent direction dd changes using the result for the adjoint vector gg of the previous iteration. In lsline the updated control

$$\texttt{xdiff}(\texttt{i},1) = \texttt{xx}(\texttt{i}-1) + \texttt{tact}(\texttt{i}-1,1) * \texttt{dd}(\texttt{i}-1)$$

serves as input for a forward and adjoint model run yielding a new gg(i,1). In general, the new solution passes the 1st and 2nd Wolfe tests so xdiff(i,1) represents the solution sought:

$$\texttt{xx}(\texttt{i}) = \texttt{xdiff}(\texttt{i},1)$$

If one of the two tests fails, an inter- or extrapolation is invoked to determine a new step size tact(i-1,2). If more than one function call is permitted, the new step size is used together with the "old" descent direction dd(i-1) (i.e. dd is not updated using the new gg(i)), to compute a new

$$\texttt{xdiff}(\texttt{i},2) = \texttt{xx}(\texttt{i}-1) + \texttt{tact}(\texttt{i}-1,2) * \texttt{dd}(\texttt{i}-1)$$

that serves as input in a new forward and adjoint run, yielding gg(i,2). If now, both Wolfe tests are successful, the updated solution is given by

$$\texttt{xx}(\texttt{i}) = \texttt{xdiff}(\texttt{i},2) = \texttt{xx}(\texttt{i}-1) + \texttt{tact}(\texttt{i}-1,2) * \texttt{dd}(\texttt{i}-1)$$

In order to save memory both the fields dd and xdiff have a double usage.

- - in *lsopt_top*: used as x(i) - x(i-1) for Hessian update
  - in *lsline*: intermediate result for control update x = x + tact*dd
- - in *lsopt_top, lsline*: descent vector, dd = -gg and hessupd
  - in *dgscale*: intermediate result to compute new preconditioner

10.5.3.3 The parameter file lsopt.par

- **NUPDATE** max. no. of update pairs (gg(i)-gg(i-1), xx(i)-xx(i-1)) to be stored in OPWARMD to estimate Hessian [pair of current iter. is stored in (2*jmax+2, 2*jmax+3) jmax must be > 0 to access these entries] Presently NUPDATE must be > 0 (i.e. iteration without reference to previous iterations through OPWARMD has not been tested)
- **EPSX** relative precision on xx bellow which xx should not be improved
- **EPSG** relative precision on gg below which optimization is considered successful
- **IPRINT** controls verbose (>=1) or non-verbose output
- **NUMITER** max. number of iterations of optimisation; NUMTER = 0: cold start only, no optimization
- **ITER_NUM** index of new restart file to be created (not necessarily = NUMITER!)
- **NFUNC** max. no. of simulations per iteration (must be > 0); is used if step size tact is inter-/extrapolated; in this case, if NFUNC > 1, a new simulation is performed with same gradient but "improved" step size
- **FMIN** first guess cost function value (only used as long as first iteration not completed, i.e. for jmax <= 0)

10.5.3.4 OPWARMI, OPWARMD files

Two files retain values of previous iterations which are used in latest iteration to update Hessian:

- **OPWARMI**: contains index settings and scalar variables

n = nn	no. of control variables
fc = ff	cost value of last iteration
isize	no. of bytes per record in OPWARMD
m = nupdate	max. no. of updates for Hessian
jmin, jmax	pointer indices for OPWARMD file (cf. below)
gnorm0	norm of first (cold start) gradient gg
iabsiter	total number of iterations with respect to cold start

- **OPWARMD**: contains vectors (control and gradient)

entry	name	description
1	xx(i)	control vector of latest iteration
2	gg(i)	gradient of latest iteration
3	xdiff(i),diag	preconditioning vector; (1,. . . ,1) for cold start
2*jmax+2	gold=g(i)-g(i-1)	for last update (jmax)
2*jmax+3	xdiff=tact*d=xx(i)-xx (i-1)	for last update (jmax)

```
Example 1: jmin = 1, jmax = 3, mupd = 5

  1   2   3   |   4   5     6   7     8   9     empty     empty
|___|___|___| | |___|___| |___|___| |___|___| |___|___| |___|___|
      0       |     1         2         3

Example 2: jmin = 3, jmax = 7, mupd = 5   ---> jmax = 2

  1   2   3   |
|___|___|___| | |___|___| |___|___| |___|___| |___|___| |___|___|
              |     6         7         3         4         5
```

10.5.3.5 Error handling

```
lsopt_top
    |
    |---- check arguments
    |---- CALL INSTORE
    |       |
    |       |---- determine whether OPWARMI available:
    |                 * if no:  cold start: create OPWARMI
    |                 * if yes: warm start: read from OPWARMI
    |             create or open OPWARMD
    |
    |---- check consistency between OPWARMI and model parameters
    |
    |---- >>> if COLD start: <<<
    |      |  first simulation with f.g. xx_0; output: first ff_0, gg_0
    |      |  set first preconditioner value xdiff_0 to 1
    |      |  store xx(0), gg(0), xdiff(0) to OPWARMD (first 3 entries)
    |      |
    |     >>> else: WARM start: <<<
    |         read xx(i), gg(i) from OPWARMD (first 2 entries)
    |         for first warm start after cold start, i=0
    |
    |
    |
    |---- /// if ITMAX > 0: perform optimization (increment loop index i)
    |      (
    |      )---- save current values of gg(i-1) -> gold(i-1), ff -> fold(i-1)
    |      (---- CALL LSUPDXX
    |      )       |
    |      (       |---- >>> if jmax=0 <<<
    |      )       |      |  first optimization after cold start:
    |      (       |      |  preconditioner estimated via ff_0 - ff_(first guess)
    |      )       |      |  dd(i-1) = -gg(i-1)*preco
    |      (       |      |
    |      )       |     >>> if jmax > 0 <<<
    |      (       |         dd(i-1) = -gg(i-1)
    |      )       |         CALL HESSUPD
    |      (       |           |
    |      )       |           |---- dd(i-1) modified via Hessian approx.
    |      (       |
    |      )       |---- >>> if <dd,gg> >= 0 <<<
    |      (       |         ifail = 4
    |      )       |
    |      (       |---- compute step size: tact(i-1)
    |      )       |---- compute update: xdiff(i) = xx(i-1) + tact(i-1)*dd(i-1)
    |      (
    |      )---- >>> if ifail = 4 <<<
    |      (         goto 1000
    |      )
    |      (---- CALL OPTLINE / LSLINE
    |      )       |
   ...    ...     ...
```

```
...    ...
 |      )
 |      (---- CALL OPTLINE / LSLINE
```

(continues on next page)

(continued from previous page)

```
 |      )       |
 |      (       |---- /// loop over simulations
 |      )              (
 |      (              )---- CALL SIMUL
 |      )              (       |
 |      (              )       |----  input: xdiff(i)
 |      )              (       |---- output: ff(i), gg(i)
 |      (              )       |---- >>> if ONLINE <<<
 |      )              (                   runs model and adjoint
 |      (              )             >>> if OFFLINE <<<
 |      )              (                   reads those values from file
 |      (              )
 |      )              (---- 1st Wolfe test:
 |      (              )     ff(i) <= tact*xpara1*<gg(i-1),dd(i-1)>
 |      )              (
 |      (              )---- 2nd Wolfe test:
 |      )              (     <gg(i),dd(i-1)> >= xpara2*<gg(i-1),dd(i-1)>
 |      (              )
 |      )              (---- >>> if 1st and 2nd Wolfe tests ok <<<
 |      (              )      |  320: update xx: xx(i) = xdiff(i)
 |      )              (      |
 |      (              )     >>> else if 1st Wolfe test not ok <<<
 |      )              (      |  500: INTERpolate new tact:
 |      (              )      |  barr*tact < tact < (1-barr)*tact
 |      )              (      |  CALL CUBIC
 |      (              )      |
 |      )              (     >>> else if 2nd Wolfe test not ok <<<
 |      (              )         350: EXTRApolate new tact:
 |      )              (         (1+barmin)*tact < tact < 10*tact
 |      (              )         CALL CUBIC
 |      )              (
 |      (              )---- >>> if new tact > tmax <<<
 |      )              (      |  ifail = 7
 |      (              )      |
 |      )              (---- >>> if new tact < tmin OR tact*dd < machine precision <<<
 |      (              )      |  ifail = 8
 |      )              (      |
 |      (              )---- >>> else <<<
 |      )              (         update xdiff for new simulation
 |      (              )
 |      )             \\\ if nfunc > 1: use inter-/extrapolated tact and xdiff
 |      (                               for new simulation
 |      )                               N.B.: new xx is thus not based on new gg, but
 |      (                                     rather on new step size tact
 |      )
 |      (---- store new values xx(i), gg(i) to OPWARMD (first 2 entries)
 |      )---- >>> if ifail = 7,8,9 <<<
 |      (         goto 1000
 |      )
...    ...
```

```
...    ...
 |      )
 |      (---- store new values xx(i), gg(i) to OPWARMD (first 2 entries)
 |      )---- >>> if ifail = 7,8,9 <<<
 |      (         goto 1000
```

(continues on next page)

(continued from previous page)

```
|      )
|      (---- compute new pointers jmin, jmax to include latest values
|      )     gg(i)-gg(i-1), xx(i)-xx(i-1) to Hessian matrix estimate
|      (---- store gg(i)-gg(i-1), xx(i)-xx(i-1) to OPWARMD
|      )     (entries 2*jmax+2, 2*jmax+3)
|      (
|      )---- CALL DGSCALE
|      (       |
|      )       |---- call dostore
|      (       |       |
|      )       |       |---- read preconditioner of previous iteration diag(i-1)
|      (       |             from OPWARMD (3rd entry)
|      )       |
|      (       |---- compute new preconditioner diag(i), based upon diag(i-1),
|      )       |     gg(i)-gg(i-1), xx(i)-xx(i-1)
|      (       |
|      )       |---- call dostore
|      (               |
|      )               |---- write new preconditioner diag(i) to OPWARMD (3rd entry)
|      (
|---- \\\ end of optimization iteration loop
|
|
|
|---- CALL OUTSTORE
|       |
|       |---- store gnorm0, ff(i), current pointers jmin, jmax, iterabs to OPWARMI
|
|---- >>> if OFFLINE version <<<
|         xx(i+1) needs to be computed as input for offline optimization
|          |
|          |---- CALL LSUPDXX
|          |       |
|          |       |---- compute dd(i), tact(i) -> xdiff(i+1) = x(i) + tact(i)*dd(i)
|          |
|          |---- CALL WRITE_CONTROL
|          |       |
|          |       |---- write xdiff(i+1) to special file for offline optim.
|
|---- print final information
|
O
```

10.5.4 Alternative code to optim and lsopt

The non-MITgcm package optim_m1qn3 is based on the same quasi-Newton variable storage method (BFGS) [GLemarechal89] as the package in subdirectory `lsopt`, but it uses a reverse communication version of the latest (and probably last) release of the subroutine m1qn3. This avoids having to define a dummy subroutine `simul` and also simplifies the code structure. As a consequence this package is simple(r) to compile and use, because `m1qn3.f` contains all necessary subroutines and only one extra routine (`ddot`, which was copied from BLAS) is required.

The principle of reverse communication is outlined in this example:

```
external simul_rc
...
```

(continues on next page)

(continued from previous page)

```
reverse = .true.
do while (.true.)
  call m1qn3 (simul_rc,...,x,f,g,...,reverse,indic,...)
  if (reverse) break
  call simul (indic,n,x,f,g)
end while
```

`simul_rc` is an empty ''model simulator'', and `simul` generates a new state based on the value of `indic`.

The original `m1qn3` has been modified to work "offline", i.e. the simulator and the driver of `m1qn3_offline` are separate programs that are called alternatingly from a (shell-)script. This requires that the "state" of `m1qn3` is saved before this program terminates. This state is saved in a single file `OPWARM.optXXX` per simulation, where `XXX` is the simulation number. Communication with the routine, writing and restoring the state of `m1qn3` is achieved via three new common-blocks that are contained in three header files. `simul` is replaced by reading and storing the model state and gradient vectors. Schematically the driver routine `optim_sub` does the following:

```
external simul_rc
...

call optim_readdata( nn, ctrlname, ...,   xx ) ! read control vector
call optim_readdata( nn, costname, ..., adxx ) ! read gradient vector
call optim_store_m1qn3( ..., .false. )         ! read state of m1qn3
reverse = .true.
call m1qn3 (simul_rc,...,xx,objf,adxx,...,reverse,indic,...)
call optim_store_m1qn3( ..., .true. )          ! write state of m1qn3
call optim_writedata( nn, ctrlname, ..., xx )  ! write control vector
```

The optimization loop is executed outside of this program within a script.

The code can be obtained at https://github.com/mjlosch/optim_m1qn3 . The `README` contains short instructions how to build and use the code in combination with the `tutorial_global_oce_optim` experiment. The usage is very similar to the optim package.

10.6 Test Cases For Estimation Package Capabilities

First, download the model as explained in *Getting Started with MITgcm* via the MITgcm git server

```
% git clone https://github.com/user_name/MITgcm.git
```

Then, download the setup from the *MITgcm_contrib/* area by logging into the cvs server

```
% setenv CVSROOT ':pserver:cvsanon@mitgcm.org:/u/gcmpack'
% cvs login
%     ( enter the CVS password: "cvsanon" )
```

and following the directions provided here for global_oce_cs32 or here for global_oce_llc90. These model configurations are used for daily regression tests to ensure continued availability of the tested estimation package features discussed in *Ocean State Estimation Packages*. Daily results of these tests, which currently run on the *glacier* cluster, are reported on this site. To this end, one sets a crontab job that typically executes the script reported below. The various commands can also be used to run these examples outside of crontab, directly at the command line via the testreport capability.

Note: Users are advised against running *global_oce_llc90/* tests with fewer than 12 cores (96 for adjoint tests) to avoid potential memory overloads. global_oce_llc90/ (595M) uses the same LLC90 grid as the production *ECCO*

version 4 setup does [FCH+15]. The much coarser resolution global_oce_cs32/ (614M) uses the CS32 grid and can run on any modern laptop.

```
% #!/bin/csh -f
% setenv PATH ~/bin:$PATH
% setenv MODULESHOME /usr/share/Modules
% source /usr/share/Modules/init/csh
% module use /usr/share/Modules
% module load openmpi-x86_64
% setenv MPI_INC_DIR $MPI_INCLUDE
%
% cd ~/MITgcm
% #mkdir gitpull.log
% set D=`date +%Y-%m-%d`
% git pull -v > gitpull.log/gitpull.$D.log
%
% cd verification
%
% #ieee case:
% ./testreport -clean -t 'global_oce_*'
% ./testreport -of=../tools/build_options/linux_amd64_gfortran -MPI 24 -t 'global_oce_
↪*' -addr username@something.whatever
% ../tools/do_tst_2+2 -t 'global_oce_*' -mpi -exe 'mpirun -np 24 ./mitgcmuv' -a␣
↪username@something.whatever
%
% #devel case:
% ./testreport -clean -t 'global_oce_*'
% ./testreport -of=../tools/build_options/linux_amd64_gfortran -MPI 24 -devel -t
↪'global_oce_*' -addr username@something.whatever
% ../tools/do_tst_2+2 -t 'global_oce_*' -mpi -exe 'mpirun -np 24 ./mitgcmuv' -a␣
↪username@something.whatever
%
% #fast case:
% ./testreport -clean -t 'global_oce_*'
% ./testreport -of=../tools/build_options/linux_amd64_gfortran -MPI 24 -t 'global_oce_
↪*' -fast -addr username@something.whatever
% ../tools/do_tst_2+2 -t 'global_oce_*' -mpi -exe 'mpirun -np 24 ./mitgcmuv' -a␣
↪username@something.whatever
%
% #adjoint case:
% ./testreport -clean -t 'global_oce_*'
% ./testreport -of=../tools/build_options/linux_amd64_gfortran -MPI 24 -ad -t 'global_
↪oce_*' -addr username@something.whatever
```

CHAPTER 11

Under Development

CHAPTER 12

Related Projects and Highlighted Papers

12.1 Projects Related to MITgcm

12.1.1 Estimating the Circulation and Climate of the Ocean (ECCO)

ECCO is a community of MITgcm users who create and analyze ocean state estimates. ECCO typically optimizes initial conditions, surface forcing fields, and internal parameters to fit a multi-decadal model solution to various data constraints using MITgcm's adjoint capabilities. Unlike other data assimilation products, ECCO solutions are dynamically self-consistent, have closed budgets, and can easily be re-run by users.

websites: https://ecco.jpl.nasa.gov/, http://eccov4.readthedocs.io/en/latest/

12.1.2 Gcmfaces: Gridded Earth Variables In Matlab And Octave

The gcmfaces toolbox handles gridded Earth variables as sets of connected arrays. This object-oriented approach allows users to write generic, compact analysis codes that readily become applicable to a wide variety of grids. gcmfaces notably allows for analysis of MITgcm output on any of its familiar grids.

website: http://gcmfaces.readthedocs.io/en/latest/

12.1.3 MITprof: In-Situ Ocean Data In Matlab And Octave

The MITprof toolbox handles unevenly distributed in-situ ocean observations. It is notably used, along with gcmfaces, to generate input files for MITgcm's profiles package (MITgcm/pkg/profiles).

website: https://github.com/gaelforget/MITprof

12.1.4 OceanParcels - Lagrangian Particle Tracker

Parcels provides a set of Python classses and methods to create customizable particle tracking simulations, focussing on tracking of both passive water parcels as well as active plankton, plastic and fish.

website: http://oceanparcels.org/

12.1.5 Southern Ocean State Estimation (SOSE)

SOSE uses the same techniques as ECCO to produce an eddy-permitting state estimate of the Southern Ocean.

website: http://sose.ucsd.edu/

12.1.6 Xgcm: General Circulation Model Postprocessing with xarray

Xgcm is a python packge for working with the datasets produced by numerical General Circulation Models (GCMs) and similar gridded datasets that are amenable to finite volume analysis. In these datasets, different variables are located at different positions with respect to a volume or area element (e.g. cell center, cell face, etc.) xgcm solves the problem of how to interpolate and difference these variables from one position to another.

website: http://xgcm.readthedocs.io/en/latest/

12.1.7 Xmitgcm

Xmitgcm is a Python module that loads MITgcm MDS output files as xarray datasets with the associated grid information. These can be easily exported as NetCDF files.

website: http://xmitgcm.readthedocs.io/en/latest/

12.2 Highlighted Papers

Bibliography

[AHM99] Adcroft, A., C. Hill, and J. Marshall. A new treatment of the coriolis terms in c-grid models at both high and low resolutions. *Mon. Wea. Rev.*, 127:1928–1936, 1999. URL: http://mitgcm.org/pdfs/mwr_1999.pdf, doi:10.1175/1520-0493\%281999\%29127<1928:ANTOTC>2.0.CO;2.

[Adc95] A. Adcroft. *Numerical Algorithms for use in a Dynamical Model of the Ocean*. PhD thesis, Imperial College, London, 1995.

[AC04] A. Adcroft and J.-M. Campin. Re-scaled height coordinates for accurate representation of free-surface flows in ocean circulation models. *Ocean Modelling*, 7:269–284, 2004. doi:10.1016/j.ocemod.2003.09.003.

[ACHM04] A. Adcroft, J.-M. Campin, C. Hill, and J. Marshall. Implementation of an atmosphere-ocean general circulation model on the expanded spherical cube. *Mon. Wea. Rev.*, 132:2845–2863, 2004. URL: http://mitgcm.org/pdfs/mwr_2004.pdf, doi:10.1175/MWR2823.1.

[AHJMC+04] A. Adcroft, C. Hill, J.-M. Campin, J. Marshall, and P. Heimbach. Overview of the formulation and numerics of the MITgcm. In *Proceedings of the ECMWF seminar series on Numerical Methods, Recent developments in numerical methods for atmosphere and ocean modelling*, 139–149. ECMWF, 2004. URL: http://mitgcm.org/pdfs/ECMWF2004-Adcroft.pdf.

[AHM97] A.J. Adcroft, C.N. Hill, and J. Marshall. Representation of topography by shaved cells in a height coordinate ocean model. *Mon. Wea. Rev.*, 125:2293–2315, 1997. URL: http://mitgcm.org/pdfs/mwr_1997.pdf, doi:10.1175/1520-0493\%281997\%29125<2293:ROTBSC>2.0.CO;2.

[AM98] A.J. Adcroft and D. Marshall. How slippery are piecewise-constant coastlines in numerical ocean models? *Tellus*, 50(1):95–108, 1998.

[AMH+11] T. Albrecht, M. Martin, M. Haseloff, R. Winkelmann, and A. Levermann. Parameterization for subgrid-scale motion of ice-shelf calving fronts. *The Cryosphere*, 5(1):35–44, 2011. URL: https://www.the-cryosphere.net/5/35/2011/, doi:10.5194/tc-5-35-2011.

[AL77] A. Arakawa and V. Lamb. Computational design of the basic dynamical processes of the ucla general circulation model. *Meth. Comput. Phys.*, 17:174–267, 1977.

[BFKP17] Scott D. Bachman, Baylor Fox-Kemper, and Brodie Pearson. A scale-aware subgrid model for quasi-geostrophic turbulence. *J. Geophys. Res. Ocean.*, 122(2):1529–1554, 2017. doi:10.1002/2016JC012265.

[BHT99] A. Beckmann, H. H. Hellmer, and R. Timmermann. A numerical model of the weddell sea: large-scale circulation and water mass distribution. *J. Geophys. Res. Oceans*, 104(C10):23375–23391, 1999. doi:10.1029/1999JC900194.

[BHWE01] C.M. Bitz, M.M. Holland, A.J. Weaver, and M. Eby. Simulating the ice-thickness distribution in a coupled climate model. *J. Geophys. Res.*, 106:2441, 2001. doi:10.1029/1999JC000113.

[BFLM13] S. Bouillon, T. Fichefet, V. Legat, and G. Madec. The elastic-viscous-plastic method revisited. *Ocean Modelling*, 71(0):2–12, 2013. Arctic Ocean. URL: http://dx.doi.org/10.1016/j.ocemod.2013.05.013, doi:10.1016/j.ocemod.2013.05.013.

[Bry63] K. Bryan. A numerical investigation of a nonlinear model of a wind-driven ocean. *J. Atmos. Sci.*, 20:594–606, 1963.

[BC72] K. Bryan and M.D. Cox. An approximate equation of state for numerical models of ocean circulation. *J. Phys. Oceanogr.*, 2:510–514, 1972. doi:10.1175/1520-0485(1972)002<0510:AAEOSF>2.0.CO;2.

[BL79] K. Bryan and L.J. Lewis. A water mass model of the world ocean. *J. Geophys. Res.*, 84(C5):2503–2517, 1979. doi:10.1029/JC084iC05p02503.

[BMP75] K. Bryan, S. Manabe, and R.C. Pacanowski. A global ocean-atmosphere climate model. part ii. the oceanic circulation. *J. Phys. Oceanogr.*, 5:30–46, 1975.

[BH77] D.M. Burridge and J. Haseler. A model for medium range weather forecasting: adiabatic formulation. Technical Report 4, ECMWF, Bracknell, U.K., 1977. URL: https://www.ecmwf.int/sites/default/files/elibrary/1977/8495-model-medium-range-weather-forecasts-adiabatic-formulation.pdf.

[CAHM04] J.-M. Campin, A. Adcroft, C. Hill, and J. Marshall. Conservation of properties in a free-surface model. *Ocean Modelling*, 6:221–244, 2004.

[CMF08] J.-M. Campin, J. Marshall, and D. Ferreira. Sea-ice ocean coupling using a rescaled vertical coordinate z$^\ast $. *Ocean Modelling*, 24(1–2):1–14, 2008. doi:10.1016/j.ocemod.2008.05.005.

[CMKL+14] K. Castro-Morales, F. Kauker, M. Losch, S. Hendricks, K. Riemann-Campe, and R. Gerdes. Sensitivity of simulated Arctic sea ice to realistic ice thickness distributions and snow parameterizations. *J. Geophys. Res. Oceans*, 119(1):559–571, 2014. URL: http://dx.doi.org/10.1002/2013JC009342, doi:10.1002/2013JC009342.

[Cho90] M-D. Chou. Parameterizations for the absorption of solar radiation by o$_2$ and co$_2$ with applications to climate studies. *J. Clim.*, 3:209–217, 1990.

[Cho92] M-D. Chou. A solar radiation model for use in climate studies. *J. Atmos. Sci.*, 49:762–772, 1992.

[CS94] M-D. Chou and M.J. Suarez. An efficient thermal infrared radiation parameterization for use in general circulation models. NASA Technical Memorandum 104606-Vol 3, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, 1994. http://www.gmao.nasa.gov/.

[Chr94] B. Christianson. Reverse accumulation and attractive fixed points. *Optim. Method. Softw.*, 9:307–322, 1994. doi:10.1080/10556789408805572.

[Cla70] R.H. Clarke. Observational studies in the atmospheric boundary layer. *Q. J. R. Meteorol. Soc.*, 96:91–114, 1970.

[Cox87] M.D. Cox. An isopycnal diffusion in a z-coordinate ocean model. *Ocean modelling*, 74:1–5 (Unpublished manuscript), 1987.

[CRB11] B. Cushman-Roisin and J.-M. Beckers. *Introduction to Geophysical Fluid Dynamics, 2nd Edition*. Academic Press, New York, 2011.

[DT94] R.S. Defries and J.R.G. Townshend. Ndvi-derived land cover classification at global scales. *Int'l J. Rem. Sens.*, 15:3567–3586, 1994.

[DS89] J.L. Dorman and P.J. Sellers. A global climatology of albedo, roughness length and stomatal resistance for atmospheric general circulation models as represented by the simple biosphere model (sib). *J. Appl. Meteor.*, 28:833–855, 1989.

[FWDH92] G.M. Flato and III W.D. Hibler. Modeling pack ice as a cavitating fluid. *J. Phys. Oceanogr.*, 22:626–651, 1992.

[FRM83] P. Fofonoff and Jr. R. Millard. Algorithms for computation of fundamental properties of seawater. UNESCO Technical Papers in Marine Science 44, UNESCO, Paris, 1983.

[FCH+15] G. Forget, J.-M. Campin, P. Heimbach, C. N. Hill, R. M Ponte, and C. Wunsch. ECCO version 4: an integrated framework for non-linear inverse modeling and global ocean state estimation. *Geoscientific Model Development*, 8(10):3071–3104, 2015. URL: http://www.geosci-model-dev.net/8/3071/2015/, doi:10.5194/gmd-8-3071-2015.

[FWL+15] Ichiro Fukumori, Ou Wang, William Llovel, Ian Fenty, and Gael Forget. A near-uniform fluctuation of ocean bottom pressure and sea level across the deep ocean basins of the arctic ocean and the nordic seas. *Progress in Oceanography*, 134(0):152 – 172, 2015. URL: http://www.sciencedirect.com/science/article/pii/S0079661115000245, doi:http://dx.doi.org/10.1016/j.pocean.2015.01.013.

[GGregorisL90] P. Gaspar, Y. Grégoris, and J.-M. Lefevre. A simple eddy kinetic energy model for simulations of the oceanic vertical mixing: tests at station papa and long-term upper ocean study site. *J. Geophys. Res.*, 95(C9):16,179–16,193, 1990. doi:10.1029/JC095iC09p16179.

[GM90] P.R. Gent and J.C. McWilliams. Isopycnal mixing in ocean circulation models. *J. Phys. Oceanogr.*, 20:150–155, 1990.

[GWMM95] P.R. Gent, J. Willebrand, T.J. McDougall, and J.C. McWilliams. Parameterizing eddy-induced tracer transports in ocean circulation models. *J. Phys. Oceanogr.*, 25:463–474, 1995.

[GKW91] R. Gerdes, C. Koberle, and J. Willebrand. The influence of numerical advection schemes on the results of ocean general circulation models. *Clim. Dynamics*, 5(4):211–226, 1991. doi:10.1007/BF00210006.

[Gie00] R. Giering. Tangent linear and adjoint biogeochemical models. In P. Kasibhatla, M. Heimann, P. Rayner, N. Mahowald, R. G. Prinn, and D. E. Hartley, editors, *Inverse Methods in Global Biogeochemical Cycles*, pages 33–48. American Geophysical Union, Washington, D.C., 2000. doi:10.1029/GM114p0033.

[GK98] R. Giering and T. Kaminski. Recipes for adjoint code construction. *ACM Transactions on Mathematical Software*, 24(4):437–474, 1998. doi:10.1145/293686.293695.

[GLemarechal89] J.C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-newton algorithms. *Math. Programming*, 45:407–435, 1989. doi:10.1007/BF01589113.

[Gil82] A.E. Gill. *Atmosphere-Ocean Dynamics*. Academic Press, New York, 1982.

[Gol11] D.N. Goldberg. A variationally-derived, depth-integrated approximation to a higher-order glaciologial flow model. *J. of Glaciology*, 57:157–170, 2011.

[GH13] D.N. Goldberg and P. Heimbach. Parameter and state estimation with a time-dependent adjoint marine ice sheet model. *The Cryosphere*, 7:1659–1678, 2013.

[GHJS15] D.N. Goldberg, P. Heimbach, I. Joughin, and B. Smith. Committed retreat of smith, pope, and kohler glaciers over the next 30 years inferred by transient model calibration. *The Cryosphere*, 9:2429–2446, 2015.

[GNHU16] D.N. Goldberg, S.H.K. Narayanan, L. Hascoet, and J. Utke. An optimized treatment for algorithmic differentiation of an important glaciological fixed-point problem. *Geoscientific Model Development*, 9:1891–1904, 2016.

[Gri92] A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1:35–54, 1992.

[GW08] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition*. SIAM, Philadelphia, 2008.

[Gri98] S.M. Griffies. The Gent-McWilliams skew flux. *J. Phys. Oceanogr.*, 28:831–841, 1998.

[GGP+98] S.M. Griffies, A. Gnanadesikan, R.C. Pacanowski, V. Larichev, J.K. Dukowicz, and R.D. Smith. Isoneutral diffusion in a z-coordinate ocean model. *J. Phys. Oceanogr.*, 28:805–830, 1998.

[GH00] S.M. Griffies and R.W. Hallberg. Biharmonic friction with a smagorinsky-like viscosity for use in large-scale eddy-permitting ocean models. *Mon. Wea. Rev.*, 128(8):2935–2946, 2000.

[GGD97] K. Grosfeld, R. Gerdes, and J. Determann. Thermohaline circulation and interaction between ice shelf cavities and the adjacent open water. *J. Geophys. Res. Oceans*, 102(C7):15595–15610, 1997. doi:10.1029/97JC00891.

[HW65] F.H. Harlow and J.E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8:2182–2189, 1965.

[HWP+11] Patrick Heimbach, Carl Wunsch, Rui M Ponte, Gael Forget, Chris Hill, and Jean Utke. Timescales and regions of the sensitivity of Atlantic meridional volume and heat transport: toward observing system design. *Deep Sea Research Part II: Topical Studies in Oceanography*, 58(17):1858–1879, 2011.

[HS94] I.M. Held and M.J. Suarez. A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models. *Bulletin of the American Meteorological Society*, 75(10):1825–1830, 1994.

[HL88] H.M. Helfand and J.C. Labraga. Design of a non-singular level 2.5 second-order closure model for the prediction of atmospheric turbulence. *J. Atmos. Sci.*, 45:113–132, 1988.

[HS95] H.M. Helfand and S.D. Schubert. Climatology of the simulated great plains low-level jet and its contribution to the continental moisture budget of the united states. *J. Clim.*, 8:784–806, 1995.

[HO89] Hellmer, H. H. and D. J. Olbers. A two-dimensional model of the thermohaline circulation under an ice shelf. *Antarct. Sci.*, 1(4):325–336, 1989. doi:10.1017/S0954102089000490.

[Hib79] W.D. Hibler, III. A dynamic thermodynamic sea ice model. *J. Phys. Oceanogr.*, 9:815–846, 1979.

[Hib80] W.D. Hibler, III. Modeling a variable thickness sea ice cover. *Mon. Wea. Rev.*, 1:1943–1973, 1980.

[Hib84] W.D. Hibler, III. The role of sea ice dynamics in modeling co2 increases. In J. E. Hansen and T. Takahashi, editors, *Climate processes and climate sensitivity*, volume 29 of Geophysical Monograph, pages 238–253. AGU, Washington, D.C., 1984.

[HB87] W.D. Hibler, III and K. Bryan. A diagnostic ice-ocean model. *J. Phys. Oceanogr.*, 17(7):987–1015, 1987.

[HAJM99] C. Hill, A. Adcroft, D. Jamous, and J. Marshall. A strategy for terascale climate modeling. In *In Proceedings of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, 406–425. World Scientific, 1999.

[HM95] C. Hill and J. Marshall. Application of a parallel navier-stokes model to ocean circulation in parallel computational fluid dynamics. In N. Satofuka A. Ecer, J. Periaux and S. Taylor, editors, *Implementations and Results Using Parallel Computers*, pages 545–552. Elsevier Science B.V.: New York, 1995.

[HHA99] J. C. Hoe, C. Hill, and A. Adcroft. A personal supercomputer for climate research. In *SC'99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, 59. IEEE, 1999. doi:10.1109/SC.1999.10009.

[HJ99] D.M Holland and A. Jenkins. Modeling thermodynamic ice–ocean interactions at the base of an ice shelf. *J. Phys. Oceanogr.*, 29:1787–1800, 1999. doi:10.1175/1520-0485(1999)029<1787:MTIOIA>2.0.CO;2.

[Hol78] W.R. Holland. The role of mesoscale eddies in the general circulation of the ocean-numerical experiments using a wind-driven quasi-geostrophic model. *J. Phys. Oceanogr.*, 8:363–392, 1978.

[Hun01] E.C. Hunke. Viscous-plastic sea ice dynamics with the EVP model: linearization issues. *J. Comput. Phys.*, 170:18–38, 2001. doi:10.1006/jcph.2001.6710.

[HD97] E.C. Hunke and J.K. Dukowicz. An elastic-viscous-plastic model for sea ice dynamics. *J. Phys. Oceanogr.*, 27:1849–1867, 1997.

[HJL04] J.K. Hutchings, H. Jasak, and S.W. Laxon. A strength implicit correction scheme for the viscous-plastic sea ice model. *Ocean Modelling*, 7(1–2):111–133, 2004. doi:10.1016/S1463-5003(03)00040-4.

[ISI10] IOC, SCOR, and IAPSO. The international thermodynamic equation of seawater 2010 (teos-10): calculation and use of thermodynamic properties. Intergovernmental Oceanographic Commission, Manuals and Guides 56, UNESCO (English), Paris, France, 2010. URL: http://www.teos-10.org/pubs/TEOS-10_Manual.pdf.

[JM95] D. R. Jackett and T. J. McDougall. Minimal adjustment of hydrographic profiles to achieve static stability. *J. Atmos. Ocean. Technol.*, 12(4):381–389, 1995.

[JHH01] A. Jenkins, H. H. Hellmer, and D. M. Holland. The role of meltwater advection in the formulation of conservative boundary conditions at an ice-ocean interface. *J. Phys. Oceanogr.*, 31:285–296, 2001. doi:10.1175/1520-0485(2001)031<0285:TROMAI>2.0.CO;2.

[KDL15] M. Kimmritz, S. Danilov, and M. Losch. On the convergence of the modified elastic-viscous-plastic method of solving for sea-ice dynamics. *J. Comput. Phys.*, 296:90–100, 2015. doi:10.1016/j.jcp.2015.04.051.

[KDL16] M. Kimmritz, S. Danilov, and M. Losch. The adaptive EVP method for solving the sea ice momentum equation. *Ocean Modelling*, 101:59–67, 2016. doi:10.1016/j.ocemod.2016.03.004.

[KL10] J.M. Klymak and S.M. Legg. A simple mixing scheme for models that resolve breaking internal waves. *Ocean Modelling*, 33:224–234, 2010. doi:10.1016/j.ocemod.2010.02.005.

[Kon75] J. Kondo. Air-sea bulk transfer coefficients in diabatic conditions. *Bound. Layer Meteorol.*, 9:91–112, 1975.

[KS91] R.D. Koster and M.J. Suarez. A simplified treatment of sib's land surface albedo parameterization. NASA Technical Memorandum 104538, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, 1991. http://www.gmao.nasa.gov/.

[KS92] R.D. Koster and M.J. Suarez. Modeling the land surface boundary in climate models as a composite of independent vegetation stands. *J. Geophys. Res.*, 97(D3):2697–2715, 1992. doi:10.1029/91JD01696.

[LH74] A.A. Lacis and J.E. Hansen. A parameterization for the absorption of solar radiation in the earth's atmosphere. *J. Atmos. Sci.*, 31:118–133, 1974.

[LDDM97] W.G. Large, G. Danabasoglu, S.C. Doney, and J.C. McWilliams. Sensitivity to surface forcing and boundary layer mixing in a global ocean model: annual-mean climatology. *J. Phys. Oceanogr.*, 27(11):2418–2447, 1997.

[LMD94] W.G. Large, J.C. McWilliams, and S.C. Doney. Oceanic vertical mixing: a review and a model with nonlocal boundary layer parameterization. *Rev. Geophys.*, 32:363–403, 1994.

[LP81] W.G. Large and S. Pond. Open ocean momentum flux measurements in moderate to strong winds. *J. Phys. Oceanogr.*, 11:324–336, 1981.

[Lei68] C.E. Leith. Large eddy simulation of complex engineering and geophysical flows. *Physics of Fluids*, 10:1409–1416, 1968.

[Lei96] C.E. Leith. Stochastic models of chaotic systems. *Physica D.*, 98:481–491, 1996.

[LKT+12] J.-F. Lemieux, D. Knoll, B. Tremblay, D.M. Holland, and M. Losch. A comparison of the Jacobian-free Newton-Krylov method and the EVP model for solving the sea ice momentum equation with a viscous-plastic formulation: a serial algorithm study. *J. Comput. Phys.*, 231(17):5926–5944, 2012. doi:10.1016/j.jcp.2012.05.024.

[LTSedlavcek+10] J.-F. Lemieux, B. Tremblay, J. Sedláček, P. Tupper, S. Thomas, D. Huard, and J.-P. Auclair. Improving the numerical convergence of viscous-plastic sea ice models with the Jacobian-free Newton-Krylov method. *J. Comput. Phys.*, 229:2840–2852, 2010. doi:10.1016/j.jcp.2009.12.011c.

[Lepparanta83] M. Leppäranta. A growth model for black ice, snow ican and snow thickness in subarctic basins. *Nordic Hydrology*, 14:59–70, 1983.

[Lip01] W.H. Lipscomb. Remapping the thickness distribution in sea ice models. *J. Geophys. Res.*, 106(C7):13989–14000, 2001. doi:10.1029/2000JC000518.

[LHMJ07] W.H. Lipscomb, E.C. Hunke, W. Maslowski, and J. Jakacki. Ridging, strength, and stability in high-resolution sea ice models. *J. Geophys. Res.*, 112:1–18, 2007. doi:10.1029/2005JC003355.

[Los08] M. Losch. Modeling ice shelf cavities in a z-coordinate ocean general circulation model. *J. Geophys. Res. Oceans*, 113(C08043):129–144, 2008. doi:10.1029/2007JC004368.

[LFLV14] M. Losch, A. Fuchs, J.-F. Lemieux, and A. Vanselow. A parallel Jacobian-free Newton-Krylov solver for a coupled sea ice-ocean model. *J. Comput. Phys.*, 257(A):901–910, 2014. doi:10.1016/j.jcp.2013.09.026.

[LMC+10] M. Losch, D. Menemenlis, J.-M. Campin, P. Heimbach, and C. Hill. On the formulation of sea-ice models. Part 1: effects of different solver implementations and parameterizations. *Ocean Modelling*, 33(1–2):129–144, 2010. doi:10.1016/j.ocemod.2009.12.008.

[Mac89] D.R. MacAyeal. Large-scale ice flow over a viscous basal sediment: theory and application to Ice Stream B, Antarctica. *Journal of Geophysical Research – Solid Earth*, 94:4071–4087, 1989.

[MBS79] S. Manabe, K. Bryan, and M.J. Spelman. A global ocean-atmosphere climate model with seasonal variation for future studies of climate sensitivity. *Dyn. Atmos. Oceans*, 3:393–426, 1979. doi:10.1016/0377-0265(79)90021-6.

[MGZ+99] J. Marotzke, R. Giering, K.Q. Zhang, D. Stammer, C. Hill, and T. Lee. Construction of the adjoint mit ocean general circulation model and application to atlantic heat transport variability. *J. Geophys. Res.*, 104(C12):29,529–29,547, 1999. doi:10.1029/1999JC900236.

[MAC+04] J. Marshall, A. Adcroft, J.-M. Campin, C. Hill, and A. White. Atmosphere-ocean modeling exploiting fluid isomorphisms. *Mon. Wea. Rev.*, 132:2882–2894, 2004. URL: http://mitgcm.org/pdfs/a_o_iso.pdf, doi:10.1175/MWR2835.1.

[MAH+97] J. Marshall, A. Adcroft, C. Hill, L. Perelman, and C. Heisey. A finite-volume, incompressible navier stokes model for studies of the ocean on parallel computers. *J. Geophys. Res.*, 102(C3):5753–5766, 1997. URL: http://mitgcm.org/pdfs/96JC02776.pdf, doi:10.1029/96JC02775.

[MHPA97] J. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling. *J. Geophys. Res.*, 102(C3):5733–5752, 1997. URL: http://mitgcm.org/pdfs/96JC02775.pdf, doi:10.1029/96JC02776.

[MJH98] J. Marshall, H. Jones, and C. Hill. Efficient ocean modeling using non-hydrostatic algorithms. *J. Mar. Sys.*, 18:115–134, 1998. URL: http://mitgcm.org/pdfs/journal_of_marine_systems_1998.pdf, doi:10.1016/S0924-7963\%2898\%2900008-6.

[MB11] T. McDougall and P. M. Barker. Getting started with teos-10 and the gibbs seawater (gsw) oceanographic toolbox. ISBN 978-0-646-55612-5, SCOR/IAPSO WG127, 2011. URL: http://www.teos-10.org/pubs/gsw/pdf/Getting_Started.pdf.

[MJWF03] T. J. McDougall, D. R. Jackett, D. G. Wright, and R. Feistel. Accurate and computationally efficient algorithms for potential temperature and density of seawater. *J. Atmos. Ocean. Technol.*, 5:730–741, 2003.

[Mil10] F. J. Millero. History of the equation of state of seawater. *Oceanography*, 23:18–33, 2010. doi:10.5670/oceanog.2010.21.

[Mol09] A. Molod. Running GCM physics and dynamics on different grids: algorithm and tests. *Tellus*, 61A:381–393, 2009.

[MS92] S. Moorthi and M.J. Suarez. Relaxed arakawa schubert: a parameterization of moist convection for general circulation models. *Mon. Wea. Rev.*, 120:978–1002, 1992.

[Mou96] J.N. Moum. Energy-containing scales of turbulence in the ocean thermocline. *J. Geophys. Res.*, 101(C6):14095–14109, 1996. doi:10.1029/96JC00507.

[Mun50] W.H. Munk. On the wind-driven ocean circulation. *J. Meteor.*, 7:79–932, 1950.

[NUH+06] U. Naumann, J. Utke, P. Heimbach, C. Hill, D. Ozyurt, C. Wunsch, M. Fagan, N. Tallent, and and M. Strout. Adjoint code by source transformation with OpenAD/F. In Pieter Wesseling, Jacques Périaux, and Eugenio Oñate, editors, *European Conference on Computational Fluid Dynamics (ECCOMAS CFD 2006)*. TU Delft, The Netherlands, 2006.

[Orl76] I. Orlanski. A simple boundary condition for unbounded hyperbolic flows. *J. Comput. Phys.*, 21:251–269, 1976.

[PR97] T. Paluszkiewicz and R.D. Romea. A one-dimensional model for the parameterization of deep convection in the ocean. *Dyn. Atmos. Oceans*, 26:95–130, 1997.

[Pan73] H.A. Panofsky. Tower micrometeorology. In D. A. Haugen, editor, *Workshop on Micrometeorology*. American Meteorological Society, 1973.

[PW79] Claire L. Parkinson and Warren M. Washington. A large-scale numerical model of sea ice. *J. Geophys. Res.*, 84(C1):311–337, January 1979. doi:10.1029/JC084iC01p00311.

[Ped87] J. Pedlosky. *Geophysical Fluid Dynamics, Second Edition*. Spring-Verlag, New York, 1987.

[Pot73] D. Potter. *Computational Physics*. John Wiley, New York, 1973.

[Pra86] M.J. Prather. Numerical advection by conservation of second-order moments. *J. Geophys. Res.*, 91(D6):6671–6681, 1986. doi:10.1029/JD091iD06p06671.

[Red82] M.H. Redi. Oceanic Isopycnal Mixing by Coordinate Rotation. *J. Phys. Oceanogr.*, 12(10):1154–1158, oct 1982. doi:10.1175/1520-0485(1982)012<1154:OIMBCR>2.0.CO;2.

[RLG98] J. Restrepo, G. Leaf, and A. Griewank. Circumventing storage limitations in variational data assimilation studies. *SIAM J. Sci. Comput.*, 19:1586–1605, 1998.

[Roe85] P.L. Roe. Some contributions to the modelling of discontinuous flows. In B.E. Engquist, S. Osher, and R.C.J. Somerville, editors, *Large-Scale Computations in Fluid Mechanics*, volume 22 of Lectures in Applied Mathematics, pages 163–193. American Mathematical Society, Providence, RI, 1985.

[RMMB15] F. Roquet, G. Madec, T. J. McDougall, and P. M. Barker. Accurate polynomial expressions for the density and specific volume of seawater using the teos-10 standard. *Ocean Modelling*, 90:29–43, 2015. doi:10.1016/j.ocemod.2015.04.002.

[RSG87] J.E. Rosenfield, M.R. Schoeberl, and M.A. Geller. A computation of the stratospheric diabatic circulation using an accurate radiative transfer model. *J. Atmos. Sci.*, 44:859–876, 1987.

[Rot75] D.A. Rothrock. The energetics of the plastic deformation of pack ice by ridging. *J. Geophys. Res.*, 80(33):4514–4519, 1975. doi:10.1029/JC080i033p04514.

[Sad75] R. Sadourny. The dynamics of finite-difference models of the shallow-water equations. *J. Atmos. Sci.*, 32:680–689, 1975. doi:10.1175/1520-0469(1975)032<0680:TDOFDM>2.0.CO;2.

[SG94] H.E. Seim and M.C. Gregg. Detailed observations of a naturally occurring shear instability. *J. Geophys. Res.*, 99(C5):10049–10073, 1994. doi:10.1029/94JC00168.

[Sem76] A.J. Semtner, Jr. A model for the thermodynamic growth of sea ice in numerical investigations of climate. *J. Phys. Oceanogr.*, 6:379–389, 1976.

[Sha70] R. Shapiro. Smoothing, filtering, and boundary effects. *Rev. Geophys. Space Phys.*, 8(2):359–387, 1970.

[Sma63] J. Smagorinsky. General circulation experiments with the primitive equations i: the basic experiment. *Mon. Wea. Rev.*, 91(3):99–164, 1963.

[Sma93] J. Smagorinsky. Large eddy simulation of complex engineering and geophysical flows. In B. Galperin and S.A. Orszag, editors, *Evolution of Physical Oceanography*, pages 3–36. Cambridge University Press, 1993.

[Smo89] P.K. Smolarkiewicz. Comment on "a positive definite advection scheme obtained by nonlinear renormalization of the advective fluxes". *Mon. Wea. Rev.*, 117(11):2626–2632, 1989. doi:10.1175/1520-0493(1989)117<2626:COPDAS>2.0.CO;2.

[SWG+02] D. Stammer, C. Wunsch, R. Giering, C. Eckert, P. Heimbach, J. Marotzke, A. Adcroft, C. Hill, and J. Marshall. The global ocean circulation and transports during 1992 - 1997, estimated from ocean observations and a general circulation model. *J. Geophys. Res.*, 107(C9):3118, 2002. doi:10.1029/2001JC000888.

[SWG+97] D. Stammer, C. Wunsch, R. Giering, Q. Zhang, J. Marotzke, J. Marshall, and C. Hill. The global ocean circulation estimated from TOPEX/POSEIDON altimetry and a general circulation model. CGCS Report Series 49, Massachusetts Institute of Technology, Cambridge, MA, 1997. URL: https://cgcs.mit.edu/publications/cgcs-report/global-ocean-circulation-estimated-topexposeidon-altimetry-and-mit-general.

[Ste90] D.P. Stevens. On open boundary conditions for three dimensional primitive equation ocean circulation models. *Geophys. Astrophys. Fl. Dyn.*, 51:103–133, 1990.

[Sto48] H. Stommel. The western intensification of wind-driven ocean currents. *Trans. Am. Geophys. Union*, 29:206, 1948.

[SM88] Y.C. Sud and A. Molod. The roles of dry convection, cloud-radiation feedback processes and the influence of recent improvements in the parameterization of convection in the gla gcm. *Mon. Wea. Rev.*, 116:2366–2387, 1988.

[TS96] L.L. Takacs and M.J. Suarez. Dynamical aspects of climate simulations using the geos general circulation model. NASA Technical Memorandum 104606 Volume 10, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, 1996. http://www.gmao.nasa.gov/.

[TRMC75] A.S. Thorndike, D.A. Rothrock, G.A. Maykut, and R. Colony. The thickness distribution of sea ice. *J. Geophys. Res.*, 80:4501–4513, 1975.

[Tho77] S.A. Thorpe. Turbulence and mixing in a scottish loch. *Phil. Trans. R. Soc. Lond.*, 286:125–181, 1977.

[UTML17] M. Ungermann, L.B. Tremblay, T. Martin, and M. Losch. Impact of the ice strength formulation on the performance of a sea ice thickness distribution model in the Arctic. *J. Geophys. Res.*, 122(3):2090–2107, 2017. URL: http://dx.doi.org/10.1002/2016JC012128, doi:10.1002/2016JC012128.

[UNF+08] J. Utke, U. Naumann, M. Fagan, N. Tallent, M. Strout, P. Heimbach, C. Hill, and C. Wunsch. OpenAD/F: a modular open-source tool for automatic differentiation of fortran codes. *ACM Transactions on Mathematical Software (TOMS)*, 34(4):18, 2008. doi:10.1145/1377596.1377598.

[Val17] G.K. Vallis. *Atmospheric and Oceanic Fluid Dynamics: Fundamentals and Large-Scale Circulation, 2nd Edition*. Cambridge University Press, 17. doi:10.1017/9781107588417.

[VMHS97] M. Visbeck, J. Marshall, T. Haine, and M. Spall. Specification of eddy transfer coefficients in coarse-resolution ocean circulation models. *J. Phys. Oceanogr.*, 27(3):381–402, 1997.

[Waj93] R. Wajsowicz. A consistent formulation of the anisotropic stress tensor for use in models of the large-scale ocean circulation. *J. Comput. Phys.*, 105(2):333–338, 1993.

[WG94] J.C. Wesson and M.C. Gregg. Mixing at camarinal sill in the strait of gibraltar. *Q. J. R. Meteorol. Soc.*, 99 (C5):9847–9878, 1994.

[WB95] A.A. White and R.A. Bromley. Dynamically consistent, quasi-hydrostatic equations for global models with a complete representation of the coriolis force. *Q. J. R. Meteorol. Soc.*, 121:399–418, 1995. doi:10.1002/qj.49712152208.

[Wil69] G.P. Williams. Numerical integration of the three-dimensional navier stokes equations for incompressible flow. *J. Fluid Mech.*, 37:727–750, 1969.

[Win00] M. Winton. A reformulated three-layer sea ice model. *J. Atmos. Ocean. Technol.*, 17:525–531, 2000.

[YK74] A.M. Yaglom and B.A. Kader. Heat and mass transfer between a rough wall and turbulent fluid flow at high reynolds and peclet numbers. *J. Fluid Mech.*, 62:601–623, 1974.

[Yam77] T. Yamada. A numerical experiment on pollutant dispersion in a horizontally-homogenious atmospheric boundary layer. *Atmos. Environ.*, 11:1015–1024, 1977.

[ZH97] J. Zhang and W.D. Hibler, III. On an efficient numerical method for modeling sea ice dynamics. *J. Geophys. Res.*, 102(C4):8691–8702, 1997. doi:10.1029/96JC03744.

[ZWDHSR98] J. Zhang, III W.D. Hibler, M. Steele, and D.A. Rothrock. Arctic ice-ocean modeling with and without climate restoring. *J. Phys. Oceanogr.*, 28:191–217, 1998.

[ZSL95] J. Zhou, Y.C. Sud, and K.-M. Lau. Impact of orographically induced gravity wave drag in the gla gcm. *Q. J. R. Meteorol. Soc.*, 122:903–927, 1995.