OCS Study MMS 2008-???

DRAFT Technical Manual for a Coupled Sea-Ice/Ocean Circulation Model (Version 3)

Katherine S. Hedström
Arctic Region Supercomputing Center
University of Alaska Fairbanks

U.S. Department of the Interior
Minerals Management Service
Anchorage, Alaska

Contract No. ???

OCS Study MMS 2008-???

DRAFT Technical Manual for a Coupled Sea-Ice/Ocean Circulation Model (Version 3)

Katherine S. Hedström
Arctic Region Supercomputing Center
University of Alaska Fairbanks

Oct 2008

This study was funded by the Alaska Outer Continental Shelf Region of the Minerals Management Service, U.S. Department of the Interior, Anchorage, Alaska, through Contract **???** with Rutgers University, Institute of Marine and Coastal Sciences.

This document was prepared with LaTeX xfig, and inkscape.

Acknowledgments

The ROMS model is descended from the SPEM and SCRUM models, but has been entirely rewritten by Sasha Shchepetkin and Hernan Arango, with many, many other contributors. We are indebted to every one of them for their hard work.

Bill Hibler first came up with the viscous-plastic rheology we are using. Paul Budgell has rewritten the dynamic sea-ice model, improving the solution procedure and making the water-stress term implicit in time, then changing it again to use the elastic-viscous-plastic rheology of Hunke and Dukowicz. We are very grateful that he is allowing us to use his version of the code. The sea-ice thermodynamics is derived from Sirpa Häkkinen's implementation of the Mellor-Kantha scheme. She was kind enough to allow us to start with her code.

Thanks to the internet community for providing great tools like Perl, patch, cpp, svn, and gmake to aid in software development (and to make it more fun).

Development and testing of the ROMS model has been funded by...

Abstract

The Regional Ocean Modeling System (ROMS), authored by many, most notably Sasha Shchepetkin, is one approach to regional and basin-scale ocean modeling. This user's manual for ROMS describes the model equations and algorithms, as well as additional user configurations necessary for specific applications. ROMS itself has now branched out as well - the version described here is that available through the myroms.org svn site with modifications to include sea ice and other minor changes.

Contents

List of Figures

List of Tables

1 Introduction

This user's manual for the Regional Ocean Modeling System (ROMS) describes the model equations and algorithms, as well as additional user configurations necessary for specific applications. This manual also describes the sea-ice model that we are using (Budgell [5]).

The principle attributes of the model are as follows:

General

- Primitive equations with potential temperature, salinity, and an equation of state.
- Hydrostatic and Boussinesq approximations.
- Optional third-order upwind advection scheme.
- Optional Smolarkiewicz advection scheme for tracers (potential temperature, salinity, etc.).
- Optional Lagrangian floats.
- Option for point sources and sinks.

Horizontal

- Orthogonal-curvilinear coordinates.
- Arakawa C grid.
- Closed basin, periodic, prescribed, radiation, and gradient open boundary conditions.
- Masking of land areas.

Vertical

- σ (terrain-following) coordinate.
- Free surface.
- Tridiagonal solve with implicit treatment of vertical viscosity and diffusivity.

Ice

- Hunke and Dukowicz elastic-viscous-plastic dynamics.
- Mellor-Kantha thermodynamics.
- Orthogonal-curvilinear coordinates.
- Arakawa C grid.
- Smolarkiewicz or third-order upwind advection of tracers.

Mixing options

- Horizontal Laplacian and biharmonic diffusion along constant s, z or density surfaces.
- Horizontal Laplacian and biharmonic viscosity along constant s or z surfaces.
- Optional Smagorinsky horizontal viscosity and diffusion (but not recommended for diffusion).
- Horizontal free-slip or no-slip boundaries.
- Vertical harmonic viscosity and diffusion with a spatially variable coefficient, with options to compute the coefficients with Large et al. [21], Mellor-Yamada [27], or generic length scale (GLS) [45] mixing schemes.

Implementation

- Dimensional in meter, kilogram, second (MKS) units.
- Fortran 90.
- Runs under UNIX, requires the C preprocessor, gnu make, and Perl.
- All input and output is done in NetCDF [34] (Network Common Data Format), requires the NetCDF library.
- Options include serial, parallel with MPI, and parallel with OpenMP.

These lists haven't changed so very much in the past ten to fifteen years, but many of the numerical details have changed a great deal. Examples include consistent temporal averaging of the barotropic mode to guarantee both exact conservation and constancy preservation properties for tracers; redefined barotropic pressure-gradient terms to account for local variations in the density field; vertical interpolation performed using conservative parabolic splines; and higher-order, quasi-monotone advection algorithms.

ROMS now comes with a full suite of advanced data assimilation routines; these options are beyond the scope of this document.

Chapters 2 and 3 describe the model physics and numerical techniques and contain information from Haidvogel et al. [10] and blah. Chapter **??** lists the model subroutines and functions. As distributed, ROMS is ready to run with a number of example problems. The process of configuring ROMS for a particular application and running it is described in Chapter **??**, including a discussion of a few example applications. Chapter **??** describes Hernan Arango's plotting programs **cnt**, **ccnt**, **sec**, and **csec**. Chapter 4 describes the ice equations while chapter **??** describes the ice subroutines and the coupling procedure.

1.1 Acquiring the ROMS code

The version of the model described in this document is a merger between ROMS 3.1 and a sea-ice model. The main ROMS code is available for download via **svn** once you have a login on the ROMS site. The sea ice code is a branch off of that and requires special access - contact Dave Robertson (robertson@marine.rutgers.edu) for more information.

1.2 The ROMS forum

We maintain an electronic home for ROMS users at www.myroms.org. The code lives here on a subversion server, there is a discussion forum for all things ROMS, from jobs to debugging help. There's a bug tracking system, and a wiki too. The wiki is given parts of this manual as they are created, but the nature of wikis is that they can be more fluid, with more authors, than a static document such as this. Again, Dave Robertson is the one to talk to.

1.3 Warnings and bugs

ROMS is not a large program by some standards, but it is still complex enough to require some effort to use effectively. Section **??** attempts to describe what the user is responsible for—please read it carefully.

More specific things to be wary of include:

- It is recommended that you use 64 bits of precision rather than 32 bits.
- The code must be run through the C preprocessor before it is compiled. This can occasionally be dangerous, especially with the newer ANSI C versions of **cpp**. Potential problems are listed in Appendix F. The gnu **cpp** with the **-traditional** flag is known to work well.

- The vertical σ coordinate was chosen as being a sensible way to handle variations in the water depth. It has been used with success when the maximum and minimum depths differ by a factor of twenty or less, and the value of the stretching parameter, **THETA_S**, is between zero and five. It is also desirable to have the depth variations be well resolved by the horizontal grid. For realistic problems we often fail to resolve the bathymetric slopes and we then resort to bathymetric smoothing. This in turn changes the shape of the basin and leads to its own set of problems, such as altered sill depths. Also, the currents will react to the change in shelf slope—you are now solving a different problem.

2 Ocean Model Formulation

2.1 Equations of motion

ROMS is a member of a general class of three-dimensional, free-surface, terrain-following numerical models that solve the Reynolds-averaged Navier-Stokes equations using the hydrostatic and Boussinesq assumptions. The governing equations in Cartesian coordinates can be written:

$$\frac{\partial u}{\partial t} + \vec{v} \cdot \nabla u - fv = -\frac{\partial \phi}{\partial x} - \frac{\partial}{\partial z}\left(\overline{u'w'} - \nu \frac{\partial u}{\partial z}\right) + \mathcal{F}_u + \mathcal{D}_u \tag{1}$$

$$\frac{\partial v}{\partial t} + \vec{v} \cdot \nabla v + fu = -\frac{\partial \phi}{\partial y} - \frac{\partial}{\partial z}\left(\overline{v'w'} - \nu \frac{\partial v}{\partial z}\right) + \mathcal{F}_v + \mathcal{D}_v \tag{2}$$

$$\frac{\partial \phi}{\partial z} = \frac{-\rho g}{\rho_o} \tag{3}$$

with the continuity equation:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0. \tag{4}$$

and scalar transport given by:

$$\frac{\partial C}{\partial t} + \vec{v} \cdot \nabla C = -\frac{\partial}{\partial z}\left(\overline{C'w'} - \nu_\theta \frac{\partial C}{\partial z}\right) + \mathcal{F}_C + \mathcal{D}_C. \tag{5}$$

An equation of state is also required:

$$\rho = \rho(T, S, P) \tag{6}$$

The variables are shown in Table 2.1. An overbar represents a time average and a prime represents a fluctuation about the mean. These equations are closed by parameterizing the Reynolds stresses and turbulent tracer fluxes as:

$$\overline{u'w'} = -K_M \frac{\partial u}{\partial z}; \qquad \overline{v'w'} = -K_M \frac{\partial v}{\partial z}; \qquad \overline{C'w'} = -K_C \frac{\partial C}{\partial z}. \tag{7}$$

Equations (1) and (2) express the momentum balance in the x- and y-directions, respectively. The time evolution of all scalar concentration fields, including those for $T(x, y, z, t)$ and $S(x, y, z, t)$, are governed by the advective-diffusive equation (5). The equation of state is given by equation (6). In the Boussinesq approximation, density variations are neglected in the momentum equations except in their contribution to the buoyancy force in the vertical momentum equation (3). Under the hydrostatic approximation, it is further assumed that the vertical pressure gradient balances the buoyancy force. Lastly, equation (4) expresses the continuity equation for an incompressible fluid. For the moment, the effects of forcing and horizontal dissipation will be represented by the schematic terms $\mathcal{F}$ and $\mathcal{D}$, respectively. The horizontal and vertical mixing will be described more fully in §**??**.

Variable	Description
$C(x,y,z,t)$	scalar quantity, i.e. temperature, salinity, nutrient concentration
$\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_C$	optional horizontal diffusive terms
$\mathcal{F}_u, \mathcal{F}_v, \mathcal{F}_C$	forcing/source terms
$f(x,y)$	Coriolis parameter
g	acceleration of gravity
$h(x,y)$	depth of sea floor below mean sea level
$H_z(x,y,z)$	vertical grid spacing
ν, ν_θ	molecular viscosity and diffusivity
K_M, K_C	vertical eddy viscosity and diffusivity
P	total pressure $P \approx -\rho_o g z$
$\phi(x,y,z,t)$	dynamic pressure $\phi = (P/\rho_o)$
$\rho_o + \rho(x,y,z,t)$	total *in situ* density
$S(x,y,z,t)$	salinity
t	time
$T(x,y,z,t)$	potential temperature
u, v, w	the (x,y,z) components of vector velocity $\vec{v}$
x, y	horizontal coordinates
z	vertical coordinate
$\zeta(x,y,t)$	the surface elevation

Table 1: The variables used in the description of the ocean model

2.2 Vertical boundary conditions

The vertical boundary conditions can be prescribed as follows:

$$
\begin{aligned}
\text{top } (z = \zeta(x,y,t)) \qquad & K_m \tfrac{\partial u}{\partial z} = \tau_s^x(x,y,t) \\
& K_m \tfrac{\partial v}{\partial z} = \tau_s^y(x,y,t) \\
& K_C \tfrac{\partial C}{\partial z} = \tfrac{Q_C}{\rho_o c_P} \\
& w = \tfrac{\partial \zeta}{\partial t} \\
\text{and bottom } (z = -h(x,y)) \qquad & K_m \tfrac{\partial u}{\partial z} = \tau_b^x(x,y,t) \\
& K_m \tfrac{\partial v}{\partial z} = \tau_b^y(x,y,t) \\
& K_C \tfrac{\partial C}{\partial z} = 0 \\
& -w + \vec{v} \cdot \nabla h = 0.
\end{aligned}
$$

Variable	Description
Q_C	surface concentration flux
τ_s^x, τ_s^y	surface wind stress
τ_b^x, τ_b^y	bottom stress

Table 2: The variables used in the vertical boundary conditions for the ocean model

The surface boundary condition variables are defined in Table 2.2. Since Q_T is a strong function of the surface temperature, we usually choose to compute Q_T using the surface temperature and the atmospheric fields in an atmospheric bulk flux parameterization. This bulk flux routine also computes the wind stress from the winds.

On the variable bottom, $z = -h(x,y)$, the horizontal velocity has a prescribed bottom stress which is a choice between linear, quadratic, or logarithmic terms. The vertical concentration flux

may also be prescribed at the bottom, although it is usually set to zero.

2.3 Horizontal boundary conditions

As distributed, the model can easily be configured for a periodic channel, a doubly periodic domain, or a closed basin. Code is also included for open boundaries which may or may not work for your particular application. Appropriate boundary conditions are provided for u, v, T, S, and ζ.

The model domain is logically rectangular, but it is possible to mask out land areas on the boundary and in the interior. Boundary conditions on these masked regions are straightforward, with a choice of no-slip or free-slip walls.

If biharmonic friction is used, a higher order boundary condition must also be provided. The model currently has this built into the code where the biharmonic terms are calculated. The high order boundary conditions used for u are $\frac{\partial}{\partial x}\left(\nu\frac{\partial^2 u}{\partial x^2}\right) = 0$ on the eastern and western boundaries and $\frac{\partial}{\partial y}\left(\nu\frac{\partial^2 u}{\partial y^2}\right) = 0$ on the northern and southern boundaries. The boundary conditions for v and C are similar. These boundary conditions were chosen because they preserve the property of no gain or loss of volume-integrated momentum or scalar concentration.

2.4 Terrain-following coordinate system

From the point of view of the computational model, it is highly convenient to introduce a stretched vertical coordinate system which essentially "flattens out" the variable bottom at $z = -h(x, y)$. Such "σ" coordinate systems have long been used, with slight appropriate modification, in both meteorology and oceanography (e.g., Phillips [31] and Freeman et al. [7]). To proceed, we make the coordinate transformation:

$$\begin{aligned} \hat{x} &= x \\ \hat{y} &= y \\ \sigma &= \sigma(x, y, z) \\ z &= z(x, y, \sigma) \end{aligned}$$

and

$$\hat{t} = t.$$

See Appendix B for the form of σ used here. Also, see Shchepetkin and McWilliams, 2005 [35] for a discussion about the nature of this form of σ and how it differs from that used in SCRUM.

In the stretched system, the vertical coordinate σ spans the range $-1 \le \sigma \le 0$; we are therefore left with level upper ($\sigma = 0$) and lower ($\sigma = -1$) bounding surfaces. The chain rules for this transformation are:

$$\begin{aligned} \left(\frac{\partial}{\partial x}\right)_z &= \left(\frac{\partial}{\partial x}\right)_\sigma - \left(\frac{1}{H_z}\right)\left(\frac{\partial z}{\partial x}\right)_\sigma \frac{\partial}{\partial \sigma} \\ \left(\frac{\partial}{\partial y}\right)_z &= \left(\frac{\partial}{\partial y}\right)_\sigma - \left(\frac{1}{H_z}\right)\left(\frac{\partial z}{\partial y}\right)_\sigma \frac{\partial}{\partial \sigma} \\ \frac{\partial}{\partial z} &= \left(\frac{\partial s}{\partial z}\right)\frac{\partial}{\partial \sigma} = \frac{1}{H_z}\frac{\partial}{\partial \sigma} \end{aligned}$$

where

$$H_z \equiv \frac{\partial z}{\partial \sigma}$$

As a trade-off for this geometric simplification, the dynamic equations become somewhat more complicated. The resulting dynamic equations are, after dropping the carats:

$$\frac{\partial u}{\partial t} - fv + \vec{v}\cdot\nabla u = -\frac{\partial \phi}{\partial x} - \left(\frac{g\rho}{\rho_o}\right)\frac{\partial z}{\partial x} - g\frac{\partial \zeta}{\partial x} + \frac{1}{H_z}\frac{\partial}{\partial \sigma}\left[\frac{K_m}{H_z}\frac{\partial u}{\partial \sigma}\right] + \mathcal{F}_u + \mathcal{D}_u \tag{8}$$

$$\frac{\partial v}{\partial t} + fu + \vec{v}\cdot\nabla v = -\frac{\partial \phi}{\partial y} - \left(\frac{g\rho}{\rho_o}\right)\frac{\partial z}{\partial y} - g\frac{\partial \zeta}{\partial y} + \frac{1}{H_z}\frac{\partial}{\partial \sigma}\left[\frac{K_m}{H_z}\frac{\partial v}{\partial \sigma}\right] + \mathcal{F}_v + \mathcal{D}_v \tag{9}$$

$$\frac{\partial C}{\partial t} + \vec{v}\cdot\nabla C = \frac{1}{H_z}\frac{\partial}{\partial \sigma}\left[\frac{K_C}{H_z}\frac{\partial C}{\partial \sigma}\right] + \mathcal{F}_T + \mathcal{D}_T \tag{10}$$

$$\rho = \rho(T, S, P) \tag{11}$$

$$\frac{\partial \phi}{\partial \sigma} = \left(\frac{-gH_z\rho}{\rho_o}\right) \tag{12}$$

$$\frac{\partial H_z}{\partial t} + \frac{\partial (H_z u)}{\partial x} + \frac{\partial (H_z v)}{\partial y} + \frac{\partial (H_z \Omega)}{\partial \sigma} = 0 \tag{13}$$

where

$$\vec{v} = (u, v, \Omega)$$

$$\vec{v}\cdot\nabla = u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + \Omega\frac{\partial}{\partial \sigma}.$$

The vertical velocity in σ coordinates is

$$\Omega(x, y, \sigma, t) = \frac{1}{H_z}\left[w - \left(\frac{z+h}{\zeta+h}\right)\frac{\partial \zeta}{\partial t} - u\frac{\partial z}{\partial x} - v\frac{\partial z}{\partial y}\right]$$

and

$$w = \frac{\partial z}{\partial t} + u\frac{\partial z}{\partial x} + v\frac{\partial z}{\partial y} + \Omega H_z.$$

In the stretched coordinate system, the vertical boundary conditions become:

$$\begin{aligned}
&\text{top } (\sigma = 0) && \left(\tfrac{K_m}{H_z}\right)\tfrac{\partial u}{\partial \sigma} = \tau_s^x(x, y, t)\\
&&& \left(\tfrac{K_m}{H_z}\right)\tfrac{\partial v}{\partial \sigma} = \tau_s^y(x, y, t)\\
&&& \left(\tfrac{K_C}{H_z}\right)\tfrac{\partial C}{\partial \sigma} = \tfrac{Q_C}{\rho_o c_P}\\
&&& \Omega = 0\\
&\text{and bottom } (\sigma = -1) && \left(\tfrac{K_m}{H_z}\right)\tfrac{\partial u}{\partial \sigma} = \tau_b^x(x, y, t)\\
&&& \left(\tfrac{K_m}{H_z}\right)\tfrac{\partial v}{\partial \sigma} = \tau_b^y(x, y, t)\\
&&& \left(\tfrac{K_C}{H_z}\right)\tfrac{\partial C}{\partial s} = 0\\
&&& \Omega = 0.
\end{aligned}$$

Note the simplification of the boundary conditions on vertical velocity that arises from the σ coordinate transformation.

2.5 Horizontal curvilinear coordinates

In many applications of interest (e.g., flow adjacent to a coastal boundary), the fluid may be confined horizontally within an irregular region. In such problems, a horizontal coordinate system which conforms to the irregular lateral boundaries is advantageous. It is often also true in many geophysical problems that the simulated flow fields have regions of enhanced structure (e.g., boundary currents or fronts) which occupy a relatively small fraction of the physical/computational domain. In these problems, added efficiency can be gained by placing more computational resolution in such regions.

The requirement for a boundary-following coordinate system and for a laterally variable grid resolution can both be met, for suitably smooth domains, by introducing an appropriate orthogonal

coordinate transformation in the horizontal. Let the new coordinates be $\xi(x,y)$ and $\eta(x,y)$, where the relationship of horizontal arc length to the differential distance is given by:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi \tag{14}$$

$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta \tag{15}$$

Here, $m(\xi,\eta)$ and $n(\xi,\eta)$ are the scale factors which relate the differential distances $(\Delta\xi, \Delta\eta)$ to the actual (physical) arc lengths. Appendix C contains the curvilinear version of several common vector quantities.

Denoting the velocity components in the new coordinate system by

$$\vec{v} \cdot \hat{\xi} = u \tag{16}$$

and

$$\vec{v} \cdot \hat{\eta} = v \tag{17}$$

the equations of motion (8)-(13) can be re-written (see, e.g., Arakawa and Lamb [1]) as:

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{H_z u}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u^2}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z uv}{m}\right) + \frac{\partial}{\partial \sigma}\left(\frac{H_z u\Omega}{mn}\right) \\ - \left\{\left(\frac{f}{mn}\right) + v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\} H_z v = \\ - \left(\frac{H_z}{n}\right)\left(\frac{\partial \phi}{\partial \xi} + \frac{g\rho}{\rho_o}\frac{\partial z}{\partial \xi} + g\frac{\partial \zeta}{\partial \xi}\right) + \frac{1}{mn}\frac{\partial}{\partial \sigma}\left[\frac{K_m}{H_z}\frac{\partial u}{\partial \sigma}\right] + \frac{H_z}{mn}\left(\mathcal{F}_u + \mathcal{D}_u\right)\end{aligned} \tag{18}$$

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{H_z v}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z uv}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v^2}{m}\right) + \frac{\partial}{\partial \sigma}\left(\frac{H_z v\Omega}{mn}\right) \\ + \left\{\left(\frac{f}{mn}\right) + v\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - u\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right\} H_z u = \\ - \left(\frac{H_z}{m}\right)\left(\frac{\partial \phi}{\partial \eta} + \frac{g\rho}{\rho_o}\frac{\partial z}{\partial \eta} + g\frac{\partial \zeta}{\partial \eta}\right) + \frac{1}{mn}\frac{\partial}{\partial \sigma}\left[\frac{K_m}{H_z}\partial v \partial \sigma\right] + \frac{H_z}{mn}\left(\mathcal{F}_v + \mathcal{D}_v\right)\end{aligned} \tag{19}$$

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{H_z C}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z uC}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z vC}{m}\right) + \frac{\partial}{\partial \sigma}\left(\frac{H_z \Omega C}{mn}\right) = \\ \frac{1}{mn}\frac{\partial}{\partial s}\left[\frac{K_C}{H_z}\frac{\partial C}{\partial \sigma}\right] + \frac{H_z}{mn}\left(\mathcal{F}_C + \mathcal{D}_C\right)\end{aligned} \tag{20}$$

$$\rho = \rho(T, S, P) \tag{21}$$

$$\frac{\partial \phi}{\partial \sigma} = -\left(\frac{gH_z\rho}{\rho_o}\right) \tag{22}$$

$$\frac{\partial}{\partial t}\left(\frac{H_z}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v}{m}\right) + \frac{\partial}{\partial \sigma}\left(\frac{H_z \Omega}{mn}\right) = 0. \tag{23}$$

All boundary conditions remain unchanged.

3 Numerical Solution Technique

3.1 Vertical and horizontal discretization

3.1.1 Horizontal grid

In the horizontal (ξ, η), a traditional, centered, second-order finite-difference approximation is adopted. In particular, the horizontal arrangement of variables is as shown in Fig. 1. This is equivalent to the well known Arakawa "C" grid, which is well suited for problems with horizontal resolution that is fine compared to the first radius of deformation (Arakawa and Lamb [1]).

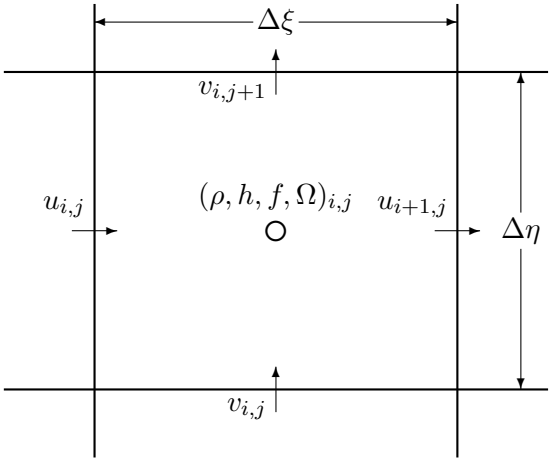

Figure 1: Placement of variables on an Arakawa C grid

3.1.2 Vertical grid

The vertical discretization also uses a second-order finite-difference approximation. Just as we use a staggered horizontal grid, the model was found to be more well-behaved with a staggered vertical grid. The vertical grid is shown in Fig. 2.

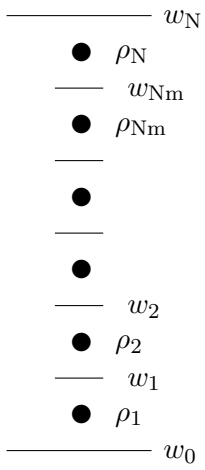

Figure 2: Placement of variables on staggered vertical grid

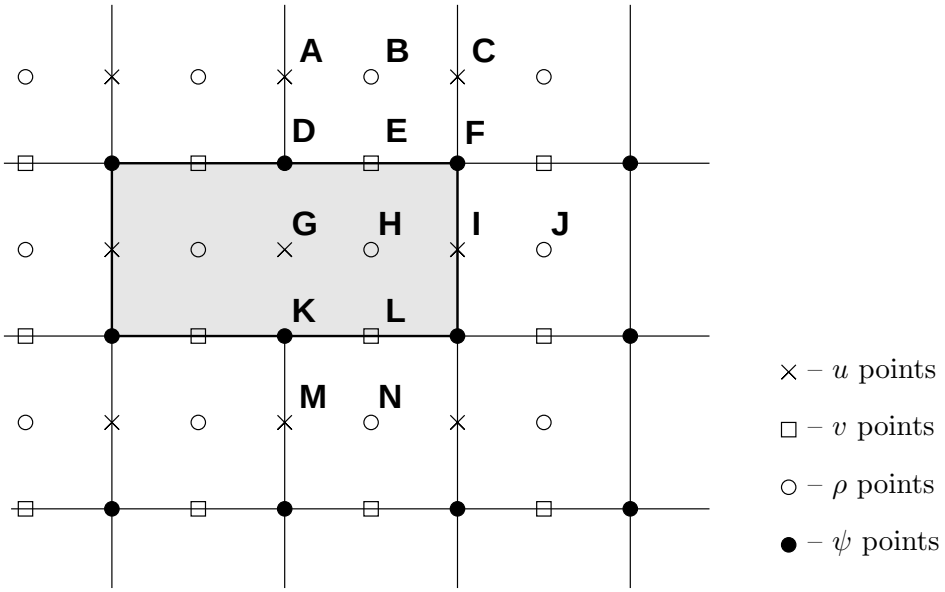

Figure 3: Masked region within the domain

3.2 Masking of land areas

ROMS has the ability to work with interior land areas, although the computations occur over the entire model domain. One grid cell is shown in Fig. 1 while several cells are shown in Fig. 3, including two land cells. The process of defining which areas are to be masked is described in §**??**, while this section describes how the masking affects the computation of the various terms in the equations of motion.

3.2.1 Velocity

At the end of every time step, the values of many variables within the masked region are set to zero by multiplying by the mask for either the u, v or ρ points. This is appropriate for the v points **E** and **L** in Fig. 3, since the flow in and out of the land should be zero. It is likewise appropriate for the u point at **I**, but is not necessarily correct for point **G**. The only term in the u equation that requires the u value at point **G** is the horizontal viscosity, which has a term of the form $\frac{\partial}{\partial\eta}\nu\frac{\partial u}{\partial\eta}$. Since point **G** is used in this term by both points **A** and **M**, it is not sufficient to replace its value with that of the image point for **A**. Instead, the term $\frac{\partial u}{\partial\eta}$ is computed and the values at points **D** and **K** are replaced with the values appropriate for either free-slip or no-slip boundary conditions. Likewise, the term $\frac{\partial}{\partial\xi}\nu\frac{\partial v}{\partial\xi}$ in the v equation must be corrected at the mask boundaries.

This is accomplished by having a fourth mask array defined at the ψ points, in which the values are set to be no-slip in **metrics**. For no-slip boundaries, we count on the values inside the land (point **G**) having been zeroed out. For point **D**, the image point at **G** should contain minus the value of u at point **A**. The desired value of $\frac{\partial u}{\partial\eta}$ is therefore $2u_{\mathbf{A}}$ while instead we have simply $u_{\mathbf{A}}$. In order to achieve the correct result, we multiply by a mask which contains the value 2 at point **D**. It also contains a 2 at point **K** so that $\frac{\partial u}{\partial\eta}$ there will acquire the desired value of $-2u_{\mathbf{M}}$. The corner point **F** is set to have a value of 1.

3.2.2 Temperature, salinity and surface elevation

The handling of masks by the temperature, salinity and surface elevation equations is similar to that in the momentum equations, and is in fact simpler. Values of T, S and ζ inside the land

masks, such as point **H** in Fig. 3, are set to zero after every time step. This point would be used by the horizontal diffusion term for points **B**, **J**, and **N**. This is corrected by setting the first derivative terms at points **E**, **I**, and **L** to zero, to be consistent with a no-flux boundary condition. Note that the equation of state must be able to handle $T = S = 0$ since this is the value inside masked regions.

3.2.3 Free surface and pressure gradients

If needed.

3.2.4 Wetting and drying

There is now an option to have wetting and drying in the model, in which a cell can switch between being wet or being dry as the tides come in and go out, for instance. Cells which are masked out as in Fig. 3 are never allowed to be wet, however.

- In the case of wetting and drying, a critical depth, D_{crit}, is supplied by the user.
- The total water depth ($D = h + \zeta$) is compared to D_{crit}. If the water level is less than this depth, no flux is allowed out of that cell. Water can always flow in and resubmerge the cell.
- The wetting and drying only happens during the 2-D computations; the 3-D computations see a depth of D_{crit} in the "dry" areas.

3.3 Time-stepping overview

While time stepping the model, we have a stored history of the model fields at time $n - 1$, an estimate of the fields at the current time n, and we need to come up with an estimate for time $n + 1$. For reasons of efficiency, we choose to use a split-explicit time step, integrating the depth-integrated equations with a shorter time step than the full 3-D equations. There is an integer ratio M between the time steps. The exact details of how the time stepping is done varies from one version of ROMS to the next, with the east coast ROMS described here being older than other branches. Still, all versions have these steps:

1. Take a predictor step for at least the 3-D tracers to time $n + \frac{1}{2}$.
2. Compute $\overline{\rho}$ and ρ^* for use in the depth-integrated time steps, from the density either at time n or time $n + \frac{1}{2}$.
3. Depth integrate the 3-D momentum right-hand side terms at time $n + \frac{1}{2}$ for use in the depth-integrated time steps (or extrapolate to obtain an estimate of those terms).
4. Take all the depth-integrated steps. Store weighted time-means of the $\overline{u}$, $\overline{v}$ fields centered at both time $n + \frac{1}{2}$ and time $n + 1$ (plus ζ at time $n + 1$). The latter requires this time stepping to extend past time $n + 1$, using M^* steps rather than just M.
5. Use the weighted time-means from depth-integrated fields to complete the corrector step for the 3-D fields to time $n + 1$.

Great care is taken to avoid the introduction of a mode-splitting instability due to the use of shorter time steps for the depth-integrated computations.

The mode coupling has evolved through the various ROMS versions, as shown in Fig. 4 (from [37]). The time stepping schemes are also listed in Table 3.3 and described in detail in [35] and [36]; the relevant ones are described in Appendix A.

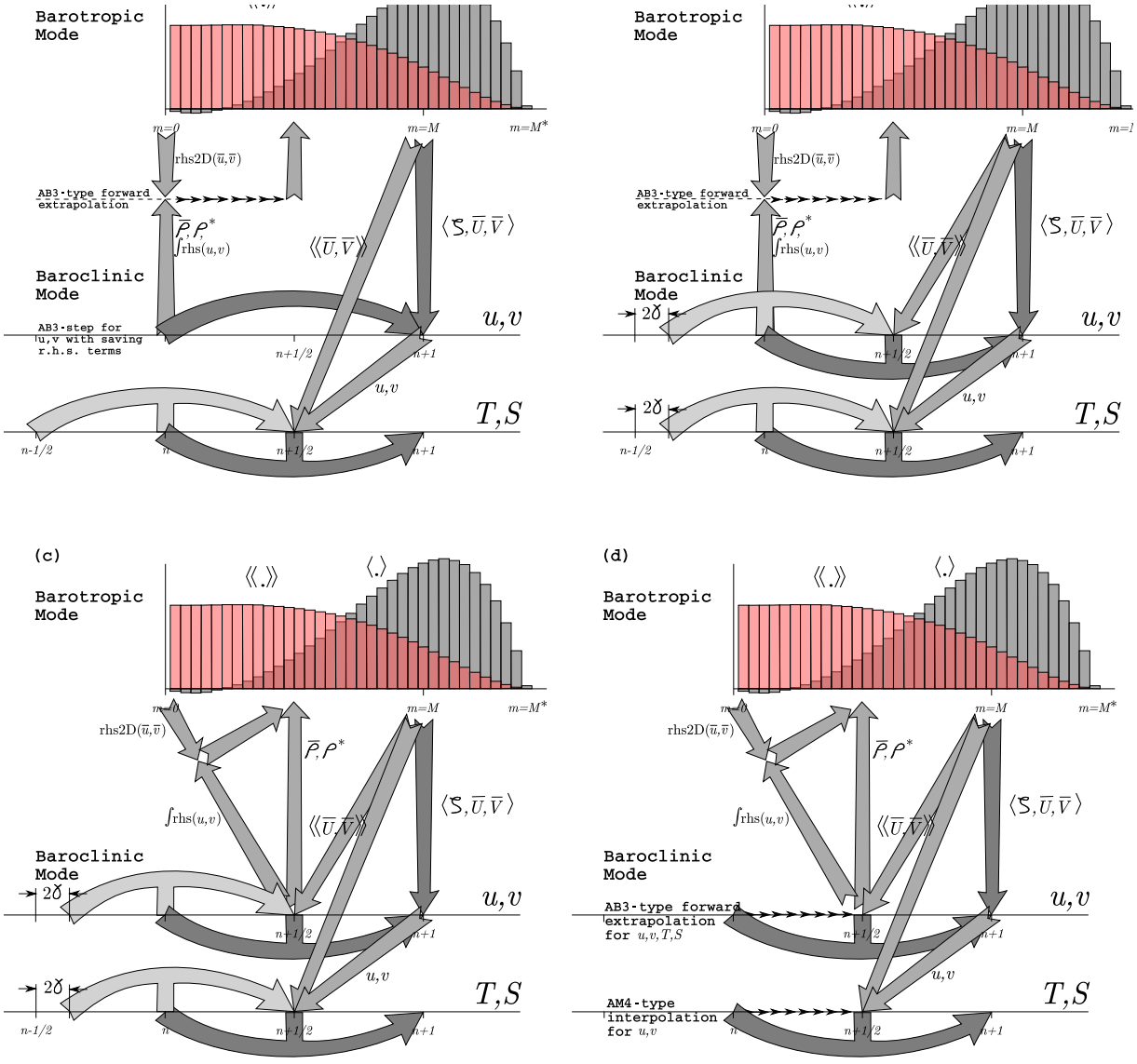

Figure 4: Diagrams of the time stepping and mode coupling used in various ROMS versions. (a) Rutgers University ROMS, (b) ROMS AGRIF, (c) UCLA ROMS, described in [35], (d) non-hydrostatic ROMS ([19]). In all, the curved arrows update the 3-D fields; those with "pillars" are leapfrog in nature with the pillar representing the r.h.s. terms. Straight arrows indicate exchange between the barotropic and baroclinic modes. The shape functions for the fast time steps show just one option out of many possibilities. The grey function has weights to produce an estimate at time $n+1$, while the light red function has weights to produce an estimate at time $n+\frac{1}{2}$.

	SCRUM 3.0	Rutgers	AGRIF	UCLA	Non-hydrostatic
Reference	[15]	[11]	[30]	[35]	[19]
Barotropic mode	LF-TR	LF-AM3 with FB feedback	LF-AM3 with FB feedback[1]	Gen. FB (AB3-AM4)	Gen. FB (AB3-AM4)
2-D $\alpha_{\max}$, iter.	$\sqrt{2}$, $(2)^2$	1.85, (2)	1.85, (2)	1.78, (1)	1.78, (1)
3-D momenta	AB3	AB3	LF-AM3	LF-AM3	AB3 (mod)
Tracers	AB3	LF-TR	LF-AM3	LF-AM3	AB3 (mod)
Internal waves	AB3	Gen. FB (AB3-TR)	LF-AM3, FB feedback	LF-AM3, FB feedback	Gen. FB (AB3-AM4)
$\alpha_{\max}$, advect.	0.72	0.72	1.587	1.587	0.78
$\alpha_{\max}$, Cor.	0.72	0.72	1.587	1.587	0.78
$\alpha_{\max}$, int. w.	0.72, (1)	1.14, (1,2)	1.85, (2)	1.85, (2)	1.78, (1)

Table 3: The time stepping schemes used in the various ROMS versions. $\alpha \equiv \omega \delta t$ is the Courant number and $\omega = ck$ is the frequency for a wave component with wavenumber k.

3.4 Conservation properties

From Shchepetkin and McWilliams (2005) [35], we have a tracer concentration equation in advective form:

$$\frac{\partial C}{\partial t} + (u \cdot \nabla) C = 0 \tag{24}$$

and also a tracer concentration equation in conservation form:

$$\frac{\partial C}{\partial t} + \nabla \cdot (uC) = 0. \tag{25}$$

The continuity equation:

$$(\nabla \cdot u) = 0 \tag{26}$$

can be used to get from one tracer equation to the other. As a consequence of eq. (24), if the tracer is spatially uniform, it will remain so regardless of the velocity field (constancy preservation). On the other hand, as a consequence of (25), the volume integral of the tracer concentration is conserved in the absence of internal sources and fluxes through the boundary. Both properties are valuable and should be retained when constructing numerical ocean models.

The semi-discrete form of the tracer equation (20) is:

$$\frac{\partial}{\partial t}\left(\frac{H_z C}{mn}\right) + \delta_\xi\left(\frac{u\overline{H_z}^{\xi}\overline{C}^{\xi}}{\overline{n}^{\xi}}\right) + \delta_\eta\left(\frac{v\overline{H_z}^{\eta}\overline{C}^{\eta}}{\overline{m}^{\eta}}\right) + \delta_\sigma\left(\overline{C}^{\sigma}\frac{H_z \Omega}{mn}\right) = \frac{1}{mn}\frac{\partial}{\partial \sigma}\left(\frac{K_m}{\Delta z}\frac{\partial C}{\partial \sigma}\right) + \mathcal{D}_C + \mathcal{F}_C \tag{27}$$

Here δ_ξ, δ_η and δ_σ denote simple centered finite-difference approximations to $\partial/\partial\xi$, $\partial/\partial\eta$ and $\partial/\partial\sigma$ with the differences taken over the distances $\Delta\xi$, $\Delta\eta$ and $\Delta\sigma$, respectively. Δz is the vertical distance from one ρ point to another. $\overline{(\quad)}^{\xi}$, $\overline{(\quad)}^{\eta}$ and $\overline{(\quad)}^{s}$ represent averages taken over the distances $\Delta\xi$, $\Delta\eta$ and $\Delta\sigma$.

The finite volume version of the same equation is no different, except that a quantity C is defined as the volume-averaged concentration over the grid box ΔV:

$$C = \frac{mn}{H_z}\int_{\Delta V}\frac{H_z C}{mn}\delta\xi\,\delta\eta\,\delta\sigma \tag{28}$$

The quantity $\left(\frac{u\overline{H_z}^{\xi}\overline{C}^{\xi}}{\overline{n}^{\xi}}\right)$ is the flux through an interface between adjacent grid boxes.

This method of averaging was chosen because it internally conserves first moments in the model domain, although it is still possible to exchange mass and energy through the open boundaries. The method is similar to that used in Arakawa and Lamb [1]; though their scheme also conserves enstrophy. Instead, we will focus on (nearly) retaining constancy preservation while coupling the barotropic (depth-integrated) equations and the baroclinic equations.

The time step in eq. (27) is assumed to be from time n to time $n+1$, with the other terms being evaluated at time $n+\frac{1}{2}$ for second-order accuracy. Setting C to 1 everywhere reduces eq. (27) to:

$$\frac{\partial}{\partial t}\left(\frac{H_z}{mn}\right)+\delta_\xi\left(\frac{u\overline{H_z}^{\xi}}{\overline{n}^{\xi}}\right)+\delta_\eta\left(\frac{v\overline{H_z}^{\eta}}{\overline{m}^{\eta}}\right)+\delta_\sigma\left(\frac{H_z\Omega}{mn}\right)=0 \tag{29}$$

If this equation holds true for the step from time n to time $n+1$, then our constancy preservation will hold.

In a hydrostatic model such as ROMS, the discrete continuity equation is needed to compute vertical velocity rather than grid-box volume $\frac{H_z}{mn}$ (the latter is controlled by changes in ζ in the barotropic mode computations). Here, $\frac{H_z\Omega}{mn}$ is the finite-volume flux across the *moving* grid-box interface, vertically on the w grid.

The vertical integral of the continuity eq. (23), using the vertical boundary conditions on Ω, is:

$$\frac{\partial}{\partial t}\left(\frac{\zeta}{mn}\right)+\delta_\xi\left(\frac{\overline{u}\overline{D}^{\xi}}{\overline{n}^{\xi}}\right)+\delta_\eta\left(\frac{\overline{v}\overline{D}^{\eta}}{\overline{m}^{\eta}}\right)=0 \tag{30}$$

where ζ is the surface elevation, $D=h+\zeta$ is the total depth, and $\overline{u},\overline{v}$ are the depth-integrated horizontal velocities. This equation and the corresponding 2-D momentum equations are time stepped on a shorter time step than eq. (27) and the other 3-D equations. Due to the details in the mode coupling, it is only possible to maintain constancy preservation to the accuracy of the barotropic time steps.

3.5 Depth-integrated equations

The depth average of a quantity A is given by:

$$\overline{A}=\frac{1}{D}\int_{-1}^{0}H_z A\,d\sigma \tag{31}$$

where the overbar indicates a vertically averaged quantity and

$$D\equiv\zeta(\xi,\eta,t)+h(\xi,\eta) \tag{32}$$

is the total depth of the water column. The vertical integral of equation (18) is:

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{D\overline{u}}{mn}\right)+\frac{\partial}{\partial\xi}\left(\frac{D\overline{uu}}{n}\right)+\frac{\partial}{\partial\eta}\left(\frac{D\overline{uv}}{m}\right)-\frac{Df\overline{v}}{mn}\\ -\left[\overline{vv}\frac{\partial}{\partial\xi}\left(\frac{1}{n}\right)-\overline{uv}\frac{\partial}{\partial\eta}\left(\frac{1}{m}\right)\right]D=-\frac{D}{n}\left(\frac{\partial\overline{\phi_2}}{\partial\xi}+g\frac{\partial\zeta}{\partial\xi}\right)\\ +\frac{D}{mn}\left(\overline{\mathcal{F}}_u+\overline{\mathcal{D}}_{h_u}\right)+\frac{1}{mn}\left(\tau_s^{\xi}-\tau_b^{\xi}\right)\end{aligned} \tag{33}$$

where ϕ_2 includes the $\frac{\partial z}{\partial\xi}$ term, $\overline{\mathcal{D}}_{h_u}$ is the horizontal viscosity and the vertical viscosity only contributes through the upper and lower boundary conditions. The corresponding vertical integral

of equation (19) is:

$$\frac{\partial}{\partial t}\left(\frac{D\overline{v}}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{uv}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{vv}}{m}\right)+\frac{Df\overline{u}}{mn} + \left[\overline{uv}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-\overline{uu}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]D=-\frac{D}{m}\left(\frac{\partial\overline{\phi_2}}{\partial \eta}+g\frac{\partial \zeta}{\partial \eta}\right) + \frac{D}{mn}\left(\overline{\mathcal{F}}_v+\overline{\mathcal{D}}_{h_v}\right)+\frac{1}{mn}\left(\tau_s^\eta-\tau_b^\eta\right). \quad (34)$$

We also need the vertical integral of equation (23), shown above as eq. (30).

The presence of a free surface introduces waves which propagate at a speed of $\sqrt{gh}$. These waves usually impose a more severe time-step limit than any of the internal processes. We have therefore chosen to solve the full equations by means of a split time step. In other words, the depth integrated equations (33), (34), and (30) are integrated using a short time step and the values of $\overline{u}$ and $\overline{v}$ are used to replace those found by integrating the full equations on a longer time step. A diagram of the barotropic time stepping is shown in Fig. 5.

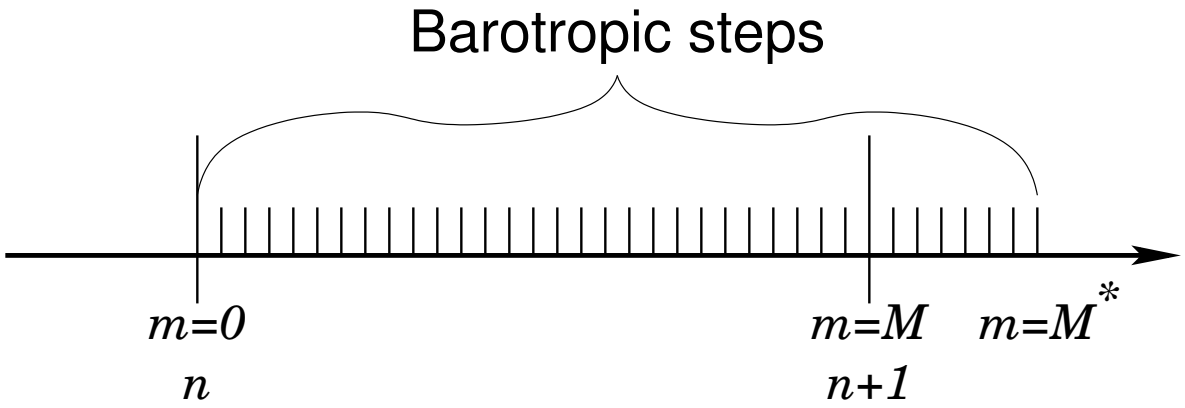

Figure 5: The split time stepping used in the model.

Some of the terms in equations (33) and (34) are updated on the short time step while others are not. The contributions from the slow terms are computed once per long time step and stored. If we call these terms $R_{u_{\rm slow}}$ and $R_{v_{\rm slow}}$, equations (33) and (34) become:

$$\frac{\partial}{\partial t}\left(\frac{D\overline{u}}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{u}\,\overline{u}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{u}\,\overline{v}}{m}\right)-\frac{Df\overline{v}}{mn} - \left[\overline{v}\,\overline{v}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-\overline{u}\,\overline{v}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]D=R_{u_{\rm slow}}-\frac{gD}{n}\frac{\partial \zeta}{\partial \xi}+\frac{D}{mn}\mathcal{D}_{\overline{u}}-\frac{1}{mn}\tau_b^\xi \quad (35)$$

$$\frac{\partial}{\partial t}\left(\frac{D\overline{v}}{mn}\right)+\frac{\partial}{\partial \xi}\left(\frac{D\overline{u}\,\overline{v}}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{D\overline{v}\,\overline{v}}{m}\right)+\frac{Df\overline{u}}{mn} + \left[\overline{u}\,\overline{v}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)-\overline{u}\,\overline{u}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]D=R_{v_{\rm slow}}-\frac{gD}{m}\frac{\partial \zeta}{\partial \eta}+\frac{D}{mn}\mathcal{D}_{\overline{v}}-\frac{1}{mn}\tau_b^\eta. \quad (36)$$

When time stepping the model, we compute the right-hand-sides for equations (18) and (19) as well as the right-hand-sides for equations (35) and (36). The vertical integral of the 3-D right-hand-sides are obtained and then the 2-D right-hand-sides are subtracted. The resulting fields are the slow forcings $R_{u_{\rm slow}}$ and $R_{v_{\rm slow}}$. This was found to be the easiest way to retain the baroclinic contributions of the non-linear terms such as $\overline{uu}-\overline{u}\,\overline{u}$.

The model is time stepped from time n to time $n+1$ by using short time steps on equations (35), (36) and (30). Equation (30) is time stepped first, so that an estimate of the new D is available for the time rate of change terms in equations (35) and (36). A third-order predictor-corrector time stepping is used. In practice, we actually time step all the way to time $(n+\mathbf{dtfast}\times M^\star)$ and while maintaining weighted averages of the values of $\overline{u}$, $\overline{v}$ and ζ. The averages are used to replace the values at time $n+1$ in both the baroclinic and barotropic modes, and for recomputing the vertical grid spacing H_z. Fig. 6 shows one option for how these weights might look.

The primary weights, a_m, are used to compute $\langle\zeta\rangle^{n+1} \equiv \sum_{m=1}^{M^\star} a_m \zeta^m$. There is a related set of secondary weights b_m, used as $\langle\langle\overline{u}\rangle\rangle^{n+\frac{1}{2}} \equiv \sum_{m=1}^{M^\star} b_m \overline{u}^m$. In order to maintain constancy preservation, this relation must hold:

$$\langle\zeta\rangle_{i,j}^{n+1} = \langle\zeta\rangle_{i,j}^{n} - (mn)_{i,j}\Delta t\left[\left\langle\left\langle\frac{D\overline{u}}{n}\right\rangle\right\rangle_{i+\frac{1}{2},j}^{n+\frac{1}{2}} - \left\langle\left\langle\frac{D\overline{u}}{n}\right\rangle\right\rangle_{i-\frac{1}{2},j}^{n+\frac{1}{2}} + \left\langle\left\langle\frac{D\overline{v}}{m}\right\rangle\right\rangle_{i,j+\frac{1}{2}}^{n+\frac{1}{2}} - \left\langle\left\langle\frac{D\overline{v}}{m}\right\rangle\right\rangle_{i,j-\frac{1}{2}}^{n+\frac{1}{2}}\right] \tag{37}$$

Shchepetkin and McWilliams ([35]) introduce a range of possible weights, but the ones used here have a shape function:

$$A(\tau) = A_0\left\{\left(\frac{\tau}{\tau_0}\right)^p\left[1-\left(\frac{\tau}{\tau_0}\right)^q\right] - r\frac{\tau}{\tau_0}\right\} \tag{38}$$

where p, q are parameters and A_0, τ_0, and r are chosen to satisfy normalization, consistency, and second-order accuracy conditions,

$$I_n = \int_0^{\tau^\star} \tau^n A(\tau) d\tau = 1, \quad n = 0, 1, 2 \tag{39}$$

using Newton iterations. $\tau^\star$ is the upper limit of τ with $A(\tau) \geq 0$. In practice we initially set

$$A_0 = 1, r = 0 \text{and} \tau = \frac{(p+2)(p+q+2)}{(p+1)(p+q+1)}$$

compute $A(\tau)$ using eq. (38), normalize using:

$$\sum_{m=1}^{M^\star} a_m \equiv 1, \quad \sum_{m=1}^{M^\star} a_m \frac{m}{M} \equiv 1, \tag{40}$$

and adjust r iteratively to satisfy the $n = 2$ condition of (39). We are using values of $p = 2$, $q = 4$, and $r = 0.284$. This form allows some negative weights for small m, allowing $M^\star$ to be less than $1.5M$.

ROMS also supports an older cosine weighting option, which isn't recommended since it is only first-order accurate.

3.6 Density in the mode coupling

Equation (35) contains the term $R_{u_{\text{slow}}}$, computed as the difference between the 3-D right-hand-side and the 2-D right-hand-side. The pressure gradient therefore has the form:

$$-\frac{gD}{n}\frac{\partial\zeta}{\partial\xi} + \left[\frac{gD}{n}\frac{\partial\zeta}{\partial\xi} + \mathcal{F}\right] \tag{41}$$

where the term in square brackets is the mode coupling term and is held fixed over all the barotropic steps and

$$\mathcal{F} = -\frac{1}{\rho_0 n}\int_{-h}^{\zeta}\frac{\partial P}{\partial\xi}dz \tag{42}$$

is the vertically integrated pressure gradient. The latter is a function of the bathymetry, free surface gradient, and the free surface itself, as well as the vertical distribution of density.

The disadvantage of this approach is that after the barotropic time stepping is complete and the new free surface is substituted into the full baroclinic pressure gradient, its vertical integral will no longer be equal to the sum of the new surface slope term and the original coupling term based on the old free surface. This is one form of mode-splitting error which can lead to trouble because the vertically integrated pressure gradient is not in balance with the barotropic mass flux.

Instead, let us define the following:

$$\overline{\rho} = \frac{1}{D}\int_{-h}^{\zeta} \rho dz, \quad \rho^{\star} = \frac{1}{\frac{1}{2}D^2}\int_{-h}^{\zeta}\left\{\int_{z}^{\zeta}\rho dz'\right\}dz \tag{43}$$

Changing the vertical coordinate to σ yields:

$$\overline{\rho} = \int_{-1}^{0} \rho d\sigma, \quad \rho^{\star} = 2\int_{-1}^{0}\left\{\int_{\sigma}^{0}\rho d\sigma'\right\}d\sigma \tag{44}$$

which implies that $\overline{\rho}$ and $\rho^{\star}$ are actually independent of ζ as long as the density profile $\rho = \rho(\sigma)$ does not change. The vertically integrated pressure gradient becomes:

$$-\frac{1}{\rho_0}\frac{g}{n}\left\{\frac{\partial}{\partial\xi}\left(\frac{\rho^{\star}D^2}{2}\right) - \overline{\rho}D\frac{\partial h}{\partial\xi}\right\} = -\frac{1}{\rho_0}\frac{g}{n}D\left\{\rho^{\star}\frac{\partial\zeta}{\partial\xi} + \frac{D}{2}\frac{\partial\rho^{\star}}{\partial\xi} + (\rho^{\star}-\overline{\rho})\frac{\partial h}{\partial\xi}\right\} \tag{45}$$

In the case of uniform density ρ_0, we obtain $\rho^{\star} \equiv \overline{\rho} \equiv \rho_0$, but we otherwise have two new terms. The accuracy of these terms depends on an accurate vertical integration of the density, as described in Shchepetkin and McWilliams (2005, [35]).

3.7 Time stepping: internal velocity modes and tracers

The momentum equations (18) and(19) are advanced before the tracer equation, by computing all the terms except the vertical viscosity and then using the implicit scheme described in §**??** to find the new values for u and v. The depth-averaged component is then removed and replaced by the $\langle\overline{u}\rangle$ and $\langle\overline{v}\rangle$ computed as in §3.5. A third-order Adams-Bashforth (AB3) time stepping is used, requiring multiple right-hand-side time levels (see Appendix A). These stored up r.h.s. values can be used to extrapolate to a value at time $n+\frac{1}{2}$ for use in the barotropic steps as shown in Fig. 4.

The tracer concentration equation (27) is advanced in a predictor-corrector leapfrog-trapezoidal step, with great care taken to optimize both the conservation and constancy-preserving properties of the continuous equations. The corrector step can maintain both, as long as it uses velocities and column depths which satisfy eq. (37). This also requires tracer values centered at time $n+\frac{1}{2}$, obtained from the predictor step. The vertical diffusion is computed as in §**??**.

The predictor step cannot be both constancy-preserving and conservative; it was therefore decided to make it constancy-preserving. Also, since it is only being used to compute the advection for the corrector step, the expensive diffusion operations are not carried out during the predictor step.

The preceeding notes on tracer advection refer to all but the MPDATA option. The MPDATA algorithm has its own predictor-corrector with emphasis on not allowing values to exceed their original range; it therefore gives up the constancy-preservation. This is most noticeable in shallow areas with large tides.

3.8 Advection schemes

Thus far, the advection scheme presented here is a centered second-order scheme. This scheme is known to have some unfortunate properties in the presence of strong gradients, such as large

over- and under-shoots of tracers, leading to the need for large amounts of horizontal smoothing. SCRUM also provides two alternative advection schemes with better behavior in many situations. At present, the alternatives are only implemented in the full 3-D engine of the model.

3.8.1 Third-order Upwind

There is a class of third-order upwind advection schemes, both one-dimensional (Leonard [22]) and two-dimensional (Rasch [32]). This scheme is known as UTOPIA (Uniformly Third-Order Polynomial Interpolation Algorithm). Applying flux limiters to UTOPIA is explored in Thuburn [44], although it is not implemented in SCRUM. The two-dimensional formulation in Rasch contains terms of order $u^2\psi$ and $u^3\psi$, including cross terms ($uv\psi$). The terms which are nonlinear in velocity have been dropped in SCRUM, leaving one extra upwind term in the computation of the advective fluxes:

$$F^{\xi} = \frac{H_z u}{n}\left(\psi - \gamma\frac{\partial^2\psi}{\partial\xi^2}\right) \tag{46}$$

$$F^{\eta} = \frac{H_z v}{m}\left(\psi - \gamma\frac{\partial^2\psi}{\partial\eta^2}\right) \tag{47}$$

The second derivative terms are centered on a ρ point in the grid, but are needed at a u or v point in the flux. The upstream value is used [see equation (**??**)]. The value of γ in the model is $\frac{1}{8}$ while that in Rasch [32] is $\frac{1}{6}$.

Because the third-order upwind scheme is designed to be two-dimensional, it is not used in the vertical (though one might argue that we are simply performing one-dimensional operations here). Instead, we use a centered fourth-order scheme in the vertical when the third-order upwind option is turned on:

$$F^{s} = \frac{H_z w}{mn}\left[-\frac{1}{16}\psi_{i,j,k-1} + \frac{9}{16}\psi_{i,j,k} + \frac{9}{16}\psi_{i,j,k+1} - \frac{1}{16}\psi_{i,j,k+2}\right] \tag{48}$$

One advantage of UTOPIA over MPDATA is that it can be used on variables having both negative and positive values. Therefore, it can be used on velocity as well as scalars (is there a reference for this?). For the u-velocity, we have:

$$F^{\xi} = \left(u - \gamma\frac{\partial^2 u}{\partial\xi^2}\right)\left[\frac{H_z u}{n} - \gamma\frac{\partial^2}{\partial\xi^2}\left(\frac{H_z u}{n}\right)\right] \tag{49}$$

$$F^{\eta} = \left(u - \gamma\frac{\partial^2 u}{\partial\eta^2}\right)\left[\frac{H_z v}{m} - \gamma\frac{\partial^2}{\partial\xi^2}\left(\frac{H_z v}{m}\right)\right] \tag{50}$$

$$F^{\sigma} = \frac{H_z w}{mn}\left[-\frac{1}{16}u_{i,j,k-1} + \frac{9}{16}u_{i,j,k} + \frac{9}{16}u_{i,j,k+1} - \frac{1}{16}u_{i,j,k+2}\right] \tag{51}$$

while for the v-velocity we have:

$$F^{\xi} = \left(v - \gamma\frac{\partial^2 v}{\partial\xi^2}\right)\left[\frac{H_z u}{n} - \gamma\frac{\partial^2}{\partial\eta^2}\left(\frac{H_z u}{n}\right)\right] \tag{52}$$

$$F^{\eta} = \left(v - \gamma\frac{\partial^2 v}{\partial\eta^2}\right)\left[\frac{H_z v}{m} - \gamma\frac{\partial^2}{\partial\eta^2}\left(\frac{H_z v}{m}\right)\right] \tag{53}$$

$$F^{\sigma} = \frac{H_z w}{mn}\left[-\frac{1}{16}v_{i,j,k-1} + \frac{9}{16}v_{i,j,k} + \frac{9}{16}v_{i,j,k+1} - \frac{1}{16}v_{i,j,k+2}\right] \tag{54}$$

In all these terms, the second derivatives are evaluated at an upstream location.

3.9 Determination of the vertical velocity and density fields

Having obtained a complete specification of the u, v, T, and S fields at the next time level by the methods outlined above, the vertical velocity and density fields can be calculated. The vertical velocity is obtained by combining equations (23) and (30) to obtain:

$$\frac{\partial}{\partial \xi}\left(\frac{H_z u}{n}\right)+\frac{\partial}{\partial \eta}\left(\frac{H_z v}{m}\right)+\frac{\partial}{\partial \sigma}\left(\frac{H_z \Omega}{mn}\right)-\frac{\partial}{\partial \xi}\left(\frac{D\overline{u}}{n}\right)-\frac{\partial}{\partial \eta}\left(\frac{D\overline{v}}{m}\right)=0. \tag{55}$$

Solving for $H_z\Omega/mn$ and using the semi-discrete notation of §3.4 we obtain:

$$\frac{H_z \Omega}{mn}=\int\left[\delta_\xi\left(\frac{\overline{u}\overline{D}^{\xi}}{\overline{n}^{\xi}}\right)+\delta_\eta\left(\frac{\overline{v}\overline{D}^{\eta}}{\overline{m}^{\eta}}\right)-\delta_\xi\left(\frac{u\overline{H_z}^{\xi}}{\overline{n}^{\xi}}\right)-\delta_\eta\left(\frac{v\overline{H_z}^{\eta}}{\overline{m}^{\eta}}\right)\right]d\sigma. \tag{56}$$

The integral is actually computed as a sum from the bottom upwards and also as a sum from the top downwards. The value used is a linear combination of the two, weighted so that the surface down value is used near the surface while the other is used near the bottom.

The density is obtained from temperature and salinity via an equation of state. SCRUM provides a choice of a nonlinear equation of state $\rho = \rho(T, S, z)$ or a linear equation of state $\rho = \rho(T)$. The nonlinear equation of state has been modified and now corresponds to the UNESCO equation of state as derived by Jackett and McDougall [18]. It computes *in situ* density as a function of potential temperature, salinity and pressure.

Warning: although we have used it quite extensively, McDougall (personal communication) claims that the single-variable ($\rho = \rho(T)$) equation of state is not dynamically appropriate as is. He has worked out the extra source and sink terms required, arising from vertical motions and the compressibility of water. They are quite complicated and we have not implemented them to see if they alter the flow.

3.10 The pressure gradient terms

The pressure gradient terms in equations (18) and (19) are written in the form

$$H_z\nabla\phi+\frac{g\rho H_z}{\rho_o}\nabla z+gH_z\nabla\zeta \tag{57}$$

This is the form traditionally used in sigma-coordinate models to account for the horizontal differences being taken along surfaces of constant σ. This form can be shown to lead to significant errors when $|\nabla h|$ is large (Haney [13]; and Beckmann and Haidvogel [4]). Shchepetkin....

3.11 Open boundary conditions

Need update for ROMS here...

3.11.1 Gradient boundary condition

This boundary condition is extremely simple and consists of setting the gradient of a field to zero at the edge. The outside value is set equal to the closest interior value. It is probably too simple to be useful in realistic problems.

3.11.2 Radiation boundary condition

In realistic domains, open boundary conditions can be extremely difficult to get right. There can be situations where incoming flow and outgoing flow happen along the same boundary or even at the same horizontal location. Orlanski [28] proposed a radiation scheme in which a local phase

velocity is computed and used to radiate things out (if it is indeed going out). This works well for a wave propagating normal to the boundary, but has problems when waves approach the boundary at an angle. Raymond and Kuo [33] have modified the scheme to account for propagation in all three directions. In SCRUM, only the two horizontal directions are accounted for:

$$\frac{\partial \psi}{\partial t} = -\left(C_x \frac{\partial \psi}{\partial \xi} + C_y \frac{\partial \psi}{\partial \eta}) \right) \tag{58}$$

where

$$C_x = \frac{F \frac{\partial \psi}{\partial \xi}}{\left(\frac{\partial \psi}{\partial \xi}\right)^2 + \left(\frac{\partial \psi}{\partial \eta}\right)^2} \tag{59}$$

$$C_y = \frac{F \frac{\partial \psi}{\partial \eta}}{\left(\frac{\partial \psi}{\partial \xi}\right)^2 + \left(\frac{\partial \psi}{\partial \eta}\right)^2} \tag{60}$$

$$F = -\frac{\partial \psi}{\partial t} \tag{61}$$

These terms are evaluated at the closest interior point in a manner consistent with the time stepping scheme used. The phase velocities are limited so that the local CFL condition is satisfied. They are then applied to the boundary point using equation (58), again using a consistent time stepping scheme. Raymond and Kuo give the form used for centered differencing and a leapfrog time step while SCRUM uses one-sided differences.

The radiation approach is appropriate for waves leaving the domain. A check is made to see which way the phase velocity is headed. If it is entering the domain, a zero gradient condition is applied.

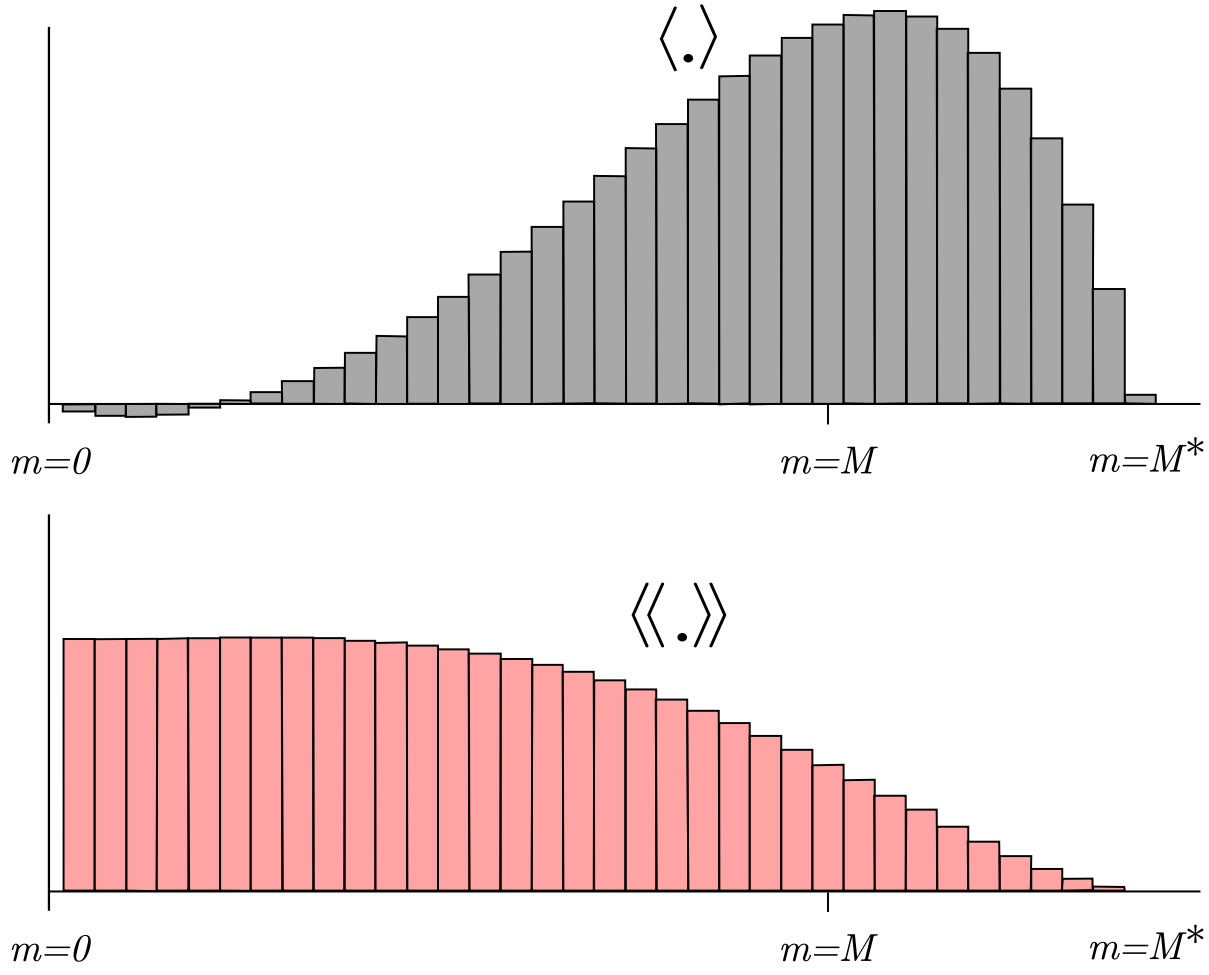

Figure 6: Weights for the barotropic time stepping. The upper panel shows the primary weights, centered at time $n+1$, while the lower panel shows the secondary weights weights, centered at time $n+\frac{1}{2}$.

4 Ice Model Formulation

Hibler [16] has described a model for the simulation of sea ice circulation and thickness. The model has been tested over a seasonal cycle and the results are also shown in that article. This report was derived from the documentation for the sea ice model written by Hibler [17] since the dynamical equations are much the same. The model itself has been rewritten by Paul Budgell and is now implemented on an orthogonal, curvilinear Arakawa C-grid, has a new elliptic solver, and the nonlinear advection of momentum has been omitted. The thermodynamics are derived from Mellor and Kantha [26] (MK89 below). Sirpa Häkkinen allowed us to use her implementation of MK89; we obtained it from the Norwegians, who call it "hakkis".

4.1 Model structure

The overall structure consists of two principal components—the momentum equations and the ice continuity equations. The momentum balance includes air and water stress, Coriolis force, internal ice stress, inertial forces and ocean tilt as shown in equations (62) and (63):

$$M\frac{\partial u}{\partial t} = Mfv - Mg\frac{\partial \zeta_w}{\partial x} + \tau_a^x + \tau_w^x + \mathcal{F}_x \tag{62}$$

$$M\frac{\partial v}{\partial t} = -Mfu - Mg\frac{\partial \zeta_w}{\partial y} + \tau_a^y + \tau_w^y + \mathcal{F}_y. \tag{63}$$

In this model, we neglect the nonlinear advection terms as well as the curvilinear terms in the internal ice stress. Nonlinear formulas are used for both the ocean-ice and air-ice surface stress:

$$\vec{\tau}_a = \rho_a C_a |\vec{V}_{10}| \vec{V}_{10} \tag{64}$$

$$C_a = \frac{1}{2} C_d \left[1 - \cos(2\pi \min(h_i + .1, .5)\right] \tag{65}$$

$$\vec{\tau}_w = \rho_w C_w |\vec{v}_w - \vec{v}| (\vec{v}_w - \vec{v}). \tag{66}$$

The force due to the internal ice stress is given by the divergence of the stress tensor σ. The rheology is given by the stress-strain relation of the medium. We would like to emulate the viscous-plastic rheology of Hibler (1979) [16]:

$$\sigma_{ij} = 2\eta\dot{\epsilon}_{ij} + (\zeta - \eta)\dot{\epsilon}_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij} \tag{67}$$

$$\dot{\epsilon}_{ij} \equiv \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{68}$$

$$P = P^* A h_i e^{-C(1-A)} \tag{69}$$

where the nonlinear viscosities are given by

$$\zeta = \frac{P}{2\left[(\epsilon_{11}^2 + \epsilon_{22}^2)(1 + 1/e^2) + 4e^{-2}\epsilon_{12}^2 + 2\epsilon_{11}\epsilon_{22}(1 - 1/e^2)\right]^{1/2}} \tag{70}$$

$$\eta = \frac{\zeta}{e^2}. \tag{71}$$

We would also like to have an explicit model that can be solved efficiently on parallel computers. The EVP rheology has a tunable coefficient E (the Young's modulus) which can be chosen to make the elastic term small compared to the other terms. We rearrange the VP rheology:

$$\frac{1}{2\eta}\sigma_{ij} + \frac{\eta - \zeta}{4\eta\zeta}\sigma_{kk}\delta_{ij} + \frac{P}{4\zeta}\delta_{ij} = \dot{\epsilon}_{ij} \tag{72}$$

then add the elastic term:

$$\frac{1}{E}\frac{\partial \sigma_{ij}}{\partial t} + \frac{1}{2\eta}\sigma_{ij} + \frac{\eta - \zeta}{4\eta\zeta}\sigma_{kk}\delta_{ij} + \frac{P}{4\zeta}\delta_{ij} = \dot{\epsilon}_{ij} \tag{73}$$

Much like the ocean model, the ice model has a split timestep. The internal ice stress term is updated on a shorter timestep so as to allow the elastic wave velocity to be resolved.

Once the new ice velocities are computed, the ice tracers can be advected using the MPDATA scheme [**?**]. The tracers in this case are the ice thickness, ice concentration, snow thickness, internal ice temperature, and surface melt ponds. The continuity equations describing the evolution of these parameters (equations (74)–(76)) also include thermodynamic terms (S_h, S_s and S_A), which will be described in §4.5:

$$\frac{\partial A h_i}{\partial t} = -\frac{\partial (u A h_i)}{\partial x} - \frac{\partial (v A h_i)}{\partial y} + S_h + \mathcal{D}_h \tag{74}$$

$$\frac{\partial A h_s}{\partial t} = -\frac{\partial (u A h_s)}{\partial x} - \frac{\partial (v A h_s)}{\partial y} + S_s + \mathcal{D}_s \tag{75}$$

$$\frac{\partial A}{\partial t} = -\frac{\partial (u A)}{\partial x} - \frac{\partial (v A)}{\partial y} + S_A + \mathcal{D}_A \qquad 0 \le A \le 1. \tag{76}$$

The first two equations represent the conservation of ice and snow. Equation 76 is discussed in some detail in MK89, but represents the advection of ice blocks in which no ridging occurs as long as there is any open water. An optional ridging term can be added (Gray and Killworth [9]):

$$\frac{\partial A}{\partial t} = -\frac{\partial (u A)}{\partial x} - \frac{\partial (v A)}{\partial y} - A\alpha(A)\,\nabla \cdot \vec{v}\, H(-\nabla \cdot \vec{v}) + S_A + \mathcal{D}_A \qquad 0 \le A \le 1. \tag{77}$$

where $\alpha(A)$ is an arbitrary function such that $\alpha(0) = 0$, $\alpha(1) = 1$, and $0 \le \alpha(A) \le 1$. The ridging term leads to an increase in h_i under convergent flow as would be produced by ridging. The function $\alpha(A)$ should be chosen so that it is near zero until the ice concentration is large enough that ridging is expected to occur, then should increase smoothly to one.

The symbols used in these equations along with the values for the constants are listed in Table 4.

4.2 Horizontal curvilinear coordinates

Applying the curvilinear transformation used in §2.5 and described in Appendix C, we use a transformation to an orthogonal curvilinear coordinate system. Denoting the velocity components in the new coordinate system by

$$\vec{v} \cdot \hat{\xi} = u \tag{78}$$

and

$$\vec{v} \cdot \hat{\eta} = v \tag{79}$$

the equations of motion (62), (63), (74)–(76) can be re-written (see, e.g., Arakawa and Lamb [1]):

$$\frac{\partial u}{\partial t} = fv - gm\frac{\partial \zeta_w}{\partial \xi} + \frac{1}{M}\left(\tau_a^\xi + \tau_w^\xi + \mathcal{F}_\xi\right) \tag{80}$$

$$\frac{\partial v}{\partial t} = -fu - gn\frac{\partial \zeta_w}{\partial \eta} + \frac{1}{M}\left(\tau_a^\eta + \tau_w^\eta + \mathcal{F}_\eta\right) \tag{81}$$

$$\frac{\partial A h_i}{\partial t} = -mn\left[\frac{\partial}{\partial \xi}\left(\frac{A h_i u}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{A h_i v}{m}\right)\right] + S_h \tag{82}$$

Variable	Value	Description
$A(x,y,t)$		ice concentration
$\alpha(A)$		ridging function
C_a		nonlinear air drag coefficient
C_d	2.2×10^{-3}	air drag coefficient
C_w	10×10^{-3}	water drag coefficient
$(\mathcal{D}_h, \mathcal{D}_s, \mathcal{D}_A)$		diffusion terms
δ_{ij}		Kronecker delta function
E		Young's modulus
e	2	eccentricity of the elliptical yield curve
$\epsilon_{ij}(x,y,t)$		strain rate tensor
$\eta(x,y,t)$		nonlinear shear viscosity
$f(x,y)$		Coriolis parameter
$(\mathcal{F}_x, \mathcal{F}_y)$		internal ice stress
g	$9.8\, m\, s^{-2}$	acceleration of gravity
H		Heaviside function
$h_i(x,y,t)$		ice thickness of ice-covered fraction
h_o	1 m	ice cutoff thickness
$h_s(x,y,t)$		snow thickness on ice-covered fraction
$M(x,y,t)$		ice mass (density times thickness)
$P(x,y,t)$		ice pressure or strength
(P^*, C)	$(2.75 \times 10^4, 20)$	ice strength parameters
(S_h, S_s, S_A)		thermodynamic terms
$\sigma_{ij}(x,y,t)$		stress tensor
$\vec{\tau}_a$		air stress
$\vec{\tau}_w$		water stress
(u,v)		the (x,y) components of ice velocity $\vec{v}$
$(\vec{V}_{10}, \vec{v}_w)$		10 meter air and surface water velocities
(ρ_a, ρ_w)	$(1.3\, kg\, m^{-3}, 1025\, kg\, m^{-3})$	air and water densities
$\zeta(x,y,t)$		nonlinear bulk viscosity
$\zeta_w(x,y,t)$		height of the ocean surface

Table 4: Variables used in the ice momentum equations

$$\frac{\partial Ah_s}{\partial t} = -mn\left[\frac{\partial}{\partial \xi}\left(\frac{Ah_s u}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{Ah_s v}{m}\right)\right] + S_s \tag{83}$$

$$\frac{\partial A}{\partial t} = -mn\left[\frac{\partial}{\partial \xi}\left(\frac{Au}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{Av}{m}\right)\right] + S_A \tag{84}$$

S_h, S_s and S_A remain unchanged.

The viscous-plastic terms can be derived from equation (**??**). In curvilinear coordinates the strain rate tensor can be written as:

$$\epsilon_{11} = m\frac{\partial u}{\partial \xi} + vmn\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right) \tag{85}$$

$$\epsilon_{12} = e_{21} = \frac{1}{2}\left[\frac{m}{n}\frac{\partial\,(nv)}{\partial \xi} + \frac{n}{m}\frac{\partial\,(mu)}{\partial \eta}\right] \tag{86}$$

$$\epsilon_{22} = n\frac{\partial v}{\partial \eta} + umn\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) \tag{87}$$

In curvilinear coordinates the divergence of a symmetric tensor $\mathbf{T}$ is:

$$\begin{aligned}\nabla \cdot \mathbf{T} = \hat{\xi}\Bigg[&m\frac{\partial \mathbf{T}_{11}}{\partial \xi} + n\frac{\partial \mathbf{T}_{12}}{\partial \eta} + \mathbf{T}_{11}mn\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)\\ &+2\mathbf{T}_{12}mn\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right) - \mathbf{T}_{22}mn\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)\Bigg]\\ +\hat{\eta}\Bigg[&m\frac{\partial \mathbf{T}_{12}}{\partial \xi} + n\frac{\partial \mathbf{T}_{22}}{\partial \eta} - \mathbf{T}_{11}mn\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\\ &+2\mathbf{T}_{12}mn\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) + \mathbf{T}_{22}mn\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\Bigg]\end{aligned} \tag{88}$$

A more general expression for derivatives of tensors is given by Aris [2] in terms of Christoffel symbols.

The viscous-plastic terms become:

$$\begin{aligned}\mathcal{F}_\xi = &m\frac{\partial}{\partial \xi}\left[(\zeta-\eta)mn\frac{\partial}{\partial \xi}\left(\frac{u}{n}\right)\right] + m\frac{\partial}{\partial \xi}\left[(\zeta-\eta)mn\frac{\partial}{\partial \eta}\left(\frac{v}{m}\right)\right] - \frac{m}{2}\frac{\partial P}{\partial \xi}\\ &+2m\frac{\partial}{\partial \xi}\left[\eta m\frac{\partial u}{\partial \xi} + \eta vmn\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right] + n\frac{\partial}{\partial \eta}\left[\eta\frac{m}{n}\frac{\partial(nv)}{\partial \xi} + \eta\frac{n}{m}\frac{\partial(mu)}{\partial \eta}\right]\\ &+2\eta m^2 n\frac{\partial u}{\partial \xi}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) + 2\eta v m^2 n^2\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) + 2\eta m^2\frac{\partial(nv)}{\partial \xi}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\\ &+2\eta n^2\frac{\partial(mu)}{\partial \eta}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right) - 2\eta mn^2\frac{\partial v}{\partial \eta}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - 2\eta u m^2 n^2\left[\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)\right]^2\end{aligned} \tag{89}$$

$$\begin{aligned}\mathcal{F}_\eta = &n\frac{\partial}{\partial \eta}\left[(\zeta-\eta)mn\frac{\partial}{\partial \xi}\left(\frac{u}{n}\right)\right] + n\frac{\partial}{\partial \eta}\left[(\zeta-\eta)mn\frac{\partial}{\partial \eta}\left(\frac{v}{m}\right)\right] - \frac{n}{2}\frac{\partial P}{\partial \eta}\\ &+m\frac{\partial}{\partial \xi}\left[\eta\frac{m}{n}\frac{\partial(nv)}{\partial \xi} + \eta\frac{n}{m}\frac{\partial(mu)}{\partial \eta}\right] + 2n\frac{\partial}{\partial \eta}\left[\eta n\frac{\partial v}{\partial \eta} + \eta umn\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)\right]\\ &-2\eta m^2 n\frac{\partial u}{\partial \xi}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right) - 2\eta v m^2 n^2\left[\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\right]^2 + 2\eta m^2\frac{\partial(nv)}{\partial \xi}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)\\ &+2\eta n^2\frac{\partial(mu)}{\partial \eta}\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) + 2\eta mn^2\frac{\partial v}{\partial \eta}\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right) + 2\eta u m^2 n^2\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right)\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right)\end{aligned} \tag{90}$$

4.3 Numerical Scheme

A key feature of the numerical scheme is a staggered grid, known as the "Arakawa C grid", where the velocity is defined on the sides of a grid box and variables such as thickness and viscosity are defined at the center of the grid box, as shown in Fig. 1.

Because of the strong ice interaction (which in this model is dissipative in nature) the momentum equations are essentially parabolic in form and hence have few numerical instability problems over long-term integrations. (While there are few numerical problems, it should be emphasized that the dissipative ice interaction terms are highly nonlinear and can lead to unstable flow fields in the absence of water drag. Such a feature is a physical characteristic of plastic flow and not a numerical artifact). However, in principle it is possible for the ice interaction to be very small even though there may be a finite ice mass. Under this situation, the momentum advection terms could cause nonlinear instabilities. To ensure against such situations, Hibler put a lower limit on the bulk viscosity parameter (and hence indirectly shear viscosity as well). It is never allowed to drop below $1.0 \times 10^7\ kg/s$, a value which negligibly modifies the ice drift.

Nonlinear instabilities over long-term integrations can also arise from the nonlinear advection terms in the thickness continuity equations. To avoid this problem, the advection terms in equations (74)–(76) can be computed with a choice of either Smolarkiewicz's MPDATA ([41]) or third-order upwind (Leonard [22]). See §3.8 for a description of these schemes. It may be possible to omit the diffusivity from these equations if the forcing fields are sufficiently smooth.

4.4 Horizontal boundary conditions

As mentioned above, initial conditions at all points and ice velocities at the boundaries are thereafter required to initiate the integration of the system of equations forward in time. The most natural boundary condition is to take the ice velocity to be zero on the boundaries. This can be done either at a land boundary or at an ocean location where there is no ice. Note that the boundary condition does not affect the ice motion in such circumstances since in the absence of ice the strength is zero. More generally, as long as the ocean boundaries are removed from the ice edge, the coupled nature of the model will cause a natural ice edge boundary condition to be created. However, it is also possible to form an "open" boundary condition by setting the strength equal to zero near a boundary. These gridboxes are called "outflow cells" in Hibler [16]. In the Arctic simulations, these outflow cells are used at the open edge near Greenland.

The boundary conditions on the momentum equations are to set u and v to zero at all boundaries, including islands. This is accomplished by the elliptic solver during the implicit timestep.

The ice and snow thickness and ice concentration equations have no-flux boundary conditions imposed along the mask boundaries. The outflow cells contain a radiation condition if the velocity is outward and no change if the velocity is inward.

The primary characteristic of the outflow cells is that the ice strength goes to zero there. The values of P, ζ, and η are all set to zero in outflow cells.

4.5 Thermodynamics

The thermodynamics used is based on the algorithm described in Mellor and Kantha [26], who have a useful description of the various melting and freezing processes, plus the coupling to a full three-dimensional ocean. Their form of equations (74) and (76) is:

$$\frac{\partial Ah_i}{\partial t} + \frac{\partial (uAh_i)}{\partial x} + \frac{\partial (vAh_i)}{\partial y} = \frac{\rho_o}{\rho_i}\left[A(W_{io} - W_{ai}) + (1-A)W_{ao} + W_{fr}\right] \tag{91}$$

$$\frac{\partial A}{\partial t} + \frac{\partial (uA)}{\partial x} + \frac{\partial (vA)}{\partial y} = \frac{\rho_o A}{\rho_i h_i}\left[\Phi(1-A)W_{ao} + (1-A)W_{fr}\right] \qquad 0 \le A \le 1. \tag{92}$$

Here, the W variables are the freeze or melt rates as shown in Fig. 7 and Table 5. The frazil ice growth W_{fr} will be discussed further in §4.5.2—note that it contributes to changes in A as well as to changes in h_i. The other term that contributes to A is W_{ao}. This term includes a factor Φ which Mellor and Kantha set to different values depending on whether ice is melting or freezing:

$$\Phi = 4.0 \qquad W_{ao} \geq 0 \tag{93}$$

$$\Phi = 0.5 \qquad W_{ao} < 0 \tag{94}$$

$$\tag{95}$$

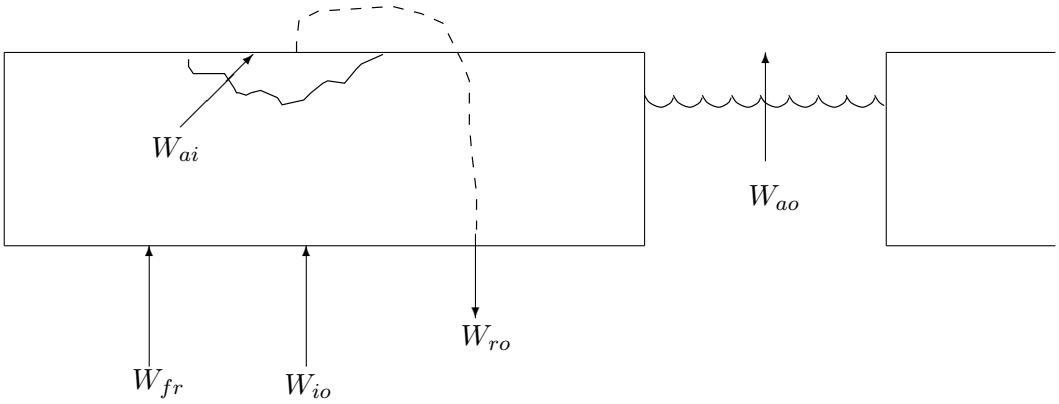

Figure 7: Diagram of the different locations where ice melting and freezing can occur.

Figure 8 shows the locations of the ice and snow temperatures and the heat fluxes. The temperature profile is assumed to be linear between adjacent temperature points. The interior of the ice contains "brine pockets", leading to a prognostic equation for the temperature T_1.

The surface flux to the air is:

$$Q_{ai} = -H\downarrow - LE\downarrow - \epsilon_s LW\downarrow - (1-\alpha_s)SW\downarrow + \epsilon_s \sigma (T_3 + 273)^4 \tag{96}$$

The formulas for sensible heat, latent heat, and incoming longwave and shortwave radiations are the same as in Parkinson and Washington [29] and are shown in Appendix E. The sensible heat is a function of T_3, as is the heat flux through the snow Q_s. Setting $Q_{ai} = Q_s$, we can solve for T_3 by setting $T_3^{n+1} = T_3^n + \Delta T_3$ and linearizing in ΔT_3. The temperature T_3 is found by an iterative solution of the surface heat flux balance (using the previous value of T_1 in equation 104). As in

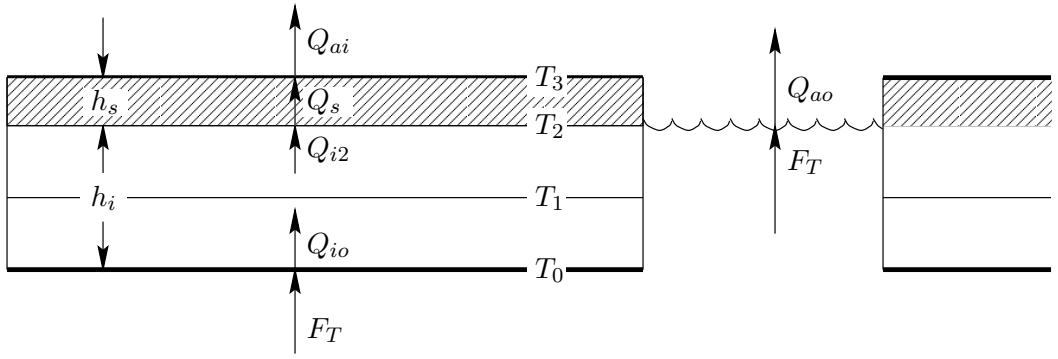

Figure 8: Diagram of internal ice temperatures and fluxes. The hashed layer is the snow.

Variable	Value	Description
α_w	0.10	shortwave albedo of water
α_i	0.60	shortwave albedo of wet ice
α_i	0.65	shortwave albedo of dry ice
α_s	0.72	shortwave albedo of wet snow
α_s	0.85	shortwave albedo of dry snow
C_k		snow correction factor
C_{pi}	2093 J kg^{-1} K^{-1}	specific heat of ice
C_{po}	3990 J kg^{-1} K^{-1}	specific heat of water
ϵ_w	0.97	longwave emissivity of water
ϵ_i	0.97	longwave emissivity of ice
ϵ_s	0.99	longwave emissivity of snow
$E(T, r)$		enthalpy of the ice/brine system
$F_T\uparrow$		heat flux from the ocean into the ice
$H\downarrow$		sensible heat
i_w		fraction of the solar heating transmitted through a lead into the water below
k_i	2.04 W m^{-1} K^{-}1	thermal conductivity of ice
k_s	0.31 W m^{-1} K^{-}1	thermal conductivity of snow
L_i	302 MJ m^{-3}	latent heat of fusion of ice
L_s	110 MJ m^{-3}	latent heat of fusion of snow
$LE\downarrow$		latent heat
$LW\downarrow$		incoming longwave radiation
m	-0.054°C/PSU	coefficient in linear $T_f(S) = mS$ equation
Φ		contribution to A equation from freezing water
Q_{ai}		heat flux out of the snow/ice surface
Q_{ao}		heat flux out of the ocean surface
Q_{i2}		heat flux up out of the ice
Q_{io}		heat flux up into the ice
Q_s		heat flux up through the snow
r		brine fraction in ice
ρ_i	910 m^3/kg	density of ice
S_i	5 PSU	salinity of the ice
$SW\downarrow$		incoming shortwave radiation
σ	5.67×10^{-8} W m^{-2} K^{-4}	Stefan-Boltzmann constant
T_0		temperature of the bottom of the ice
T_1		temperature of the interior of the ice
T_2		temperature at the upper surface of the ice
T_3		temperature at the upper surface of the snow
T_f		freezing temperature
$T_{\text{melt\_}i}$	mS_i	melting temperature of ice
$T_{\text{melt\_}s}$	0° C	melting temperature of snow
W_{ai}		melt rate on the upper ice/snow surface
W_{ao}		freeze rate at the air/water interface
W_{fr}		rate of frazil ice growth
W_{io}		freeze rate at the ice/water interface
W_{ro}	W_{ai}	rate of run-off of surface melt water

Table 5: Variables used in the ice thermodynamics

Parkinson and Washington, if T_3 is found to be above the melting temperature, it is set to $T_{\rm melt}$ and the extra energy goes into melting the snow or ice:

$$W_{ai} = \frac{Q_{ai} - Q_{i2}}{\rho_o L_3} \tag{97}$$

$$L_3 \equiv [E(T_3, 1) - E(T_1, R_1)] \tag{98}$$

Note that $L_3 = (1 - r)L_i$ plus a small sensible heat correction. We are not storing water on the surface in melt pools, so everything melted at the surface is assumed to flow into the ocean ($W_{ro} = W_{ai}$).

Inside the ice there are brine pockets in which there is salt water at the *in situ* freezing temperature. It is assumed that the ice has a uniform overall salinity of S_i and that the freezing temperature is a linear function of salinity. The brine fraction r is given by

$$r = \frac{S_i m}{T_1}$$

The enthalpy of the combined ice/brine system is given by

$$E(T, r) = r(L_i + C_{po}T) + (1 - r)C_{pi}T \tag{99}$$

Substituting in for r and differentiating gives:

$$\frac{\partial E}{\partial T} = -\frac{S_i m L_i}{T_1^2} + C_{pi} \tag{100}$$

Inside the snow, we have

$$Q_s = \frac{k_s}{h_s}(T_2 - T_3) \tag{101}$$

The heat conduction in the upper part of the ice layer is

$$Q_{I2} = \frac{2k_i}{h_i}(T_1 - T_2) \tag{102}$$

These can be set equal to each other to solve for T_2

$$T_2 = \frac{T_3 + C_k T_1}{1 + C_k} \tag{103}$$

where

$$C_k \equiv \frac{2k_i h_s}{h_i k_s}.$$

Substituting into (102), we get:

$$Q_s = Q_{I2} = \frac{2k_i}{h_i}\frac{(T_1 - T_3)}{(1 + C_k)} \tag{104}$$

Note that in the absence of snow, C_k becomes zero and we recover the formula for the no-snow case in which $T_3 = T_2$.

At the bottom of the ice, we have

$$Q_{I0} = \frac{2k_i}{h_i}(T_0 - T_1) \tag{105}$$

The difference between Q_{I0} and Q_{I2} goes into the enthalpy of the ice:

$$\rho_i h_i \left[\frac{\partial E}{\partial t} + \vec{v} \cdot \nabla E\right] = Q_{I0} - Q_{I2} \tag{106}$$

Variable	Value	Definition
b	3.0	factor
$\dot{\mathrm{E}}$		evaporation
k	0.4	von Karman's constant
ν	$1.8 \times 10^{-6} m^2 s^{-1}$	kinematic viscosity of seawater
$\dot{\mathrm{P}}$		precipitation
Pr	13.0	molecular Prandtl number
P_{rt}	0.85	turbulent Prandtl number
S_0		surface salinity
τ_{io}		stress on the ocean from the ice
τ_{ao}		stress on the ocean from the wind
T		internal ocean temperature
u_τ		friction velocity $\lvert\tau_{io}\rvert^{1/2}\rho_o^{-1/2}$
z_0		roughness parameter

Table 6: Ocean surface variables

We can use the chain rule to obtain an equation for timestepping T_1:

$$\rho_i h_i \frac{\partial E}{\partial T}\left[\frac{\partial T_1}{\partial t} + \vec{v}\cdot\nabla T_1\right] = Q_{I0} - Q_{I2} \tag{107}$$

where

$$\begin{aligned} Q_{I0} - Q_{I2} &= \frac{2k_i}{h_i}\left[(T_0 - T_1) - \frac{(T_1 - T_3)}{1 + C_k}\right] \\ &= \frac{2k_i}{h_i}\left[(T_0 + \frac{T_3 - (2 + C_k)T_1}{1 + C_k}\right] \end{aligned}$$

4.5.1 Ocean surface boundary conditions

The ocean receives surface stresses from both the atmosphere and the ice, according to the ice concentration:

$$K_m \frac{\partial u_w}{\partial z} = \frac{A}{\rho_o}\tau_{io}^x + \frac{1 - A}{\rho_o}\tau_{ao}^x \tag{108}$$

$$K_m \frac{\partial v_w}{\partial z} = \frac{A}{\rho_o}\tau_{io}^y + \frac{1 - A}{\rho_o}\tau_{ao}^y \tag{109}$$

where the relevant variables are in table 6.

The surface ocean is assumed to be at the freezing temperature for the surface salinity ($T_0 = mS$) where we use the salinity from the uppermost model point at $z = -\frac{1}{2}\Delta z$. From this, we can obtain a vertical temperature gradient for the upper ocean to use in the heat flux formula:

$$\frac{F_T}{\rho_o C_p o} = -C_{T_z}(T_0 - T) \qquad z \to 0 \tag{110}$$

where

$$C_{T_z} = \frac{u_\tau}{P_{rt}k^{-1}\ln(-z/z_0) + B_T} \tag{111}$$

$$B_T = b\left(\frac{z_0 u_\tau}{\nu}\right)^{1/2} Pr^{2/3} \tag{112}$$

Variable	Value	Definition
C_{pi}	1994 J kg^{-1} K^{-1}	specific heat of ice
C_{pw}	3987 J kg^{-1} K^{-1}	specific heat of water
γ	m_i/m_{w_2}	fraction of water that froze
L	3.16e5 J kg^{-1}	latent heat of fusion
m_i		mass of ice formed
m_{w_1}		mass of water before freezing
m_{w_2}		mass of water after freezing
m	-0.0543	constant in freezing equation
n	7.59×10^{-4}	constant in freezing equation
S_1		salinity before freezing
S_2		salinity after freezing
T_1		temperature before freezing
T_2		temperature after freezing

Table 7: Frazil ice variables

Once we have a the value for F_T, we can use it to find the ice growth rates:

$$W_{io} = \frac{1}{\rho_o L_o}(Q_{io} - F_T) \tag{113}$$

$$W_{ao} = \frac{1}{\rho_o L_o}(Q_{ao} - F_T) \tag{114}$$

$$\tag{115}$$

where

$$L_o \equiv [E(T_0, 1) - E(T_1, r_1)] \tag{116}$$

The ocean model receives the following heat and salt fluxes:

$$F_T = AQ_{io} + (1 - A)Q_{ao} - W_o L_o \tag{117}$$

$$F_S = (W_o - AW_{ro})(S_i - S_0) + (1 - A)S_o(\dot{\mathrm{P}} - \dot{\mathrm{E}})W_o \qquad \equiv AW_{io} + (1 - A)W_{ao} \tag{118}$$

4.5.2 Frazil ice formation

Following Steele et al. [43], we check to see if any of the ocean temperatures are below freezing at the end of each timestep. If so, frazil ice is assumed to form, changing the local temperature and salinity. The ice that forms is assumed to instantly float up to the surface and add to the ice layer there. We assume balances in the mass, heat, and salt before and after the ice is formed:

$$m_{w_1} = m_{w_2} + m_i \tag{119}$$

$$m_{w_1}(C_{pw}T_1 + L) = m_{w_2}(C_{pw}T_2 + L) + m_i C_{pi} T_2 \tag{120}$$

$$m_{w_1} S_1 = m_{w_2} S_2. \tag{121}$$

The variables are defined in Table 7. Defining $\gamma = m_i/m_{w_2}$ and dropping terms of order γ^2 leads to:

$$T_2 = T_1 + \gamma \left[\frac{L}{C_{pw}} + T_1 \left(1 - \frac{C_{pi}}{C_{pw}} \right) \right] \tag{122}$$

$$S_2 = S_1(1 + \gamma). \tag{123}$$

We also want the final temperature and salinity to be on the freezing line, which we approximate as:

$$T_f = mS + nz. \tag{124}$$

We can then solve for γ:

$$\gamma = \frac{-T_1 + mS_1 + nz}{\frac{L}{C_{pw}} + T_1\left(1 - \frac{C_{pi}}{C_{pw}}\right) - mS_1}. \tag{125}$$

The ocean is checked at each depth k and at each timestep for supercooling. If the water is below freezing, the temperature and salinity are adjusted as in equations (122) and (123) and the ice above is thickened by the amount:

$$\Delta h = \gamma_k \Delta z_k \frac{\rho_w}{\rho_i}. \tag{126}$$

4.5.3 Differences from Mellor and Kantha

We have tried to modify the **hakkis** model to more closely follow MK89. However, there are also ways in which we have deviated from it.

- Add advection of snow.
- Add lateral melting of snow when ice is melting laterally.
- We iterate on the solution of T_3.
- We took a shortcut in the solution of S_0, T_0 for the surface heat and salt fluxes. We also apply them differently to the ocean model.
- We added various limiters:
  - Ice concentration:$A \geq A_{\min}$, $A_{\min} = 0.02$.
  - Ice thickness: $h_i \geq h_{\min}$, $h_{\min} = 0.1$.
  - Brine fraction: $r \leq r_{\max}$, $r_{\max} = 0.2$

A Model Time-stepping Schemes

Numerical time stepping uses a discrete approximation to:

$$\frac{\partial \phi(t)}{\partial t} = \mathcal{F}(t) \tag{127}$$

where ϕ represents one of u, v, C, or ζ, and $\mathcal{F}(t)$ represents all the right-hand-side terms. In ROMS, the goal is to find time-stepping schemes which are accurate where they are valid and damping on unresolved signals ([36]). Also, the preference is for time-stepping schemes requiring only one set of the right-hand-side terms so that different time-stepping schemes can be used for different terms in the equations. Finally, as mentioned in Table 3.3, not all versions of ROMS use the same time-stepping algorithm. We list some time-stepping schemes here which are used or have been used by the ROMS/SCRUM family of models, plus a few to help explain some of the more esoteric ones.

A.1 Euler

The simplest approximation is the Euler time step:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \mathcal{F}(t) \tag{128}$$

where you predict the next ϕ value based only on the current fields. This method is accurate to first order in Δt; however, it is unconditionally unstable with respect to advection.

A.2 Leapfrog

The leapfrog time step is accurate to $\mathrm{O}(\Delta t^2)$:

$$\frac{\phi(t+\Delta t) - \phi(t-\Delta t)}{2\Delta t} = \mathcal{F}(t). \tag{129}$$

This time step is more accurate, but it is unconditionally unstable with respect to diffusion. Also, the even and odd time steps tend to diverge in a computational mode. This computational mode can be damped by taking correction steps. SCRUM's time step on the depth-integrated equations was a leapfrog step with a trapezoidal correction (LF-TR) on every step, which uses a leapfrog step to obtain an initial guess of $\phi(t+\Delta t)$. We will call the right-hand-side terms calculated from this initial guess $\mathcal{F}^*(t+\Delta t)$:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \frac{1}{2}\left[\mathcal{F}(t) + \mathcal{F}^*(t+\Delta t)\right]. \tag{130}$$

This leapfrog-trapezoidal time step is stable with respect to diffusion and it strongly damps the computational mode. However, the right-hand-side terms are computed twice per time step.

A.3 Third-order Adams-Bashforth (AB3)

The time step on SCRUM's full 3-D fields is done with a third-order Adams-Bashforth step. It uses three time-levels of the right-hand-side terms:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \alpha\mathcal{F}(t) + \beta\mathcal{F}(t-\Delta t) + \gamma\mathcal{F}(t-2\Delta t) \tag{131}$$

where the coefficients α, β and γ are chosen to obtain a third-order estimate of $\phi(t+\Delta t)$. We use a Taylor series expansion:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \phi' + \frac{\Delta t}{2}\phi'' + \frac{\Delta t^2}{6}\phi''' + \cdots \tag{132}$$

where

$$\begin{aligned}\mathcal{F}(t) &= \phi' \\ \mathcal{F}(t-\Delta t) &= \phi' - \Delta t\phi'' + \frac{\Delta t^2}{2}\phi''' + \cdots \\ \mathcal{F}(t-2\Delta t) &= \phi' - 2\Delta t\phi'' + 2\Delta t^2\phi''' + \cdots\end{aligned}$$

We find that the coefficients are:

$$\alpha = \frac{23}{12}, \qquad \beta = -\frac{4}{3}, \qquad \gamma = \frac{5}{12}$$

This requires one time level for the physical fields and three time levels of the right-hand-side information and requires special treatment on startup.

A.4 Forward-Backward

In equation 127 above, we assume that multiple equations for any number of variables are time stepped synchronously. For coupled equations, we can actually do better by time stepping asynchronously. Consider these equations:

$$\begin{aligned}\frac{\partial \zeta}{\partial t} &= \mathcal{F}(u) \\ \frac{\partial u}{\partial t} &= \mathcal{G}(\zeta)\end{aligned} \tag{133}$$

If we time step them alternately, we can always be using the newest information:

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \mathcal{F}(u^n)\Delta t \\ u^{n+1} &= u^n + \mathcal{G}(\zeta^{n+1})\Delta t\end{aligned} \tag{134}$$

This scheme is second-order accurate and is stable for longer time steps than many other schemes. It is however unstable for advection terms.

A.5 Forward-Backward Feedback (RK2-FB)

One option for solving equation 133 is a predictor-corrector with predictor step:

$$\begin{aligned}\zeta^{n+1,\star} &= \zeta^n + \mathcal{F}(u^n)\Delta t \\ u^{n+1,\star} &= u^n + \left[\beta\mathcal{G}(\zeta^{n+1,\star}) + (1-\beta)\mathcal{G}(\zeta^n)\right]\Delta t\end{aligned} \tag{135}$$

and corrector step:

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \frac{1}{2}\left[\mathcal{F}(u^{n+1,\star}) + \mathcal{F}(u^n)\right]\Delta t \\ u^{n+1} &= u^n + \frac{1}{2}\left[\epsilon\mathcal{G}(\zeta^{n+1}) + (1-\epsilon)\mathcal{G}(\zeta^{n+1,\star}) + \mathcal{G}(\zeta^n)\right]\Delta t\end{aligned} \tag{136}$$

Setting $\beta = \epsilon = 0$ in the above, it becomes a standard second order Runge-Kutta scheme, which is unstable for a non-dissipative system. Adding the β and ϵ terms adds Forward-Backward feedback to this algorithm, and allows us to improve both its accuracy and stability. The choice of $\beta = 1/3$ and $\epsilon = 2/3$ leads to a stable third-order scheme with $\alpha_{\max} = 2.14093$ ([36]).

A.6 LF-TR and LF-AM3 with FB Feedback

Another possibility is a leapfrog predictor:

$$\begin{aligned}\zeta^{n+1,\star} &= \zeta^{n-1} + 2\mathcal{F}(u^n)\Delta t \\ u^{n+1,\star} &= u^{n-1} + 2\left\{(1-2\beta)\mathcal{G}(\zeta^n) + \beta\left[\mathcal{G}(\zeta^{n+1,\star}) + \mathcal{G}(\zeta^{n-1})\right]\right\}\Delta t\end{aligned} \tag{137}$$

and either a two-time trapezoidal or a three-time Adams-Moulton corrector:

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \left[\left(\frac{1}{2}-\gamma\right)\mathcal{F}(u^{n+1,\star}) + \left(\frac{1}{2}+2\gamma\right)\mathcal{F}(u^n) - \gamma\mathcal{F}(u^{n-1})\right]\Delta t \\ u^{n+1} &= u^n + \left\{\left(\frac{1}{2}-\gamma\right)\left[\epsilon\mathcal{G}(\zeta^{n+1}) + (1-\epsilon)\mathcal{G}(\zeta^{n+1,\star})\right] + \left(\frac{1}{2}+2\gamma\right)\mathcal{G}(\zeta^n) - \gamma\mathcal{G}(\zeta^{n-1})\right\}\Delta t\end{aligned} \tag{138}$$

where the parameters β and ϵ introduce FB-feedback during both stages, while γ controls the type of corrector scheme. Without FB-feedback, we have:

$$\beta = \epsilon = 0 \Rightarrow \left\{\begin{array}{lcll} \gamma = 0 & \Rightarrow & \text{LF-TR} & \alpha_{\max} = \sqrt{2} \\ \gamma = 1/12 & \Rightarrow & \text{LF-AM3} & \alpha_{\max} = 1.5874 \\ \gamma = 0.0804 & \Rightarrow & \text{max stability} & \alpha_{\max} = 1.5876 \end{array}\right.$$

Keeping $\gamma = 1/12$ maintains third-order accuracy. The most accurate choices for β and ϵ are $\beta = 17/120$ and $\epsilon = 11/20$, which yields a fourth-order scheme with low numerical dispersion and diffusion and $\alpha_{\max} = 1.851640$ ([36]).

A.7 Generalized FB with an AB3-AM4 Step

One drawback of the previous two schemes is the need to evaluate the right-hand-side (r.h.s.) terms twice per time step. The original forward-backward scheme manages to achieve $\alpha_{\max} = 2$ while only evaluating each r.h.s. term once. It is possible to contruct a robust scheme using a combination of a three-time AB3-like step for ζ with a four-time AM4-like step for u:

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \left[\left(\frac{3}{2}+\beta\right)\mathcal{F}(u^n) - \left(\frac{1}{2}+2\beta\right)\mathcal{F}(u^{n-1}) + \beta\mathcal{F}(u^{n-2})\right]\Delta t \\ u^{n+1} &= u^n + \left[\left(\frac{1}{2}+\gamma+2\epsilon\right)\mathcal{G}(\zeta^{n+1}) + \left(\frac{1}{2}-2\gamma-3\epsilon\right)\mathcal{G}(\zeta^n) + \gamma\mathcal{G}(\zeta^{n-1}) + \epsilon\mathcal{G}(\zeta^{n-2})\right]\Delta t\end{aligned} \tag{139}$$

This is second-order accurate for any set of values for β, γ, and ϵ. It is third-order accurate if $\beta = 5/12$. However, it has a wider stability range for $\beta = 0.281105$. Shchepetkin and McWilliams ([36]) choose to use this scheme for the barotropic mode in their model with $\beta = 0.281105$, $\gamma = 0.0880$, and $\epsilon = 0.013$, to obtain $\alpha_{\max} = 1.7802$, close to the value of 2 for pure FB, but with better stability properties for the combination of waves, advection, and Coriolis terms.

B The vertical σ-coordinate

Following Song and Haidvogel [42] but modified by Shchepetkin and McWilliams [35], the vertical coordinate has been chosen to be:

$$z = \zeta(1+\sigma) + h_c\sigma + (h-h_c)C(\sigma), \qquad -1 \le \sigma \le 0 \tag{140}$$

where h_c is either the minimum depth or a shallower depth above which we wish to have more resolution. $C(\sigma)$ is defined as:

$$C(\sigma) = (1-b)\frac{\sinh(\theta\sigma)}{\sinh\theta} + b\frac{\tanh[\theta(\sigma+\frac{1}{2})] - \tanh(\frac{1}{2}\theta)}{2\tanh(\frac{1}{2}\theta)} \tag{141}$$

where θ and b are surface and bottom control parameters. Their ranges are $0 < \theta \le 20$ and $0 \le b \le 1$, respectively. Equation (140) leads to $z = \zeta$ for $\sigma = 0$ and $z = h$ for $\sigma = -1$.

Some features of this coordinate system:

- It is a generalization of the traditional σ-coordinate system. Letting θ go to zero and using L'Hopital's rule, we get:
$$z = (\zeta + h)(1+\sigma) - h \tag{142}$$
which is the classic σ-coordinate.
- It is infinitely differentiable in σ.
- The larger the value of θ, the more resolution is kept above h_c.
- For $b = 0$, the resolution all goes to the surface as θ is increased.
- For $b = 1$, the resolution goes to both the surface and the bottom equally as θ is increased.
- Some problems turn out to be sensitive to the value of θ used.

Figure 9 shows the σ-surfaces for several values of θ and b for one of our domains. It was produced by a Matlab tool written by Hernan Arango which is available from our web site (see §**??**).

We find it convenient to define:

$$H_z \equiv \frac{\partial z}{\partial \sigma} = (\zeta + h) + (h - h_c)\frac{\partial C(\sigma)}{\partial \sigma}. \tag{143}$$

The derivative of $C(\sigma)$ can be computed analytically:

$$\frac{\partial C(\sigma)}{\partial \sigma} = (1-b)\frac{\cosh(\theta\sigma)}{\sinh\theta}\theta + b\frac{\coth(\frac{1}{2}\theta)}{2\cosh^2[\theta(\sigma+\frac{1}{2})]}\theta. \tag{144}$$

However, we choose to compute H_z discretely as $\Delta z/\Delta\sigma$ since this leads to the vertical sum of H_z being exactly the total water depth D.

Note that though we have used this form of σ-coordinate, ROMS is written in such a way as to work with a variety of vertical mappings. There is one feature which is critical, however. If the free surface is at rest, $\zeta = 0$, you get one solution for the level depths $z^{(0)}(k)$. In the case of nonzero ζ, the displacements must be proportional to ζ and to the original distance from the bottom:

$$z(k) = z^{(0)}(k) + \zeta\left(1 + \frac{z^{(0)}(k)}{h}\right) \tag{145}$$

or

$$\Delta z(k) = \Delta z^{(0)}(k)\left(1 + \frac{\zeta}{h}\right) \tag{146}$$

This ensures that the vertical mass fluxes generated by a purely barotropic motion will vanish at every interface.

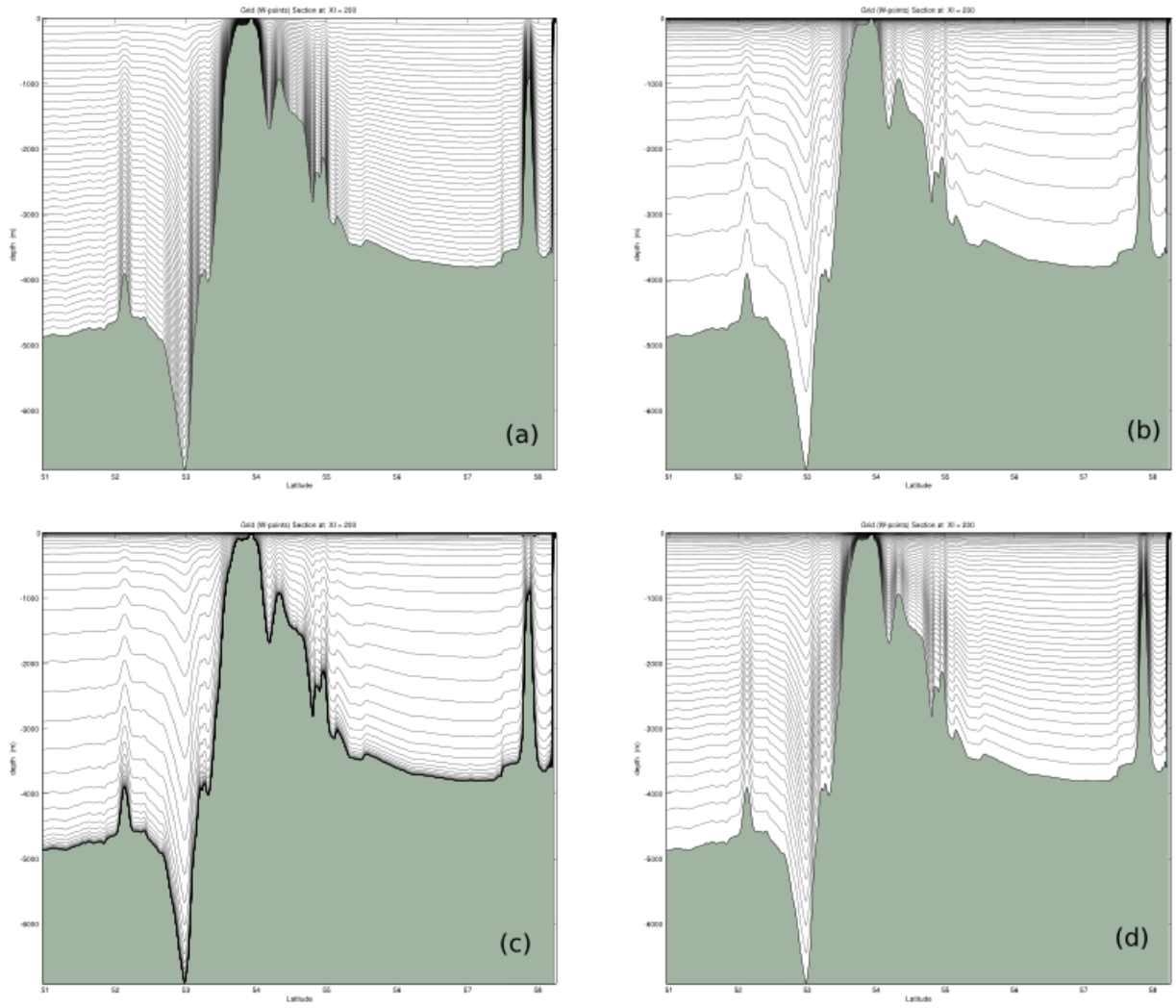

Figure 9: The σ-surfaces for the North Atlantic with (a) $\theta = 0.0001$ and $b = 0$, (b) $\theta = 8$ and $b = 0$, (c) $\theta = 8$ and $b = 1$. (d) The actual values used in this domain were $\theta = 5$ and $b = 0.4$.

C Horizontal curvilinear coordinates

The requirement for a boundary-following coordinate system and for a laterally variable grid resolution can both be met (for suitably smooth domains) by introducing an appropriate orthogonal coordinate transformation in the horizontal. Let the new coordinates be $\xi(x, y)$ and $\eta(x, y)$ where the relationship of horizontal arc length to the differential distance is given by:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi \tag{147}$$

$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta \tag{148}$$

Here, $m(\xi, \eta)$ and $n(\xi, \eta)$ are the scale factors which relate the differential distances $(\Delta\xi, \Delta\eta)$ to the actual (physical) arc lengths.

It is helpful to write the equations in vector notation and to use the formulas for div, grad, and curl in curvilinear coordinates (see Batchelor, Appendix 2, [3]):

$$\nabla\phi = \hat{\xi} m \frac{\partial\phi}{\partial\xi} + \hat{\eta} n \frac{\partial\phi}{\partial\eta} \tag{149}$$

$$\nabla \cdot \vec{a} = mn \left[\frac{\partial}{\partial\xi}\left(\frac{a}{n}\right) + \frac{\partial}{\partial\eta}\left(\frac{b}{m}\right) \right] \tag{150}$$

$$\nabla \times \vec{a} = mn \begin{vmatrix} \frac{\hat{\xi}_1}{m} & \frac{\hat{\xi}_2}{n} & \hat{k} \\ \frac{\partial}{\partial\xi} & \frac{\partial}{\partial\eta} & \frac{\partial}{\partial z} \\ \frac{a}{m} & \frac{b}{n} & c \end{vmatrix} \tag{151}$$

$$\nabla^2\phi = \nabla \cdot \nabla\phi = mn \left[\frac{\partial}{\partial\xi}\left(\frac{m}{n}\frac{\partial\phi}{\partial\xi}\right) + \frac{\partial}{\partial\eta}\left(\frac{n}{m}\frac{\partial\phi}{\partial\eta}\right) \right] \tag{152}$$

where ϕ is a scalar and $\vec{a}$ is a vector with components a, b, and c.

D Viscosity and Diffusion

D.1 Horizontal viscosity

The horizontal viscosity and diffusion coefficients are scalars which are read in from **ocean.in**. Several factors to consider when choosing these values are:

spindown time The spindown time on wavenumber k is $\frac{1}{k^2\nu_2}$ for the Laplacian operator and $\frac{1}{k^4\nu_4}$ for the biharmonic operator. The smallest wavenumber corresponds to the length $2\Delta x$ and is $k = \frac{\pi}{\Delta x}$, leading to

$$\Delta t < t_{damp} = \frac{\Delta x^2}{\pi^2\nu_2} \quad \text{or} \quad \frac{\Delta x^4}{\pi^4\nu_4} \tag{153}$$

This time should be short enough to damp out the numerical noise which is being generated but long enough on the larger scales to retain the features you are interested in. This time should also be resolved by the model timestep.

boundary layer thickness The western boundary layer has a thickness proportional to

$$\Delta x < L_{BL} = \left(\frac{\nu_2}{\beta}\right)^{\frac{1}{3}} \quad \text{and} \quad \left(\frac{\nu_4}{\beta}\right)^{\frac{1}{5}} \tag{154}$$

for the Laplacian and biharmonic viscosity, respectively. We have found that the model typically requires the boundary layer to be resolved with at least one grid cell. This leads to coarse grids requiring large values of ν.

D.2 Horizontal Diffusion

We have chosen anything from zero to the value of the horizontal viscosity for the horizontal diffusion coefficient. One common choice is an order of magnitude smaller than the viscosity.

D.3 Vertical Viscosity and Diffusion

ROMS stores the vertical mixing coefficients in arrays with three spatial dimensions called **Akv** and **Akt**. **Akt** also has a fourth dimension specifying which tracer, so that temperature and salt can have differing values. Both **Akt** and **Akv** are stored at w-points in the model; horizontal averaging is done to obtain **Akv** at the horizontal u and v-points. The values for these coefficients can be set in a number of ways, as described in §**??**.

E Radiant heat fluxes

As was seen in §4.5, the model thermodynamics requires fluxes of latent and sensible heat and longwave and shortwave radiation. We follow the lead of Parkinson and Washington [29] in computing these terms.

E.1 Shortwave radiation

The Zillman equation for radiation under cloudless skies is:

$$Q_o = \frac{S\cos^2 Z}{(\cos Z + 2.7)e \times 10^{-5} + 1.085\cos Z + 0.10} \tag{155}$$

where the variables are as in Table 8. The cosine of the zenith angle is computed using the formula:

$$\cos Z = \sin\phi \sin\delta + \cos\phi\cos\delta\cos HA. \tag{156}$$

The declination is

$$\delta = 23.44^\circ \times \cos\left[(172 - \text{day of year}) \times 2\pi/365\right] \tag{157}$$

and the hour angle is

$$HA = (12\,\text{hours} - \text{solar time}) \times \pi/12. \tag{158}$$

The correction for cloudiness is given by

$$SW\downarrow = Q_o(1 - 0.6c^3). \tag{159}$$

An estimate of the cloud fraction c will be provided by Jennifer Francis ([6]).

E.2 Longwave radiation

The clear sky formula for incoming longwave radiation is given by:

$$F\downarrow = \sigma T_a^4 \left\{1 - 0.261\exp\left[-7.77 \times 10^{-4}(273 - T_a)^2\right]\right\} \tag{160}$$

while the cloud correction is given by:

$$LW\downarrow = (1 + 0.275c)\,F\downarrow\,. \tag{161}$$

E.3 Sensible heat

The sensible heat is given by the standard aerodynamic formula:

$$H\downarrow = \rho_a c_p C_H V_{wg}(T_a - T_{sfc}). \tag{162}$$

E.4 Latent heat

The latent heat depends on the vapor pressure and the saturation vapor pressure given by:

$$e = 611 \times 10^{a(T_d - 273.16)/(T_d - b)} \tag{163}$$

$$e_s = 611 \times 10^{a(T_{sfc} - 273.16)/(T_{sfc} - b)} \tag{164}$$

The vapor pressures are used to compute specific humidities according to:

$$q_{10\text{m}} = \frac{\epsilon e}{p - (1 - \epsilon)e} \tag{165}$$

$$q_s = \frac{\epsilon e_s}{p - (1 - \epsilon)e_s} \tag{166}$$

Variable	Value	Description
(a, b)	(9.5, 7.66)	vapor pressure constants over ice
(a, b)	(7.5, 35.86)	vapor pressure constants over water
c		cloud cover fraction
C_E	1.75×10^{-3}	transfer coefficient for latent heat
C_H	1.75×10^{-3}	transfer coefficient for sensible heat
c_p	1004 J kg^{-1} K^{-1}	specific heat of dry air
δ		declination
e		vapor pressure in pascals
e_s		saturation vapor pressure
ϵ	0.622	ratio of molecular weight of water to dry air
HA		hour angle
L	2.5×10^6 J kg^{-1}	latent heat of vaporization
L	2.834×10^6 J kg^{-1}	latent heat of sublimation
ϕ		latitude
Q_o		incoming radiation for cloudless skies
q_s		surface specific humidity
$q_{10\mathrm{m}}$		10 meter specific humidity
ρ_a		air density
S	1353 W m^{-2}	solar constant
σ	5.67×10^{-8} W m^{-2} K^{-4}	Stefan-Boltzmann constant
T_a		air temperature
T_d		dew point temperature
T_{sfc}		surface temperature of the water/ice/snow
V_{wg}		geostrophic wind speed
Z		solar zenith angle

Table 8: Variables used in computing the incoming radiation and latent and sensible heat

The latent heat is also given by a standard aerodynamic formula:

$$LE\downarrow = \rho_a L C_E V_{wg}(q_{10\mathrm{m}} - q_s). \tag{167}$$

Note that these need to be computed independently for the ice-covered and ice-free portions of each gridbox since the empirical factors a and b and the factor L differ depending on the surface type.

F The C preprocessor

The C preprocessor, **cpp**, is a standalone program which comes with most C compilers. On many UNIX systems it is not in the default path, but in **/lib** or in **/usr/lib**. If you do not have a C preprocessor then there are several versions available via anonymous ftp. For instance, **ftp.uu.net** has two in the **/published/oreilly/nutshell/imake** directory—I have built and used the one from Der Mouse on a Cray. I have put this one in pub/util/cpp.tar.gz on the ahab.rutgers.edu ftp site since it supports the **#elif** construct. One also comes with **gcc**, the **gnu** C compiler. If you build this compiler, **cpp** will have a path such as

```
/usr/local/lib/gcc-lib/sparc-sun-solaris2.5/2.7.2/cpp
```

where **sparc** is the architecture, **sun** is the manufacturer, **solaris2.5** is the operating system and version, and **2.7.2** is **gcc**'s version number.

This Appendix describes the C preprocessor as used in SCRUM with the Fortran language. A more complete description is given by Kernighan and Ritchie [20]. More practical advice on using **cpp** is given by Hazard [14].

F.1 File inclusion

Placing common blocks in smaller files, which are then included in each subroutine, is the easiest way to make sure that the common blocks are declared consistently. Many Fortran compilers support an include statement where the compiler replaces the line

```
include 'file.h'
```

with the contents of **file.h**; **file.h** is assumed to be in the current directory. The C preprocessor has an equivalent include statement:

```
#include "file.h"
```

We are using the C preprocessor style of include because many of the SCRUM include files are not pure Fortran and must be processed by **cpp**.

F.2 Macro substitution

A macro definition has the form

```
#define    name           replacement text
```

where **name** would be replaced with "replacement text" throughout the rest of the file. This is used in SCRUM as a reasonably portable way to get 64-bit precision:

```
#define  BIGREAL     real*8
```

It is customary to use uppercase for **cpp** macros—the C preprocessor is case sensitive.

It is also possible to define macros with arguments, as in

```
#define av2(a1,a2)        (.5 * ((a1) + (a2)))
```

although this is riskier than the equivalent statement function

```
av2(a1,a2) = .5 * (a1 + a2)
```

The statement function is preferable because it allows the compiler to do type checking and because you don't have to be as careful about using enough parentheses.

The third form of macro has no replacement text at all:

```
#define  MASKING
```

In this case, **MASKING** will evaluate to **true** in the conditional tests described below.

F.3 Conditional inclusion

It is possible to control which parts of the code are seen by the Fortran compiler by the use of **cpp**'s conditional inclusion. For example, the statements

```
#ifdef MASKING
# include "mask.h"
#endif /* MASKING */
        :
# ifdef MASKING
c
c  Apply Land/Sea mask: slipperiness.
c
      do j=1,M
        do i=2,Lm
          Uflux(i,j)=Uflux(i,j)*pmask(i,j)
        enddo
      enddo
# endif /* MASKING */
```

will not be in the Fortran source code if **MASKING** has not been defined. Likewise, **#ifndef** tests for a macro being undefined:

```
#ifndef RMDOCINC
c  rmask       Mask at RHO-points (0=Land, 1=Sea).
c  pmask       Slipperiness mask at PSI-points (0=Land, 1=Sea,
c                                                1-gamma2=boundary).
c  umask       Mask at U-points (0=Land, 1=Sea).
c  vmask       Mask at V-points (0=Land, 1=Sea).
c
c=======================================================================
#endif
```

There are also **#else** and **#elif** (else if) statements, although **#elif** is newer and is not supported by all versions of **cpp**. An example using **#else** and **#elif** is shown:

```
#if defined BASIN
      parameter (L=181, M=141, N=12, NT=1)
#elif defined CANYON_A
      parameter (L=66, M=49, N=10, NT=1)
#elif defined CANYON_B
      parameter (L=66, M=49, N=15, NT=1)
           :
#elif defined UPWELLING
      parameter (L=42, M=81, N=16, NT=2)
#else
      parameter (L=???, M=???, N=??, NT=?)
#endif
```

Actually, **#ifdef** is a restricted version of the more general test

```
#if     expression
```

where "expression" is a constant integer value. Zero evaluates to **false** and everything else is considered **true**. Compound expressions may be built using the C logical operators:

```
&&          logical and
||          logical or
!           logical not
```

These symbols would be used as in the following example:

```
#if defined CANYON_A || defined CANYON_B
      do j=0,M
        do i=0,L
          yc=c32000-c16000*(sin(pi*xr(i,j)/xl))**24
          h(i,j)=c20+p5*(hmax-c20)*(c1+tanh((yr(i,j)-yc)/c10000))
        enddo
      enddo
#endif
```

F.4 C comments

The C preprocessor will also delete C language comments starting with /* and ending with */ as in:

```
 #endif  /* MASKING */
```

When mixed with Fortran code, it is safer to use a Fortran comment.

F.5 Potential problems

The use of the C preprocessor is not entirely free of problems, but many can be worked around or avoided by using the Der Mouse version of **cpp**.

1. Apostrophes in Fortran comments. **cpp** does not know that it is in a comment and some versions will complain about unmatched apostrophes in the following:

   ```
   c  Some useful comment about Green's functions.
   ```

   The **gnu** version of **cpp** (which comes with **gcc**) has a **-traditional** option which makes it more appropriate for use with Fortran.

2. C++ comments. Some of the newer versions of **cpp** will remove C++ comments which go from '//' to the end of the line. Some perfectly reasonable Fortran lines contain two consecutive slashes, such as:

   ```
        common // var1, var2
    44  format(//)
   ```

   and the new Fortran 90 string concatenation:

   ```
        mystring = 'Hello, ' // 'World!'
   ```

3. Macro replacement. One feature of **cpp** is that you can define macros and have it perform replacements. The code:

```
#define REAL double precision
      REAL really_long_variable, second_long_variable
```

becomes

```
      double precision really_long_variable, second_long_variable
```

and you run the risk of creating lines which are longer than 72 characters in length.

Also, make sure that your macros will not be found anywhere else in the code. I used to use **#define DOUBLE** for double precision until it was pointed out to me that **DOUBLE PRECISION** is perfectly valid Fortran. The macro processor would turn this into **1 PRECISION** since something that is defined has a value of 1.

F.6 Modern Fortran

I started working on these ocean models before 1990, much less before Fortran 90 compilers were generally available. Fortran 90's **kind** feature would be a better way to handle the **BIGREAL** type declarations. On the other hand, Fortran 90 does not include conditional compilation. However, it is deemed useful enough that the Fortran 2000 committee has a draft document describing how Fortran might support conditional compilation. We *might* start using this in about ten years.

G Building ROMS

G.1 Environment Variables for make

ROMS has a growing list of choices the user must make about the compilation before starting the compile process, set in user-defined variables. Since we now use **gnu make**, it is possible to set the value of these variables in the Unix environment, rather than necessarily inside the **Makefile** (see §H). The user-definable variables understood by the ROMS **makefile** are:

ROMS_APPLICATION Set the **cpp** option defining the particular application. This is used for setting up options inside the code specific to this application and also determines the name of the **.h** header file for it. This can be either a predefined case, such as **BENCHMARK**, or one of your own, such as **NEP5**.

MY_HEADER_DIR Sets the path to the user's header file, if any. It can be left empty for the standard cases, where **benchmark.h** and the like are found in **ROMS/Include**, which is already in the search path. In the case of **NEP5**, this is set to **Apps/NEP** where **nep5.h** resides.

MY_ANALYTICAL_DIR Sets the path to the user's analytic files described in §**??**, if any. This can be **User/Functionals** or some other location. I tend to place both the header file and the functionals in the same directory, one directory per application.

MY_CPP_FLAGS Set tunable **cpp** options. Sometimes it is desirable to activate one or more **cpp** options to run different variants of the same application without modifying its header file. If this is the case, specify each option here using the **-D** syntax. Notice that you need to use the shell's quoting syntax (either single or double quotes) to enclose the definition if you are using one of the build scripts below.

NestedGrids Integer number of grids in the setup, usually 1.

Compiler-specific Options These flags are used by the files inside the **Compilers** directory.

- **USE_DEBUG** Set this to **on** to turn off optimization and turn on the **-g** flag for debugging.
- **USE_MPI** Set this if running an MPI parallel job.
- **USE_OpenMP** Set this if running an OpenMP parallel job.
- **USE_MPIF90** I'm frankly not sure about this one. I suppose if you have both mpich and some other MPI for a given compiler/system pair, this could be used to switch between them.
- **USE_LARGE** Some systems support both 32-bit and 64-bit options. Select this to get 64-bit addressing, usually used for programs need more than 2 GB of memory.
- **NETCDF_INCDIR** The location of the **netcdf.mod** and **typesizes.mod** files.
- **NETCDF_LIBDIR** The location of the NetCDF library.
- **USE_NETCDF4** Set this if linking against the NetCDF4 library, which needs the HDF5 library and therefore:
- **HDF5_LIBDIR** The location of the HDF5 library.

FORT A shorthand name for the compiler to be used when selecting which system-compiler file is to be included from the **Compilers** directory. See section §H.2.3 and §G.2.

BINDIR Directory in which to place the binary executable. The default is "**.**", the current (top) directory.

SCRATCH_DIR Put the **.f90** and the temporary binary files in a build directory to avoid clutter. The default is **Build** under the top directory. It can also point to differing places if you want to keep these files for multiple projects at the same time, each in their own directory.

G.2 Providing the Environment

Before compiling, you will need to find out some background information:

- What is the name of your compiler?
- What is returned by **uname -s** on your system?
- Is there a working NetCDF library?
- Where is it?
- Was it built with the above compiler?
- Do you have access to MPI or OpenMP?

As described more fully in §H.2.3, the **makefile** will be looking for a file in the **Compilers** directory with the combination of your operating system and your compiler. For instance, using Linux and the Pathscale compiler, the file would be called **Linux-path.mk**. Is the corresponding file for your system and compiler in the **Compilers** directory? If not, you will have to create it following the existing examples there.

Next, there are two ways to provide the location for the NetCDF files (and optional HDF5 library). One is by editing the corresponding lines in your system-compiler file. Another way is through the Unix environment variables. If you are always going to be using the same compiler on each system, you can edit your **.profile** or **.login** files to globally set them. Here is an example for **tcsh**:

```
setenv NETCDF_INCDIR /usr/local/netcdf4/include
setenv NETCDF_LIBDIR /usr/local/netcdf4/lib
setenv HDF5_LIBDIR /usr/local/hdf5/lib
```

The **ksh/bash** equivalent is:

```
export NETCDF_INCDIR=/usr/local/netcdf4/include
export NETCDF_LIBDIR=/usr/local/netcdf4/lib
export HDF5_LIBDIR=/usr/local/hdf5/lib
```

G.2.1 Build scripts

If you have more than one application (or more than one compiler), you will get tired of editing the **makefile**. One option is to have a **makefile** for each configuration, then type:

```
make -f makefile.circle_pgi
```

for instance. If that's too cumbersome, there's also an option of keeping track of the user-defined choices in a build script. There are now two of these scripts in the **ROMS/Bin** directory: **build.sh** (which is surprisingly a **csh** script) and **build.bash**. The build scripts use environment variables to provide values for the list above, overwriting those found in the ROMS **makefile**. Just as in the multiple **makefile** option, you will need as many copies of the build script as you have applications. The scope of these variables is local to the build script, allowing you to compile different applications at the same time from the same sources as long as each **$(SCRATCH_DIR)** is unique.

Both scripts have the same options:

-j [N] Compile in parallel using **N** cpus, omit argument for all available CPUs.

-noclean Do not clean already compiled objects.

Note that the default is to compile serially and to issue a "**make clean**" before compiling. It is left as an exercise for the user if they prefer different default behavior.

There are also a few variables which are not recognized by the ROMS **makefile**, but are used locally inside the build script. These are:

MY_PROJECT_DIR This is used in setting **$(SCRATCH_DIR)** and **$(BINDIR)**.

MY_ROMS_SRC Set the path to the user's local current ROMS source code. This is used so that the script can be run from any directory, not necessarily only from the top ROMS directory.

H Makefiles

One of the software development tools which comes with the UNIX operating system is called **make**. **Make** has many uses, but is most commonly used to keep track of how a large program should be compiled. ROMS now requires the **gnu** version of **make**, sometimes known as **gmake** (Mecklenburg [25]). This appendix describes generic **make**, the **gnu** enhancements to **make**, as well as describing the **makefile** structure used in ROMS. This material first appeared in the ARSC HPC Newsletter in several segments and has been updated and restructured here.

H.1 Introduction to Portable make

Make gets its instructions from a description file, by default named **makefile** or **Makefile**. This file is called the **Makefile**, but other files can be used by invoking **make** with the **-f** option:

```
make -f Makefile.yukon
```

The ancester to ROMS originally had a **Makefile** that looked something like:

```
model: main.o init.o plot.o
<TAB>   f77 -o model main.o init.o plot.o

main.o: main.f
<TAB>   f77 -c -O main.f

init.o: init.f
<TAB>   f77 -c -O init.f

plot.o: plot.f
<TAB>   f77 -c -O0 plot.f

clean:
<TAB>   rm *.o core
```

The default thing to build is **model**, the first target. The syntax is:

```
target: dependencies
<TAB>  command
<TAB>  command
```

The target **model** depends on the object files, **main.o** and friends. They have to exist and be up to date before model's link command can be run. If the target is out of date according to the timestamp on the file, then the commands will be run. Note that the tab is required on the command lines.

The other targets tell **make** how to create the object files and that they in turn have dependencies. To compile **model**, simply type "**make**". **Make** will look for the file **makefile**, read it, and do whatever is necessary to make **model** up to date. If you edit **init.f**, that file will be newer than **init.o**. **Make** would see that **init.o** is out of date and run the "**f77 -c -O init.f**" command. Now **init.o** is newer than **model**, so the link command "**f77 -o model main.o init.o plot.o**" must be executed.

To clean up, type "**make clean**" and the **clean** target will be brought up to date. The **clean** target has no dependencies. What happens in that case is that the command ("**rm *.o core**") will always be executed.

The original version of this **Makefile** turned off optimization on **plot.f** due to a compiler bug, but hopefully you won't ever have to worry about that.

H.1.1 Macros

Make supports a simple string substitution macro. Set it with:

```
MY_MACRO = nothing today
```

and refer to it with:

```
$(MY_MACRO)
```

The convention is to put the macros near the top of your **Makefile** and to use upper case. Also, use separate macros for the name of your compiler and the flags it needs:

```
F90 = f90
F90FLAGS = -O3
LIBDIR = /usr/local/lib
LIBS = -L$(LIBDIR) -lmylib
```

Let's rewrite our **Makefile** using macros:

```
#
# IBM version
#
F90 = xlf90_r
F90FLAGS = -O3 -qstrict
LDFLAGS = -bmaxdata:0x40000000

model: main.o init.o plot.o
        $(F90) $(LDFLAGS) -o model main.o init.o plot.o

main.o: main.f
        $(F90) -c $(F90FLAGS) main.f

init.o: init.f
        $(F90) -c $(F90FLAGS) init.f

plot.o: plot.f
        $(F90) -c $(F90FLAGS) plot.f

clean:
        rm *.o core
```

Now when we change computers, we only have to change the compiler name in one place. Likewise, if we want to try a different optimization level, we only specify that in one place.

Notice that you can use comments by starting the line with a #.

H.1.2 Implicit Rules

Make has some rules already built in. For fortran, you might be able to get away with:

```
OBJS = main.o init.o plot.o

model: $(OBJS)
        $(FC) $(LDFLAGS) -o model $(OBJS)
```

as your whole **Makefile**. **Make** will automatically invoke its default Fortran compiler, possibly **f77** or **g77**, with whatever default compile options it happens to have (**FFLAGS**). One built in rule often looks like:

```
.c.o:
        $(CC) $(CFLAGS) -c $<
```

which says to compile **.c** files to **.o** files using the compiler **CC** and options **CFLAGS**. We can write our own suffix rules in this same style. The only thing to watch for is that make by default has a limited set of file extentions that it knows about. Let's write our **Makefile** using a suffix rule:

```
#
# Cray version
#
F90 = f90
F90FLAGS = -O3
LDFLAGS =

.f.o:
        $(F90) $(F90FLAGS) -c $<

model: main.o init.o plot.o
        $(F90) $(LDFLAGS) -o model main.o init.o plot.o

clean:
        rm *.o core
```

H.1.3 Dependencies

There may be additional dependencies beyond the **source->object** ones. In our little example, all our source files include a file called **commons.h**. If **commons.h** gets modified to add a new variable, everything must be recompiled. **Make** won't know that unless you tell it:

```
# include dependencies
main.o: commons.h
init.o: commons.h
plot.o: commons.h
```

Fortran 90 introduces module dependencies as well. See §**??** for how we automatically generate this dependency information.

The most common newbie mistake is to forget that the commands after a target *have* to start with a tab.

H.2 gnu make

Over the years, the community has moved from the stance of writing portable **Makefiles** to a stance of just using a powerful, portable **make**. The previous section described a portable subset of **make** features. We now delve into some of the more powerful tools available in **gnu make**.

H.2.1 Make rules

The core of **make** hasn't changed in decades, but concentrating on **gmake** allows one to make use of its nifty little extras designed by real programmers to help with real projects. The first change that matters to my **Makefiles** is the change from suffix rules to pattern rules. I've always found the **.SUFFIXES** list to be odd, especially since **.f90** is not in the default list. Good riddance to all of that! For a concrete example, the old way to provide a rule for going from **file.f90** to **file.o** is:

```
.SUFFIXES: .o .f90 .F .F90
.f90.o:
<TAB>   $(FC) -c $(FFLAGS) $<
```

while the new way is:

```
%.o: %.f90
<TAB>   $(FC) -c $(FFLAGS) $<
```

In fact, going to a uniform **make** means that we can figure out what symbols are defined and use their standard values—in this case, $**(FC)** and $**(FFLAGS)** are the built-in default names for the compiler and its options. If you have any questions about this, you can always run **make** with the **-p** (or **--print-data-base**) option. This prints out all the rules **make** knows about, such as:

```
# default
COMPILE.f = $(FC) $(FFLAGS) $(TARGET_ARCH) -c
```

Printing the rules database will show variables that **make** is picking up from the environment, from the **Makefile**, and from its built-in rules—and which of these sources is providing each value.

H.2.2 Assignments

In the old days, I only used one kind of assignment: = (equals sign). **Gmake** has several kinds of assignment (other **makes** might as well, but I no longer have to know or care). An example of the power of **gnu make** is shown by an example from my Cray X1 **Makefile**. There is a routine which runs much more quickly if a short function in another file is inlined. The way to accomplish this is through the **-O inlinefrom=file** directive to the compiler. I can't add this option to **FFLAGS**, since the inlined routine won't compile with this directive—there is only the one file that needs it. I had:

```
    FFLAGS = -O 3,aggress -e I -e m
    FFLAGS2 = -O 3,aggress -O inlinefrom=lmd_wscale.f90 -e I -e m

lmd_skpp.o:
<TAB>   $(FC) -c $(FFLAGS2) $*.f90
```

The first change I can make to this using other assignments is:

```
    FFLAGS := -O 3,aggress -e I -e m
    FFLAGS2 := $(FFLAGS) -O inlinefrom=lmd_wscale.f90
```

The **:=** assignment means to evaluate the right hand side immediately. In this case, there is no reason not to, as long as the second assigment follows the first one (since it's using the value of $**(FFLAGS)**). For the plain equals, **make** doesn't evaluate the right-hand side until its second pass through the **Makefile**. However, **gnu make** supports an assignment which avoids the need for **FFLAGS2** entirely:

```
lmd_skpp.o: FFLAGS += -O inlinefrom=lmd_wscale.f90
```

What this means is that for the target **lmd_skpp.o** only, append the inlining directive to **FFLAGS**. I think this is pretty cool!

One last kind of assignment is to set the value only if there is no value from somewhere else (the environment, for instance):

```
FC ?= mpxlf90_r
```

If we used := or =, we would override the value from the environment.

H.2.3 Include and a Few Functions

One reasonably portable **make** feature is the include directive. It can be used to clean up the **Makefile** by putting bits in an include file. The syntax is simply:

```
include file
```

and we use it liberally to keep the project information neat. We can also include a file with the system/compiler information in it, assuming we have some way of deciding *which* file to include. We can use **uname -s** to find out which operating system we're using. We also need to know which compiler we're using.

One user-defined variable is called **FORT**, the name of the Fortran compiler. This value is combined with the result of "**uname -s**" to provide a machine and compiler combination. For instance, **ftn** on Linux is the Cray cross-compiler. This would link to a different copy of the NetCDF library and use different compiler flags than the Intel compiler which might be on the same system.

```
# The user sets FORT:
  FORT ?= ftn
  OS := $(shell uname -s | sed 's/[\/ ]/-/g')

  include $(COMPILERS)/$(OS)-$(strip $(FORT)).mk
```

We pick one include file at compile time, here picking **Linux-ftn.mk**, containing the Cray cross-compiler information. The value **Linux** comes from the **uname** command, the **ftn** comes from the user, and the two are concatenated. The sed command will turn the slash in **UNICOS/mp** into a dash; the native Cray include file is **UNICOS-mp-ftn.mk**. Strip is a built-in function to strip away any extra white space.

Another tricky system is **CYGWIN**, which puts a version number in the **uname** output, such as **CYGWIN_NT-5.1**. **Gnu make** has quite a few built-in functions, one of which allows us to do string substitution:

```
OS := $(patsubst CYGWIN_%,CYGWIN,$(OS))
```

In **make**, the % symbol is a sort of wild card, much like * in the shell. Here, we match **CYGWIN** followed by an underscore and anything else, replacing the whole with simply **CYGWIN**. Another example of a built-in function is the substitution we do in:

```
objects = $(subst .F,.o,$(sources))
```

where we build our list of objects from the list of sources. There are quite a few other functions, plus the user can define their own. From the book[25]:

GNU make supports both built-in and user-defined functions. A function invocation looks much like a variable reference, but includes one or more parameters separated by commas. Most built-in functions expand to some value that is then assigned to a variable or passed to a subshell. A user-defined function is stored in a variable or macro and expects one or more parameters to be passed by the caller.

We will show some user-defined functions in §H.3.3.

H.2.4 Conditionals

We used to have way too many **Makefiles**, a separate one for each of the serial/MPI/OpenMP versions on each system (if supported). For instance, the name of the IBM compiler changes when using MPI; the options change for OpenMP. The compiler options also change when using 64-bit addressing or for debugging. A better way to organize this is to have the user select 64-bit or not, MPI or not, etc., then use conditionals. The complete list of user definable **make** variables is given in §G.1.

Gnu make supports two kinds of **if** test, **ifdef** and **ifeq** (plus the negative versions **ifndef, ifneq**). The example from the book is:

```
ifdef COMSPEC
   # We are running Windows
else
   # We are not on Windows
endif
```

An example from the IBM include file is:

```
ifdef USE_DEBUG
           FFLAGS += -g -qfullpath
else
           FFLAGS += -O3 -qstrict
endif
```

To test for equality, an example is:

```
ifeq ($(USE_MPI),on)
   # Do MPI things
endif
```

or

```
ifeq "$(USE_MPI)" "on"
   # Do MPI things
endif
```

The user has to set values for the **USE_MPI**, **USE_DEBUG**, and **USE_LARGE** switches in the **Makefile** or the build script *before* the compiler-dependent piece is included:

```
USE_MPI ?= on
USE_DEBUG ?=
USE_LARGE ?=
```

The **Makefile** uses the conditional assign "**?=**" in case a build script is used to set them in the environment. Be sure to leave the switches meant to be off set to an empty string—the string "**off**" will test true on an **ifdef** test.

H.3 Multiple Source Directories the ROMS Way

There's more than one way to divide your sources into separate directories. The choices we have made include nonrecursive **make** and putting the temporary files in their own **$(SCRATCH_DIR)** directory. These include the **.f90** files which have been through the C preprocessor, object files, module files, and libraries.

H.3.1 Directory Structure

The directory structure of the source code has the top directory, a **Master** directory, a **ROMS** directory with a number of subdirectories, and several other directories. **Master** contains the main program while the rest contain sources for libraries and other files. Note that the bulk of the source code gets compiled into files that become libraries with the **ar** command, one library per directory. There is also a **Compilers** directory for the system- and compiler-specific **Makefile** components.

H.3.2 Conditionally Including Components

The **makefile** will build the lists of libraries to create and source files to compile. They start out empty at the top of the **makefile**:

```
  sources    :=
  libraries  :=
```

That's simple enough, but the list of directories to search for these sources will depend on the options chosen by the user, not just in the **make** options (§G.1), but inside the **ROMS_HEADER** file as well. How does this happen? Once **make** knows how to find the **ROMS_HEADER**, it is used by **cpp** to generate an include file telling **make** about these other options.

```
MAKE_MACROS := Compilers/make_macros.mk
 MACROS := $(shell cpp -P $(ROMS_CPPFLAGS) Compilers/make_macros.h > \
                 $(MAKE_MACROS); $(CLEAN) $(MAKE_MACROS))
```

The **make_macros.h** file contains blocks such as:

```
#ifdef SWAN_COUPLING
  USE_SWAN := on
#else
  USE_SWAN :=
#endif
```

The resulting **make_macros.mk** file will simply end up with either

```
  USE_SWAN := on
```

or

```
  USE_SWAN :=
```

This file can then be included by the **makefile** and the variable **USE_SWAN** will have the correct state for this particular compilation. We can now use it and all the similar flags to build a list of directories.

We initialize two lists:

```
 modules  :=
 includes :=    ROMS/Include
```

Add the adjoint bits:

```
ifdef USE_ADJOINT
 modules  +=    ROMS/Adjoint
endif
ifdef USE_ADJOINT
 includes +=    ROMS/Adjoint
endif
```

Add the bits we'll always need:

```
 modules  +=    ROMS/Nonlinear \
                ROMS/Functionals \
                ROMS/Utility \
                ROMS/Modules
 includes +=    ROMS/Nonlinear \
                ROMS/Utility \
                ROMS/Drivers
```

Then we add in some more:

```
ifdef USE_SWAN
 modules  +=    Waves/SWAN/Src
 includes +=    Waves/SWAN/Src
endif

 modules  +=    Master
 includes +=    Master Compilers
```

Now that our lists are complete, let's put them to use:

```
vpath %.F $(modules)
vpath %.h $(includes)
vpath %.f90 $(SCRATCH_DIR)
vpath %.o $(SCRATCH_DIR)

include $(addsuffix /Module.mk,$(modules))

CPPFLAGS += $(patsubst %,-I%,$(includes))
```

1. **vpath** is a standard **make** feature for providing a list of directories for **make** to search for files of different types. Here we are saying that ***.F** files can be found in the directories provided in the **$(modules)** list, and so on for the others.

2. For each directory in the **$(modules)** list, **make** will include the file **Module.mk** that is found there. More on these later.

3. For each directory in the **$(includes)** list, add that directory to the list searched by **cpp** with the $-\mathbf{I}$ flag.

H.3.3 User-defined make Functions

The **Module.mk** fragments mentioned before call some user-defined functions. It's time to show these functions and talk about how they work. They get defined in the top Makefile:

```
#-------------------------------------------------------------------------
#  Make functions for putting the temporary files in $(SCRATCH_DIR)
#-------------------------------------------------------------------------

# $(call source-dir-to-binary-dir, directory-list)
source-dir-to-binary-dir = $(addprefix $(SCRATCH_DIR)/, $(notdir $1))

# $(call source-to-object, source-file-list)
source-to-object = $(call source-dir-to-bindary-dir,   \
                   $(subst .F,.o,$(filter %.F,$1)))

# $(call f90-source, source-file-list)
f90-source = $(call source-dir-to-binary-dir,     \
                   $(subst .F,.f90,$1))

# $(call make-library, library-name, source-file-list)
define make-library
   libraries += $(SCRATCH_DIR)/$1
   sources   += $2

   $(SCRATCH_DIR)/$1: $(call source-dir-to-binary-dir,    \
                      $(subst .F,.o,$2))
       $(AR) $(ARFLAGS) $$@ $$^
       $(RANLIB) $$@
endef

# $(call one-compile-rule, binary-file, f90-file, source-files)
define one-compile-rule
  $1: $2 $3
       cd $$(SCRATCH_DIR); $$(FC) -c $$(FFLAGS) $(notdir $2)

  $2: $3
       $$(CPP) $$(CPPFLAGS) $$(MY_CPP_FLAGS) $$< > $$@
       $$(CLEAN) $$@

endef

# $(compile-rules)
define compile-rules
  $(foreach f, $(local_src),       \
    $(call one-compile-rule,$(call source-to-object,$f), \
    $(call f90-source,$f),$f))
endef
```

1. We define a function to convert the path from the source directory to the **Build** directory, called **source-dir-to-binary-dir**. Note that the **Build** directory is called $**(SCRATCH_DIR)** here. All it does is strip off the leading directory with the the built-in function **notdir**, then paste on the **Build** directory.

2. Next comes **source-to-object**, which calls the function above to return the object filename when given the source filename. It assumes that all sources have a **.F** extension.

3. A similar function is **f90-source**, which returns the name of the intermediate source which is created by **cpp** from our **.F** file.

4. The **Module.mk** fragment in each library source directory invokes **make-library**, which takes the library name and the list of sources as its arguments. The function adds its **library** to the global list of **libraries** and provides rules for building itself. The double dollar signs are to delay the variable substitution. Note that we call **source-dir-to-binary-dir** instead of **source-to-object**—this is a work-around for a make bug.

5. The next, **one-compile-rule**, takes three arguments: the **.o** filename, the **.f90** filename, and the **.F** filename. From these, it produces the **make** rules for running **cpp** and the compiler.

   A note on directories: **make** uses **vpath** to find the source file where it resides. It would be possible to compile from the top directory and put the **.o** file in **Build** with the appropriate arguments, but I don't know how to get the **.mod** file into **Build** short of a **mv** command. Likewise, if we compile in the top directory, we need to know the compiler option to tell it to look in **Build** for the **.mod** files it uses. Doing a **cd** to **Build** before compiling is just simpler.

6. The last, **compile-rules**, is given a list of sources, then calls **one-compile-rule** once per source file.

Again, you can invoke **make -p** to see how **make** internally transforms all this into actual targets and rules.

H.3.4 Library Module.mk

In each library directory, there is a file called **Module.mk** which gets included by the top level **makefile**. These **Module.mk** bits build onto the list of sources and libraries to be compiled and built, respectively. These **Module.mk** files look something like:

```
local_sub  := ROMS/Nonlinear
local_lib  := libNLM.a

local_src  := $(wildcard $(local_sub)/*.F)

$(eval $(call make-library,$(local_lib),$(local_src)))

$(eval $(compile-rules))
```

First, we provide the name of the current directory and the library to be built from the resident sources. Next, we use the **wildcard** function to search the subdirectory for these sources. Note that every **.F** file found will be compiled. If you have half-baked files that you don't want used, make sure they have a different extension.

Each subdirectory is resetting the **local_src** variable. That's OK because we're saving the values in the global **sources** variable inside the **make-library** function, which also adds the local library to the **libraries** list. The **compile-rules** function uses this **local_src** variable to generate the rules for compiling each file, placing the resulting files in the **Build** directory.

H.3.5 Main Program

The main program is in a directory called **Master** and its **Module.mk** is similar to the library one:

```
local_sub  := Master
local_src  := $(wildcard $(local_sub)/*.F)

local_objs := $(subst .F,.o,$(local_src))
local_objs := $(addprefix $(SCRATCH_DIR)/, $(notdir $(local_objs)))

sources    += $(local_src)

ifdef LD_WINDOWS
$(BIN): $(libraries) $(local_objs)
        $(LD) $(FFLAGS) $(local_objs) -o $@ $(libraries) $(LIBS_WIN32) $(LDFLAGS)
else
$(BIN): $(libraries) $(local_objs)
        $(LD) $(FFLAGS) $(LDFLAGS) $(local_objs) -o $@ $(libraries) $(LIBS)
endif

$(eval $(compile-rules))
```

Instead of a rule for building a library, we have a rule for building a binary **$(BIN)**. In this case, the name of the ROMS binary is defined elsewhere. The binary depends on the **libraries** getting compiled first, as well as the local sources. During the link, the $**(libraries)** are compiled from the sources in the other directories, while $**(LIBS)** are exteral libraries such as NetCDF.

H.3.6 Top Level Makefile

Now we get to the glue that holds it all together. We've covered many things so far, but there's still a few bits which might be confusing:

1. There can be rare cases where you might have special code for some systems. You can check which system you are on in the **.F** file with:

   ```
   #ifdef X86_64
   !      special stuff
   #endif
   ```

   To be sure this is defined on each **X86_64** system, it has to be passed to **cpp**:

   ```
   CPPFLAGS += -D$(shell echo ${OS} | tr "-" "_" | tr [a-z] [A-Z])
   CPPFLAGS += -D$(shell echo ${CPU} | tr "-" "_" | tr [a-z] [A-Z])
   CPPFLAGS += -D$(shell echo ${FORT} | tr "-" "_" | tr [a-z] [A-Z])

   CPPFLAGS += -D'ROOT_DIR="$(ROOTDIR)"'
   ifdef ROMS_APPLICATION
     CPPFLAGS  += $(ROMS_CPPFLAGS)
     CPPFLAGS  += -DNestedGrids=$(NestedGrids)
     MDEPFLAGS += -DROMS_HEADER="$(HEADER)"
   endif
   ```

   This guarantees that **CPPFLAGS** will have terms in it such as:

   ```
   -DLINUX -DX86_64 -DPGI
   -D'ROOT_DIR="/export/staffdata/kate/roms/trunk"' -DSHOREFACE
   -D'HEADER="shoreface.h"' -D'ROMS_HEADER="shoreface.h"'
   -DNestedGrids=1
   ```

2. For **mod_strings.F**, there is a special case:

```
$(SCRATCH_DIR)/mod_strings.f90: CPPFLAGS += -DMY_OS='"$(OS)"' \
              -DMY_CPU='"$(CPU)"' -DMY_FORT='"$(FORT)"' \
              -DMY_FC='"$(FC)"' -DMY_FFLAGS='"$(FFLAGS)"'
```

   allowing ROMS to report in its output:

```
 Operating system : Linux
 CPU/hardware     : x86_64
 Compiler system  : pgi
 Compiler command : pgf90
 Compiler flags   :  -O3 -tp k8-64 -Mfree

 Local Root    : /export/staffdata/kate/roms/trunk
 Header Dir    : ./ROMS/Include
 Header file   : shoreface.h
```

   Though this doesn't seem to work on the Mac.

3. The very first **makefile** I showed had a set list of files to remove on **make clean**. We now build a list, called **clean_list**:

```
 clean_list := core *.ipo $(SCRATCH_DIR)

ifeq "$(strip $(SCRATCH_DIR))" "."
  clean_list := core *.o *.oo *.mod *.f90 lib*.a *.bak
  clean_list += $(CURDIR)/*.ipo
endif
```

   In other words, we want to clean up the **Build** directory unless it happens to be the top level directory, in which case we only want to remove specific files there.

4. "**all**" is the first target that gets seen by **make**, making it the default **target**. In this case, we know there is only the one binary, whose name we know—the book[25] shows what to do with more than one binary. Both "**all**" and "**clean**" are phony targets in that no files of those names get generated—make has the **.PHONY** designation for such targets. Also, the **clean** target doesn't require any compiler information, so the compiler include doesn't happen if the target is "clean":

```
ifneq "$(MAKECMDGOALS)" "clean"
  include $(COMPILERS)/$(OS)-$(strip $(FORT)).mk
endif
```

   $(MAKECMDGOALS) is a special variable containing the current **make** target.

5. We'll be creating different executable names, depending on which options we've picked:

```
BIN := $(BINDIR)/oceanS
ifdef USE_DEBUG
  BIN := $(BINDIR)/oceanG
else
```

```
 ifdef USE_MPI
   BIN := $(BINDIR)/oceanM
 endif
 ifdef USE_OpenMP
   BIN := $(BINDIR)/oceanO
 endif
endif
```

6. The NetCDF library gets included during the final link stage. However, we are now using the Fortran 90 version of it which requires its module information as well. We just copy the **.mod** files into the **Build** directory:

```
   NETCDF_MODFILE := netcdf.mod
TYPESIZES_MODFILE := typesizes.mod

$(SCRATCH_DIR)/$(NETCDF_MODFILE): | $(SCRATCH_DIR)
        cp -f $(NETCDF_INCDIR)/$(NETCDF_MODFILE) $(SCRATCH_DIR)

$(SCRATCH_DIR)/$(TYPESIZES_MODFILE): | $(SCRATCH_DIR)
        cp -f $(NETCDF_INCDIR)/$(TYPESIZES_MODFILE) $(SCRATCH_DIR)
```

   Old versions of NetCDF do not have the **typesizes.mod** file, in which case it has to be removed from the following dependency list:

```
$(SCRATCH_DIR)/MakeDepend: makefile \
                           $(SCRATCH_DIR)/$(NETCDF_MODFILE) \
                           $(SCRATCH_DIR)/$(TYPESIZES_MODFILE) \
                           | $(SCRATCH_DIR)
```

7. Then there is **MakeDepend** itself. This file is created by the **Perl** script **sfmakedepend**. **MakeDepend** gets created by "**make depend**" and included on **make**'s second pass through the **makefile**:

```
depend: $(SCRATCH_DIR)
        $(SFMAKEDEPEND) $(MDEPFLAGS) $(sources) > $(SCRATCH_DIR)/MakeDepend

ifneq "$(MAKECMDGOALS)" "clean"
  -include $(SCRATCH_DIR)/MakeDepend
endif
```

   The dash before the **include** tells **make** to ignore errors so that **make depend** will succeed before the file exists. The **MakeDepend** file will contain the include and module dependencies for each source file, such as:

```
Build/mod_diags.o: tile.h cppdefs.h globaldefs.h shoreface.h
Build/mod_diags.f90: tile.h cppdefs.h globaldefs.h shoreface.h
Build/mod_diags.o: Build/mod_kinds.o Build/mod_param.o Build/mod_diags.f90
```

   Note that the **.h** files are included by **cpp**, so that both the **.f90** and **.o** files become out of date when an include file is modified. Without the module dependencies, **make** would try to build the sources in the wrong order and the compiler would fail with a complaint about not finding **mod_param.mod**, for instance.

H.4 Final warnings

The cost of this nifty **make** stuff is:

1. We're a little closer to the **gnu make** bugs here, and we need a newer version of **gnu make** than before (version 3.81, 3.80 if you're lucky). Hence this stuff at the top of the **makefile**:

   ```
   NEED_VERSION := 3.80 3.81
   $(if $(filter $(MAKE_VERSION),$(NEED_VERSION)),,        \
    $(error This makefile requires one of GNU make version $(NEED_VERSION).))
   ```

2. One annoyance is that the compile phase of **one-compile-rule** didn't seem to be working. I was forced to keep the pattern rule for **.o** files:

   ```
   %.o: %.f90
   <TAB>  cd $(SCRATCH_DIR); $(FC) -c $(FFLAGS) $(notdir $<)
   ```

3. The **Makefile** dependencies get just a little trickier every change we make. Note that **F90** has potentially both **include** and **module** use dependencies. The book example uses the C compiler to produce its own dependencies for each source file into a corresponding **.d** file to be included by **make**. Our Fortran compilers are not so smart. For these hairy compiles, it's critical to have accurate dependency information unless we're willing to **make clean** between compiles.

References

[1] A. Arakawa and V. R. Lamb. *Methods of computational physics*, volume 17, pages 174–265. Academic Press, 1977.

[2] R. Aris. *Vectors, tensors and the basic equations of fluid mechanics*. Prentice-Hall, Englewood Cliffs, NJ, 1962.

[3] G. K. Batchelor. *An introduction to fluid dynamics*. Cambridge University Press, 1967.

[4] A. Beckmann and D. B. Haidvogel. Numerical simulation of flow around a tall, isolated seamount. part i: Problem formulation and model accuracy. *J. Phys. Oceanogr.*, 23:1736–1753, 1993.

[5] W.P. Budgell. Numerical simulation of ice-ocean variability in the barents sea region: Towards dynamical downscaling. *Ocean Dynamics*, 2005. doi:10.1007/s10236-005-0008-3.

[6] J. A. Francis and A. Schweiger. A new window opens on the arctic. *Trans. Amer. Geophys. Un.*, 81:77–83, 2000.

[7] N. G. Freeman, A. M. Hale, and M. B. Danard. A modified sigma equations' approach to the numerical modeling of great lake hydrodynamics. *J. Geophys. Res.*, 77(6):1050–1060, 1972.

[8] J. M. N. T. Gray and P. D. Killworth. Stability of the viscous-plastic sea ice rheology. *J. Phys. Oceanogr.*, 25:971–978, 1995.

[9] J. M. N. T. Gray and P. D. Killworth. Sea ice ridging schemes. *J. Phys. Oceanogr.*, 26:2420–2428, 1996.

[10] D. B. Haidvogel, H. G. Arango, W. P. Budgell, B. D. Cornuelle, E. Curchitser, E. Di Lorenzo, K. Fennel, W. R. Geyer, A. J. Hermann, L. Lanerolle, J. Levin, J. C. McWilliams, A. J. Miller, A. M. Moore, T. M. Powell, A. F. Shchepetkin, C. R. Sherwood, R. P. Signell, J. C. Warner, and J. Wilkin. Ocean forecasting in terrain-following coordinates: formulation and skill assessment of the regional ocean modeling system. *J. Comp. Phys.*, 227:3429–3430, 2007. doi:10.1016/j.jcp.2007.01.016.

[11] D. B. Haidvogel, H. G. Arango, K. Hedstrom, A. Beckmann, P. Malanotte-Rizzoli, and A. F. Shchepetkin. Model evaluation experiments in the north atlantic basin: Simulations in non-linear terrain-following coordinates. *Dyn. Atmos. Ocean.*, 32:239–281, 2000.

[12] D. B. Haidvogel and A. Beckmann. *Numerical Ocean Circulation Modeling*. Imperial College Press, 1999.

[13] R. L. Haney. On the pressure gradient force over steep topography in sigma coordinate ocean models. *J. Phys. Oceanogr.*, 21:610–619, 1991.

[14] W. P. Hazard. Using cpp to aid portability. *Computer Language*, 8(11):49–54, 1991.

[15] K. S. Hedstrom. Technical manual for a coupled sea-ice/ocean circulation model (version 2). Technical report, Institute of Marine and Coastal Sciences, New Brunswick, NJ, June 2000. OCS Study MMS 2000-047.

[16] W. D. Hibler, III. A dynamic thermodynamic sea ice model. *J. Phys. Oceanogr.*, 9:815–846, 1979.

[17] W. D. Hibler, III. Documentation for a two-level dynamic thermodynamic sea ice model. Technical report, USACRREL, Hanover, NH, 1980. Special Report 80-8.

[18] D. R. Jackett and T. J. McDougall. Stabilization of hydrographic data. *J. Atmos. Ocean. Tech.*, 12:381–389, 1995.

[19] Y. Kanarska, A. F. Shchepetkin, and J. C. McWilliams. Algorithm for non-hydrostatic dynamics in roms. *Ocean Modeling*, 18:143–174, 2007.

[20] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey 07632, second edition, 1988.

[21] W. G. Large, J. C. McWilliams, and S. C. Doney. Oceanic vertical mixing: a review and a model with a nonlocal boundary layer parameterization. *Rev. Geophys.*, 32:363–403, 1994.

[22] B. P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Comput. Method Appl. Mech. Eng.*, 19:59–98, 1979.

[23] A. Macks and J. Middleton. Numerical modelling of wind-driven upwelling and downwelling. University of New South Wales, 1993.

[24] John D. McCalpin. A comparison of second-order and fourth-order pressure gradient algorithms in a σ-coordinate ocean model. *Int. J. Num. Meth. Fluids*, 18:361–383, 1994.

[25] R. Mechlenburg. *Managing Projects with GNU Make*. O'Reilly & Associates, Inc., Sebastopol, CA, 2005.

[26] G. L. Mellor and L. Kantha. An ice-ocean coupled model. *J. Geophys. Res.*, 94:10,937–10,954, 1989.

[27] G. L. Mellor and T. Yamada. A hierarchy of turbulence closure models for planetary boundary layers. *J. Atmos. Sci.*, 31:1791–1806, 1974.

[28] I. Orlanski. A simple boundary condition for unbounded hyperbolic flows. *J. Comp. Phys.*, 21(3):251–269, July 1976.

[29] C. L. Parkinson and W. M. Washington. A large-scale numerical model of sea ice. *J. Geophys. Res.*, 84:6565–6575, 1979.

[30] P. Penven, L. Debreu, P. Marchesiello, and J. C. McWilliams. Evaluation and application of the roms 1-way embedding procedure to the central california upwelling system. *Ocean Modeling*, 12:157–187, 2006.

[31] N. A. Phillips. A coordinate system having some special advantages for numerical forecasting. *J. Meteorology*, 14(2):184–185, 1957.

[32] P. J. Rasch. Conservative shape-preserving two-dimensional transport on a spherical reduced grid. *Mon. Wea. Rev.*, 122:1337–1350, 1994.

[33] W. H. Raymond and H. L. Kuo. A radiation boundary condition for multi-dimensional flows. *Quart. J. R. Met. Soc.*, 110:535–551, 1984.

[34] R. Rew, G. Davis, S. Emmerson, and H. Davies. *NetCDF User's Guide*. Unidata, University Corporation for Atmospheric Research, Boulder, Colorado, 1996. Version 2.4.

[35] A. F. Shchepetkin and J. C. McWilliams. The regional ocean modeling system (roms): A split-explicit, free-surface, topography-following coordinates oceanic model. *Ocean Modeling*, 9:347–404, 2005.

[36] A. F. Shchepetkin and J. C. McWilliams. Computational kernel algorithms for fine-scale, multi-process, long-time oceanic simulations. In R. Temam and J. Tribbia, editors, *Handbook of Numerical Analysis: Computational Methods for the Ocean and the Atmosphere*. Elsevier Science, 2008. in press.

[37] A. F. Shchepetkin and J. C. McWilliams. A correction note for "ocean forecasting in terrain-following coordinates: formulation and skill assessment of the regional ocean modeling system". *Ocean Modeling*, 2008. submitted.

[38] P. K. Smolarkiewicz. A simple positive definite advection scheme with small implicit diffusion. *Mon. Wea. Rev.*, 111:479–486, 1983.

[39] P. K. Smolarkiewicz. A fully multidimensional positive definite advection transport algorithm with small implicit diffusion. *J. Comp. Phys.*, 54:325–362, 1984.

[40] P. K. Smolarkiewicz and T. L. Clark. The multidimensional positive definite advection transport algorithm: further development and applications. *J. Comp. Phys.*, 67:396–438, 1986.

[41] P. K. Smolarkiewicz and W. W. Grabowski. The multidimensional positive definite advection transport algorithm: non-oscillatory option. *J. Comp. Phys.*, 86:355–375, 1990.

[42] Y. Song and D. B. Haidvogel. A semi-implicit ocean circulation model using a generalized topography-following coordinate system. *J. Comp. Phys.*, 115(1):228–244, 1994.

[43] M. Steele, G. L. Mellor, and M. G. McPhee. Role of the molecular sublayer in the melting or freezing of sea ice. *J. Phys. Oceanogr.*, 19:139–147, 1989.

[44] J. Thuburn. Multidimensional flux-limited advection schemes. *J. Comp. Phys.*, 123:74–83, 1996.

[45] L. Umlauf and H. Burchard. A generic length-scale equation for geophysical turbulence models. *J. Marine Res.*, 61:235–265, 2001.

[46] J. Wilkin and K. Hedstrom. User's manual for an orthogonal curvilinear grid-generation package. Institute for Naval Oceanography, 1991.