OCS Study MMS 2009-062

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. M07PC13368

OCS Study MMS 2009-062

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

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

Nov 2009

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 **M07PC13368** 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, Hernan Arango and John Warner, with many, many other contributors. I am 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. I am 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 Paul and I 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).

This work was supported in part by a grant of HPC resources from the Arctic Region Supercomputing Center and the DoD High Performance Computing Modernization Program.

Development and testing of the ROMS model has been funded by many, including the USGS Coastal and Marine Program, the Office of Naval Research, the National Ocean Partnership Program...

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 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. [39], Mellor-Yamada [53], or generic length scale (GLS) [82] 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 [65] (Network Common Data Format), requires the NetCDF library.
- Options include serial, parallel with MPI, and parallel with OpenMP.

The above list hasn'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.

Chapter 2 has some information on getting started with ROMS. Chapters 3 and 4 describe the model physics and numerical techniques and contain information from Shchepetkin and McWilliams [72] and Haidvogel et al. [22]. Chapter 5 describes the ice equations and Chapter 6 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 7, including a discussion of a few example applications. Chapter 8 describes Hernan Arango's plotting programs **cnt**, **ccnt**, **sec**, and **csec**.

2 Getting started

2.1 myroms.org

Starting off with ROMS is not the easiest thing to do, and it just seems to be getting more complex as time goes by. There are some resources, however, beginning with the electronic home for ROMS users at www.myroms.org. Go to register, which gives you access to the subversion server for the code and to the discussion forum for all things ROMS. There is also a wiki, a bug tracking system, and even a developer blog.

The wiki contains parts of this manual, but the nature of wikis is that they can be more fluid, with more authors, than a static document such as this. Dave Robertson (robertson@marine.rutgers.edu) is the one to talk to if you would like to contribute to the wiki.

2.2 Prerequisites

As mentioned in Chapter 1, ROMS has some external requirements. These are:

- UNIX or UNIX-like environment, such as Cygwin.
- A Fortran 90 compiler.
- The NetCDF library compiled with the above compiler, including the Fortran 90 interface.
- svn, the subversion revision control software. See Appendix I and the ROMS wiki.
- Gnu make version 3.81 or higher. Appendix G contains more than you ever wanted to know about this software.
- A C preprocessor—the one from gnu with the **-traditional** flag works well. See Appendix F.
- The Perl scripting language.
- Matlab is optional, but it is a common tool for pre- and post-processing of ROMS files.

Make sure you've got the right environment before attempting to download or compile ROMS.

2.3 Acquiring the ROMS code

The main ROMS code is available for download via **svn** at https://www.myroms.org/svn/src/. The version of the model described in this document is a merger between ROMS 3.2 and a sea-ice model. The sea ice code is a branch off a different repository and requires special access—contact Dave Robertson (as above) for more information.

ROMS comes with several cases all ready to go at the flip of a switch. Try these out first and get to understand how they are set up.

- §2.4 describes how to pick the cases and set up the build environment.
- §6.7 lists all the ROMS options that can be added to your case.
- §6.5 lists the fields which can be provided to ROMS via analytic expressions.
- §7.1.12 lists the input parameters ROMS reads from a text file at run time.
- Chapters 6 and 7 are meant to be informative for the simple and not-so-simple cases. If that isn't the case, please let me know.

In addition to this manual, there are some other ROMS resources:

- You may be best served by going to the ROMS wiki which includes sections called Getting Started and Tutorials.
- Don't be afraid to use the forum. It has everything from employment opportunities to debugging help. Posting there can get you help from one of several people, improving your odds of success over private emails. Registered users get an email once a day about new postings, so you might have to wait a day (or more) for a reply.
- There have been ROMS meetings and classes in which a tutorial session is included as part of the program.
- There are various resources from these online—I've heard good things about the tutorials from Manu Di Lorenzo.

2.4 Compiling ROMS

2.4.1 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 §G). 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 §6.5, 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 §G.2.3 and §2.4.2.

Local File Options 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.

2.4.2 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 §G.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
```

2.4.3 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. Another option of keeping track of the user-defined choices in a **build script**. The advantage is that updates to the **build scripts** are less frequent than updates to the **makefile**. 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.

2.5 Running ROMS

ROMS expects to read a number of variables from an ASCII file (details of the file are in §7.1.12). For serial or OpenMP execution, the syntax is:

```
oceanS (or ocean0) < ocean.in > roms.out &
```

while MPI execution requires:

```
oceanM ocean.in > roms.out &
```

so that each process can read the file.

Realistically, you would only want to run relatively small applications such as **UPWELLING** interactively on the command line as shown here. Also, for either of the parallel options, you will have to provide some information to ROMS and to the operating system about how many threads or processes to use. Parallel computers may also have some sort of batch queuing system in place in which you would submit a job script. I have easy access to two Linux clusters with differing details in the systems, requiring different job scripts. Some MPI environments require that you submit your job with:

```
cd $PBS_O_WORKDIR
mpirun -np 32 ./oceanM ocean_benchmark3.in
```

while others need:

```
aprun -np 32 ./oceanM ocean_benchmark3.in
```

You just have to find out from the locals.

If all goes according to plan, ROMS will create both a collection of NetCDF files and a verbose text file on standard out. Chapter 8 describes one way to view the gridded NetCDF files. Other tools that have been used include Matlab, NCL, and Python.

If things don't go according to plan, the text output file is your friend. Examine it carefully. If it fails on the **UPWELLING** problem, you can compare your output to that in §7.2.8.

2.6 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. Some 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 as seen in the coastal oceans. Changes to the code have allowed us to expand the well-behaved range of depths and the range of values for **THETA_S**, plus there are some new vertical coordinate options. I used to give guidelines on "reasonable" values for **THETA_S**, but I no longer know what's reasonable.
- σ-coordinates have long had a bad reputation because errors in the pressure gradient terms can lead to spurious currents. These errors are must less troublesome than in the past due to code improvements and can also be controlled with some smoothing of the bathymetry. 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. You may want to explore a matlab tool for minimally smoothing the bathymetry found at: http://www.liga.ens.fr/~dutour/Bathymetry/index.html.
- There remain bugs in ROMS. If you find any, please report them on the forum and/or the bug tracking system at myroms.org.

3 Ocean Model Formulation

3.1 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 3.1. 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 §4.10.1.

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

3.2 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 3.2. 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.

3.3 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.

3.4 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 [60] and Freeman et al. [17]). 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 [70] 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:

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

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

3.5 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 [2]) 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.

4 Numerical Solution Technique

4.1 Vertical and horizontal discretization

4.1.1 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 [2]).

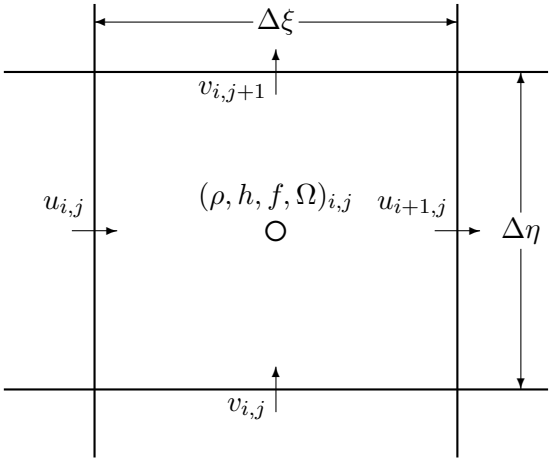

Figure 1: Placement of variables on an Arakawa C grid

4.1.2 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.

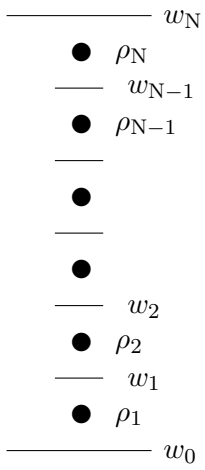

Figure 2: Placement of variables on staggered vertical grid

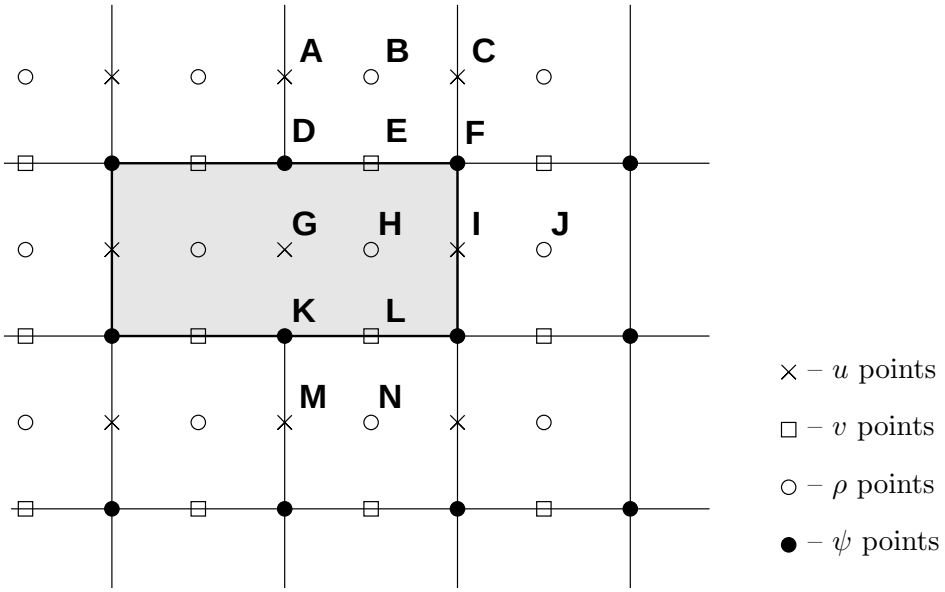

Figure 3: Masked region within the domain

4.2 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 external to ROMS and is usually accomplished in **Matlab**; this section describes how the masking affects the computation of the various terms in the equations of motion.

4.2.1 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.

4.2.2 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.

4.2.3 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.
- The ice component now checks for dry cells when computing the ice rheology.

4.3 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 vary 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 [71]). The time stepping schemes are also listed in Table 4.3 and described in detail in [70] and [72]; the relevant ones are described in Appendix A.

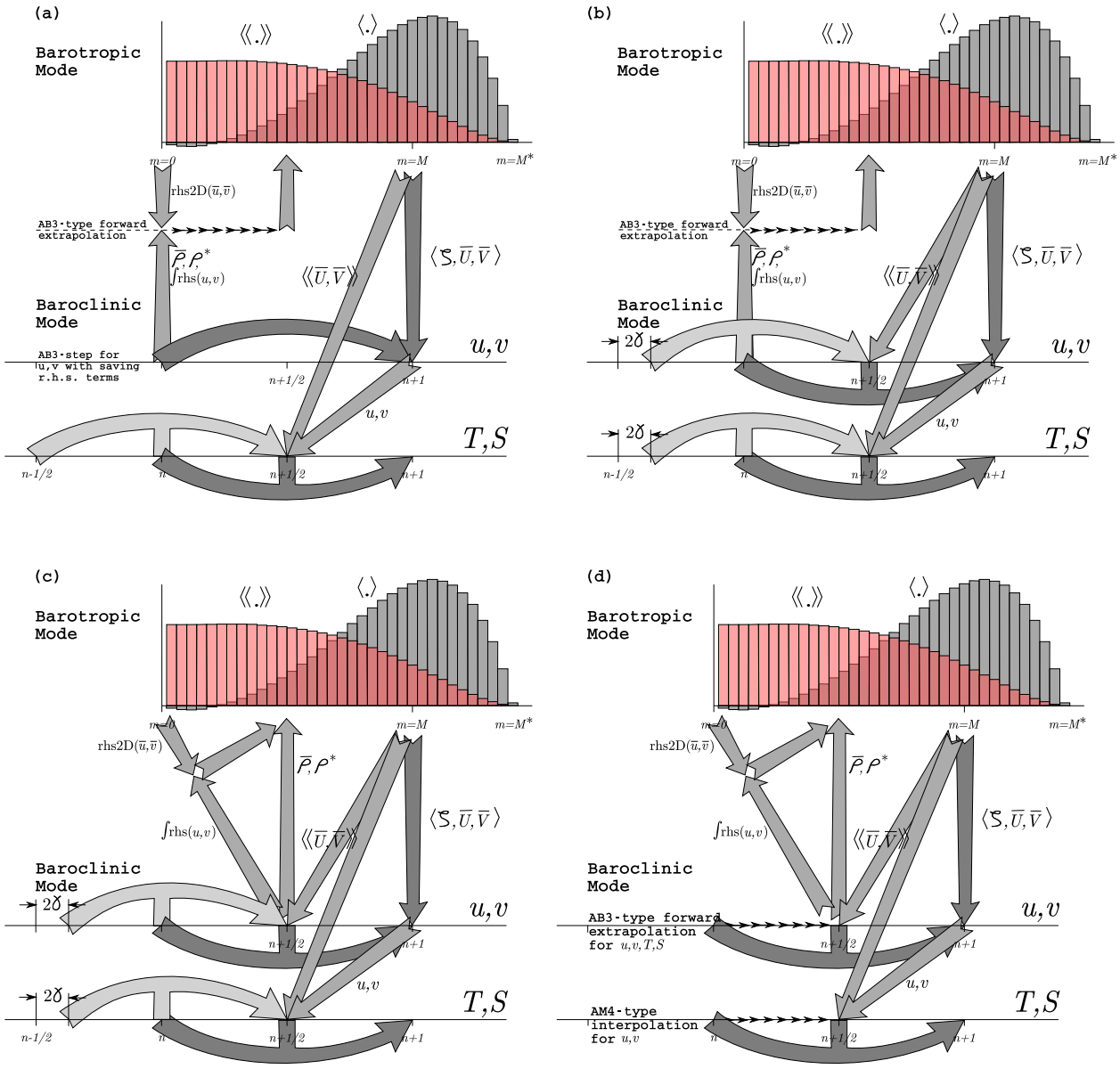

Figure 4: Diagrams of the time stepping and mode coupling used in various ROMS versions. (a) Rutgers University ROMS (from myroms.org), (b) ROMS AGRIF, (c) UCLA ROMS, described in [70], (d) non-hydrostatic ROMS ([33]). In all, the curved arrows update the 3-D fields; those with "pillars" are leapfrog 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	[26]	[23]	[58]	[70]	[33]
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.

4.4 Conservation properties

From Shchepetkin and McWilliams (2005) [70], 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 $\overline{(\quad)}^{\sigma}$ 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 [2]; 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.

4.5 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.

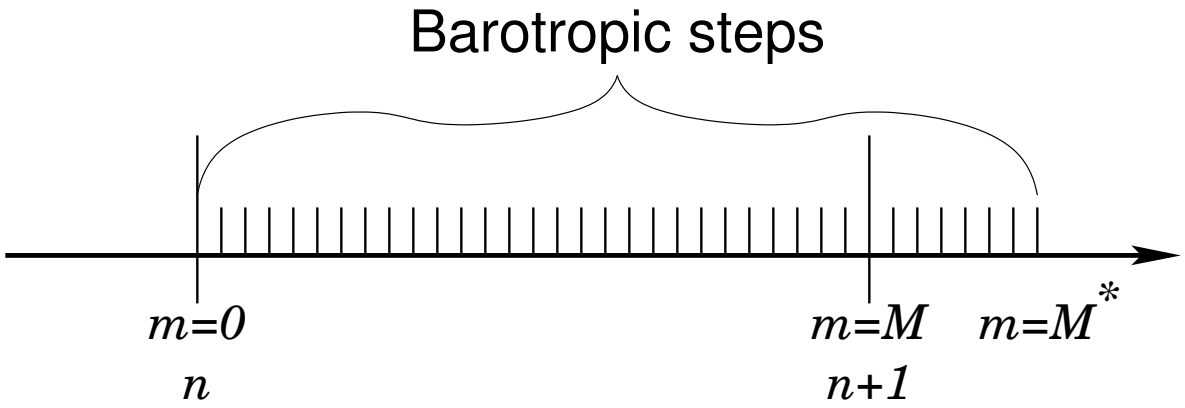

Figure 5: The split time stepping used in the model.

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 $(n + \mathbf{dtfast} \times M^\star)$, 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 ([70]) 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

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

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.

4.6 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

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

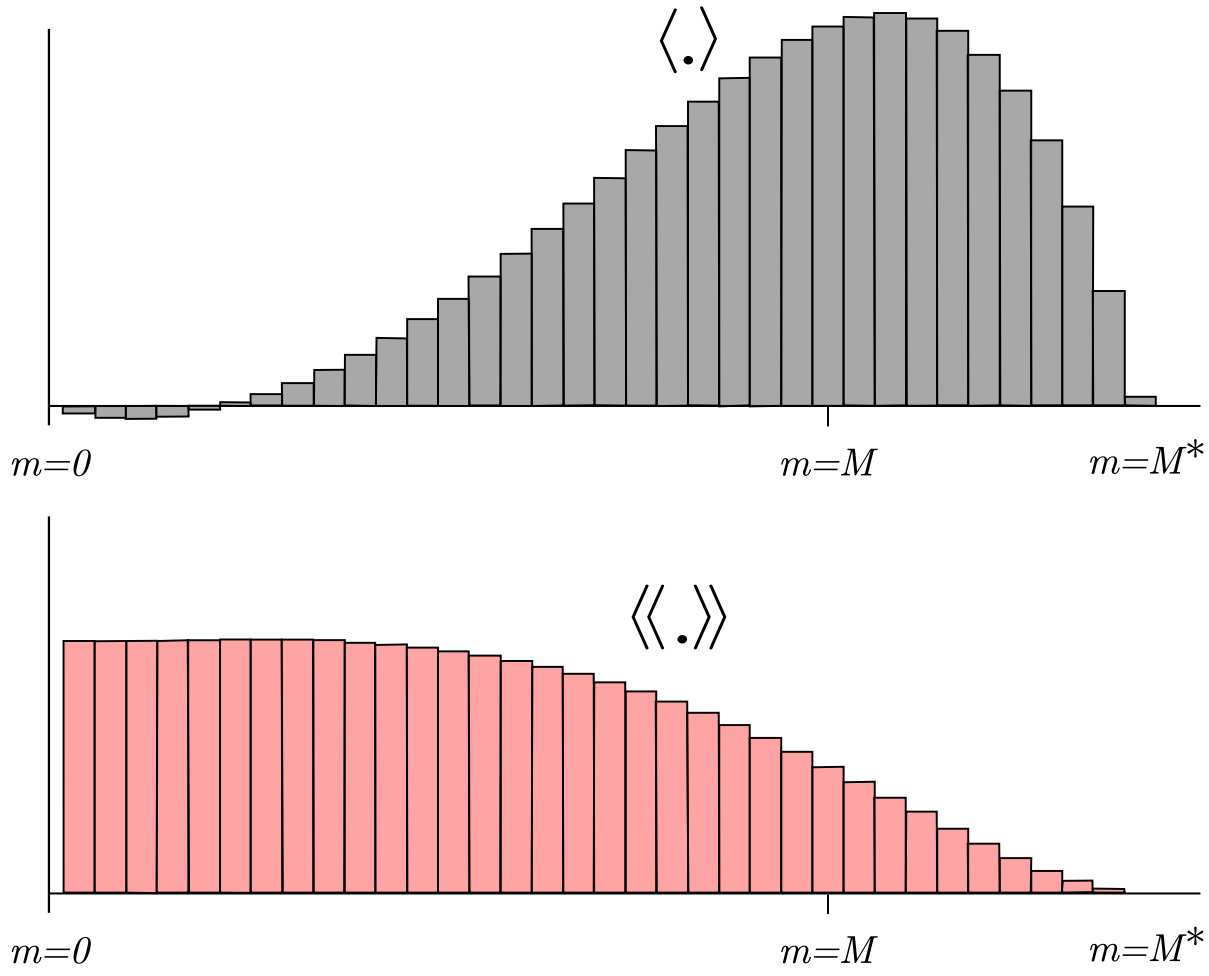

Figure 6: Weights for the barotropic time stepping. The upper panel shows the primary weights, centered at time $n+1$, while the lower panel shows the secondary weights weights, centered at time $n+\frac{1}{2}$.

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, [70]).

4.7 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 §4.12 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 §4.5. 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 §4.12.

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.

4.8 Advection schemes

The advection of a tracer C has an equation of the form

$$\frac{\partial}{\partial t}\frac{H_z C}{mn} = -\frac{\partial}{\partial\xi}F^\xi - \frac{\partial}{\partial\eta}F^\eta - \frac{\partial}{\partial\sigma}F^\sigma, \tag{46}$$

where we have introduced the advective fluxes:

$$F^{\xi} = \frac{H_z u C}{n} \tag{47}$$

$$F^{\eta} = \frac{H_z v C}{m} \tag{48}$$

$$F^{\sigma} = \frac{H_z \Omega C}{mn}. \tag{49}$$

4.8.1 Second-order Centered

The simplest form of the advective fluxes is the centered second-order:

$$F^{\xi} = \frac{\overline{H_z}^{\xi} u \overline{C}^{\xi}}{\overline{n}^{\xi}} \tag{50}$$

$$F^{\eta} = \frac{\overline{H_z}^{\eta} v \overline{C}^{\eta}}{\overline{m}^{\eta}} \tag{51}$$

$$F^{\sigma} = \frac{\overline{H_z}^{\sigma} \Omega \overline{C}^{\sigma}}{mn}. \tag{52}$$

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. ROMS provides alternative advection schemes with better behavior in many situations, but retains this one for comparison purposes.

4.8.2 Fourth-order Centered

The barotropic advection is centered fourth-order unless you specifically pick centered second-order as your horizontal advection scheme. To get fourth-order, create gradient terms:

$$G^{\xi} = \overline{\left(\frac{\partial C}{\partial \xi}\right)}^{\xi} \tag{53}$$

$$G^{\eta} = \overline{\left(\frac{\partial C}{\partial \eta}\right)}^{\eta} \tag{54}$$

$$G^{\sigma} = \overline{\left(\frac{\partial C}{\partial \sigma}\right)}^{\sigma}. \tag{55}$$

The fluxes now become:

$$F^{\xi} = \frac{\overline{H_z}^{\xi}}{\overline{n}^{\xi}} u \left(\overline{C}^{\xi} - \frac{1}{3}\frac{\partial G^{\xi}}{\partial \xi}\right) \tag{56}$$

$$F^{\eta} = \frac{\overline{H_z}^{\eta}}{\overline{m}^{\eta}} v \left(\overline{C}^{\eta} - \frac{1}{3}\frac{\partial G^{\eta}}{\partial \eta}\right) \tag{57}$$

$$F^{\sigma} = \frac{\overline{H_z}^{\sigma}}{mn} \Omega \left(\overline{C}^{\sigma} - \frac{1}{3}\frac{\partial G^{\sigma}}{\partial \sigma}\right). \tag{58}$$

4.8.3 Fourth-order Akima

An alternate fourth-order algorithm is that by Akima:

$$G^{\xi} = 2\frac{\partial C}{\partial \xi}_{i}\frac{\partial C}{\partial \xi}_{i+1}\Bigg/\left(\frac{\partial C}{\partial \xi}_{i}+\frac{\partial C}{\partial \xi}_{i+1}\right) \tag{59}$$

$$G^{\eta} = 2\frac{\partial C}{\partial \eta}_{j}\frac{\partial C}{\partial \eta}_{j+1}\Bigg/\left(\frac{\partial C}{\partial \eta}_{j}+\frac{\partial C}{\partial \eta}_{j+1}\right) \tag{60}$$

$$G^{\sigma} = 2\frac{\partial C}{\partial \sigma}_{k}\frac{\partial C}{\partial \sigma}_{k-1}\Bigg/\left(\frac{\partial C}{\partial \sigma}_{k}+\frac{\partial C}{\partial \sigma}_{k-1}\right) \tag{61}$$

$$\tag{62}$$

With the fluxes as in 56–58.

4.8.4 Third-order Upwind

There is a class of third-order upwind advection schemes, both one-dimensional (Leonard [42]) and two-dimensional (Rasch [63] and Shchepetkin and McWilliams [68]). This scheme is known as UTOPIA (Uniformly Third-Order Polynomial Interpolation Algorithm). Applying flux limiters to UTOPIA is explored in Thuburn [80], although it is not implemented in ROMS. The two-dimensional formulation in Rasch contains terms of order u^2C and u^3C, including cross terms (uvC). The terms which are nonlinear in velocity have been dropped in ROMS, leaving one extra upwind term in the computation of the advective fluxes:

$$F^{\xi} = \frac{H_z u}{n}\left(C - \gamma\frac{\partial^2 C}{\partial \xi^2}\right) \tag{63}$$

$$F^{\eta} = \frac{H_z v}{m}\left(C - \gamma\frac{\partial^2 C}{\partial \eta^2}\right) \tag{64}$$

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:

$$F^{\xi}_{i,j,k} = \frac{\overline{H_z}^{\xi}}{\overline{n}^{\xi}}\left[\max(0,u_{i,j,k})C_{i-1,j,k} + \min(0,u_{i,j,k})C_{i,j,k}\right]. \tag{65}$$

The value of γ in the model is $\frac{1}{8}$ while that in Rasch [63] 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:

$$F^{s} = \frac{H_z w}{mn}\left[-\frac{1}{16}C_{i,j,k-1} + \frac{9}{16}C_{i,j,k} + \frac{9}{16}C_{i,j,k+1} - \frac{1}{16}C_{i,j,k+2}\right] \tag{66}$$

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:

$$F^{\xi} = \left(u - \gamma\frac{\partial^2 u}{\partial \xi^2}\right)\left[\frac{H_z u}{n} - \gamma\frac{\partial^2}{\partial \xi^2}\left(\frac{H_z u}{n}\right)\right] \tag{67}$$

$$F^{\eta} = \left(u - \gamma\frac{\partial^2 u}{\partial \eta^2}\right)\left[\frac{H_z v}{m} - \gamma\frac{\partial^2}{\partial \xi^2}\left(\frac{H_z v}{m}\right)\right] \tag{68}$$

$$F^{\sigma} = \frac{H_z w}{mn}\left[-\frac{1}{16}u_{i,j,k-1} + \frac{9}{16}u_{i,j,k} + \frac{9}{16}u_{i,j,k+1} - \frac{1}{16}u_{i,j,k+2}\right] \tag{69}$$

while for the v-velocity we have:

$$F^{\xi} = \left(v - \gamma \frac{\partial^2 v}{\partial \xi^2} \right) \left[\frac{H_z u}{n} - \gamma \frac{\partial^2}{\partial \eta^2} \left(\frac{H_z u}{n} \right) \right] \tag{70}$$

$$F^{\eta} = \left(v - \gamma \frac{\partial^2 v}{\partial \eta^2} \right) \left[\frac{H_z v}{m} - \gamma \frac{\partial^2}{\partial \eta^2} \left(\frac{H_z v}{m} \right) \right] \tag{71}$$

$$F^{\sigma} = \frac{H_z w}{mn} \left[-\frac{1}{16} v_{i,j,k-1} + \frac{9}{16} v_{i,j,k} + \frac{9}{16} v_{i,j,k+1} - \frac{1}{16} v_{i,j,k+2} \right] \tag{72}$$

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

4.9 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:

$$\frac{\partial}{\partial \xi} \left(\frac{H_z u}{n} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z v}{m} \right) + \frac{\partial}{\partial \sigma} \left(\frac{H_z \Omega}{mn} \right) - \frac{\partial}{\partial \xi} \left(\frac{D\overline{u}}{n} \right) - \frac{\partial}{\partial \eta} \left(\frac{D\overline{v}}{m} \right) = 0. \tag{73}$$

Solving for $H_z \Omega / mn$ and using the semi-discrete notation of §4.4 we obtain:

$$\frac{H_z \Omega}{mn} = \int \left[\delta_{\xi} \left(\frac{\overline{u} \overline{D}^{\xi}}{\overline{n}^{\xi}} \right) + \delta_{\eta} \left(\frac{\overline{v} \overline{D}^{\eta}}{\overline{m}^{\eta}} \right) - \delta_{\xi} \left(\frac{u \overline{H_z}^{\xi}}{\overline{n}^{\xi}} \right) - \delta_{\eta} \left(\frac{v \overline{H_z}^{\eta}}{\overline{m}^{\eta}} \right) \right] d\sigma. \tag{74}$$

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. ROMS 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 [32]. 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.

4.10 Horizontal mixing

In Chapter 3, the diffusive terms were written simply as $\mathcal{D}_u, \mathcal{D}_v, \mathcal{D}_T$, and $\mathcal{D}_S$. The vertical component of these terms was described in §4.12. Here we describe the ROMS options for representing the horizontal component of these terms.

4.10.1 Deviatory stress tensor

Note: this material was copied from the wiki, where it was contributed by Hernan Arango. The horizontal components of the divergence of the stress tensor (Wajsowicz, 1993 [83]) in nondimesional, orthogonal curvilinear coordinates (ξ, η, s) with dimensional, spatially-varying metric factors ($\frac{1}{m}$, $\frac{1}{n}$, H_z) and velocity components (u, v, ωH_z) are given by:

$$F^u \equiv \widehat{\xi} \cdot (\nabla \cdot \vec{\sigma}) = \frac{mn}{H_z}\left[\frac{\partial}{\partial \xi}\left(\frac{H_z \sigma_{\xi\xi}}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z \sigma_{\xi\eta}}{m}\right) + \frac{\partial}{\partial s}\left(\frac{\sigma_{\xi s}}{mn}\right) + \right. \tag{75}$$

$$\left. H_z \sigma_{\xi\eta} \frac{\partial}{\partial \eta}\left(\frac{1}{m}\right) - H_z \sigma_{\eta\eta} \frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - \frac{1}{n}\sigma_{ss}\frac{\partial H_z}{\partial \xi}\right] \tag{76}$$

$$F^v \equiv \widehat{\eta} \cdot (\nabla \cdot \vec{\sigma}) = \frac{mn}{H_z}\left[\frac{\partial}{\partial \xi}\left(\frac{H_z \sigma_{\eta\xi}}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z \sigma_{\eta\eta}}{m}\right) + \frac{\partial}{\partial s}\left(\frac{\sigma_{\eta s}}{mn}\right) + \right. \tag{77}$$

$$\left. H_z \sigma_{\eta\xi} \frac{\partial}{\partial \xi}\left(\frac{1}{n}\right) - H_z \sigma_{\xi\xi} \frac{\partial}{\partial \eta}\left(\frac{1}{m}\right) - \frac{1}{m}\sigma_{ss}\frac{\partial H_z}{\partial \eta}\right] \tag{78}$$

where

$$\sigma_{\xi\xi} = (A_M + \nu)\, e_{\xi\xi} + (\nu - A_M)\, e_{\eta\eta}, \tag{79}$$

$$\sigma_{\eta\eta} = (\nu - A_M)\, e_{\xi\xi} + (A_M + \nu)\, e_{\eta\eta}, \tag{80}$$

$$\sigma_{ss} = 2\,\nu\, e_{ss}, \tag{81}$$

$$\sigma_{\xi\eta} = \sigma_{\eta\xi} = 2\, A_M\, e_{\xi\eta}, \tag{82}$$

$$\sigma_{\xi s} = 2\, K_M\, e_{\xi s}, \tag{83}$$

$$\sigma_{\eta s} = 2\, K_M\, e_{\eta s}, \tag{84}$$

and the strain field is:

$$e_{\xi\xi} = m\frac{\partial u}{\partial \xi} + mnv\frac{\partial}{\partial \eta}\left(\frac{1}{m}\right), \tag{85}$$

$$e_{\eta\eta} = n\,\frac{\partial v}{\partial \eta} + mnu\frac{\partial}{\partial \xi}\left(\frac{1}{n}\right), \tag{86}$$

$$e_{ss} = \frac{1}{H_z}\frac{\partial(\omega H_z)}{\partial s} + \frac{m}{H_z}u\frac{\partial H_z}{\partial \xi} + \frac{n}{H_z}v\frac{\partial H_z}{\partial \eta}, \tag{87}$$

$$2\, e_{\xi\eta} = \frac{m}{n}\frac{\partial(nv)}{\partial \xi} + \frac{n}{m}\frac{\partial(mu)}{\partial \eta}, \tag{88}$$

$$2\, e_{\xi s} = \frac{1}{mH_z}\frac{\partial(mu)}{\partial s} + mH_z\frac{\partial \omega}{\partial \xi}, \tag{89}$$

$$2\, e_{\eta s} = \frac{1}{nH_z}\,\frac{\partial(nv)}{\partial s}\, + n\, H_z\frac{\partial \omega}{\partial \eta}. \tag{90}$$

Here, $A_M(\xi, \eta)$ and $K_M(\xi, \eta, s)$ are the spatially varying horizontal and vertical viscosity coefficients, respectively, and ν is another (very small, often neglected) horizontal viscosity coefficient. Notice that because of the generalized terrain-following vertical coordinates of ROMS, we need to transform the horizontal partial derivatives from constant "z-"surfaces to constant "s-"surfaces. And the vertical metric or level thickness is the Jacobian of the transformation, $H_z = \frac{\partial z}{\partial s}$. Also in these models, the "vertical" velocity is computed as $\frac{\omega H_z}{mn}$ and has units of m^3/s.

4.10.2 Transverse stress tensor

Assuming transverse isotropy, as in Sadourny and Maynard (1997) [66] and Griffies and Hallberg (2000) [21], the deviatoric stress tensor can be split into vertical and horizontal sub-tensors. The horizontal (or transverse) sub-tensor is symmetric, it has a null trace, and it possesses axial symmetry in the local vertical direction. Then, transverse stress tensor can be derived from eq. (76) and (76), yielding

$$H_z F^u = n^2 m \frac{\partial}{\partial \xi}\left(\frac{H_z F^{u\xi}}{n}\right) + m^2 n \frac{\partial}{\partial \eta}\left(\frac{H_z F^{u\eta}}{m}\right) \tag{91}$$

$$H_z F^v = n^2 m \frac{\partial}{\partial \xi}\left(\frac{H_z F^{v\xi}}{n}\right) + m^2 n \frac{\partial}{\partial \eta}\left(\frac{H_z F^{v\eta}}{m}\right) \tag{92}$$

where

$$F^{u\xi} = \frac{1}{n} A_M \left[\frac{m}{n}\frac{\partial (nu)}{\partial \xi} - \frac{n}{m}\frac{\partial (mv)}{\partial \eta}\right], \tag{93}$$

$$F^{u\eta} = \frac{1}{m} A_M \left[\frac{n}{m}\frac{\partial (mu)}{\partial \eta} + \frac{m}{n}\frac{\partial (nv)}{\partial \xi}\right], \tag{94}$$

$$F^{v\xi} = \frac{1}{n} A_M \left[\frac{m}{n}\frac{\partial (nv)}{\partial \xi} + \frac{n}{m}\frac{\partial (mu)}{\partial \eta}\right], \tag{95}$$

$$F^{v\eta} = \frac{1}{m} A_M \left[\frac{n}{m}\frac{\partial (mv)}{\partial \eta} - \frac{m}{n}\frac{\partial (nu)}{\partial \xi}\right]. \tag{96}$$

Notice the flux form of eq. (91) and (92) and the symmetry between the $F^{u\xi}$ and $F^{v\eta}$ terms which are defined at density points on a C-grid. Similarly, the $F^{u\eta}$ and $F^{v\xi}$ terms are symmetric and defined at vorticity points. These staggering positions are optimal for the discretization of the tensor; it has no computational modes and satisfies first-moment conservation.

The biharmonic friction operator can be computed by applying twice the tensor operator eq. (91) and (92), but with the squared root of the biharmonic viscosity coefficient (Griffies and Hallberg, 2000 [21]). For simplicity and momentum balance, the thickness H_z appears only when computing the second harmonic operator as in Griffies and Hallberg (2000).

4.10.3 Rotated Transverse Stress Tensor

In some applications with tall and steep topography, it will be advantageous to reduce substantially the contribution of the stress tensor eq. (91) and (92) to the vertical mixing when operating along constant s-surfaces. The transverse stress tensor rotated along geopotentials (constant depth) is then given by

$$H_z R^u = n^2 m \frac{\partial}{\partial \xi}\left(\frac{H_z R^{u\xi}}{n}\right) + m^2 n \frac{\partial}{\partial \eta}\left(\frac{H_z R^{u\eta}}{m}\right) + \frac{\partial}{\partial s}\left(R^{us}\right) \tag{97}$$

$$H_z R^v = n^2 m \frac{\partial}{\partial \xi}\left(\frac{H_z R^{v\xi}}{n}\right) + m^2 n \frac{\partial}{\partial \eta}\left(\frac{H_z R^{v\eta}}{m}\right) + \frac{\partial}{\partial s}\left(R^{vs}\right) \tag{98}$$

where

$$R^{u\xi} = \frac{1}{n} A_M \left[\frac{1}{n} \left(m \frac{\partial (nu)}{\partial \xi} - m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nu)}{\partial s} \right) - \frac{1}{m} \left(n \frac{\partial (mv)}{\partial \eta} - n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mv)}{\partial s} \right) \right], \tag{99}$$

$$R^{u\eta} = \frac{1}{m} A_M \left[\frac{1}{m} \left(n \frac{\partial (mu)}{\partial \eta} - n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mu)}{\partial s} \right) + \frac{1}{n} \left(m \frac{\partial (nv)}{\partial \xi} - m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nv)}{\partial s} \right) \right], \tag{100}$$

$$R^{us} = m \frac{\partial z}{\partial \xi} A_M \left[\frac{1}{n} \left(m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nu)}{\partial s} - m \frac{\partial (nu)}{\partial \xi} \right) - \frac{1}{m} \left(n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mv)}{\partial s} - n \frac{\partial (mv)}{\partial \eta} \right) \right] + \tag{101}$$

$$n \frac{\partial z}{\partial \eta} A_M \left[\frac{1}{m} \left(n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mu)}{\partial s} - n \frac{\partial (mu)}{\partial \eta} \right) + \frac{1}{n} \left(m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nv)}{\partial s} - m \frac{\partial (nv)}{\partial \xi} \right) \right], \tag{102}$$

$$R^{v\xi} = \frac{1}{n} A_M \left[\frac{1}{n} \left(m \frac{\partial (nv)}{\partial \xi} - m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nv)}{\partial s} \right) + \frac{1}{m} \left(n \frac{\partial (mu)}{\partial \eta} - n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mu)}{\partial s} \right) \right], \tag{103}$$

$$R^{v\eta} = \frac{1}{m} A_M \left[\frac{1}{m} \left(n \frac{\partial (mv)}{\partial \eta} - n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mv)}{\partial s} \right) - \frac{1}{n} \left(m \frac{\partial (nu)}{\partial \xi} - m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nu)}{\partial s} \right) \right], \tag{104}$$

$$R^{vs} = m \frac{\partial z}{\partial \xi} A_M \left[\frac{1}{n} \left(m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nv)}{\partial s} - m \frac{\partial (nv)}{\partial \xi} \right) + \frac{1}{m} \left(n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mu)}{\partial s} - n \frac{\partial (mu)}{\partial \eta} \right) \right] + \tag{105}$$

$$n \frac{\partial z}{\partial \eta} A_M \left[\frac{1}{m} \left(n \frac{\partial z}{\partial \eta} \frac{1}{H_z} \frac{\partial (mv)}{\partial s} - n \frac{\partial (mv)}{\partial \eta} \right) - \frac{1}{n} \left(m \frac{\partial z}{\partial \xi} \frac{1}{H_z} \frac{\partial (nu)}{\partial s} - m \frac{\partial (nu)}{\partial \xi} \right) \right]. \tag{106}$$

Notice that transverse stress tensor remains invariant under coordinate transformation. The rotated tensor (97) and (98) retains the same properties as the unrotated tensor (91) and (92). The additional terms that arise from the slopes of s-surfaces along geopotentials are discretized using a modified version of the triad approach of Griffies *et al.* (1998) [20].

4.10.4 Biharmonic

Biharmonic mixing has less of a physical justification and is used for damping of numerical noise at the $2\Delta x$ scale. The biharmonic operator is $\nabla^4 = \nabla^2 \nabla^2$; the corresponding term is computed using a temporary variable Y:

$$Y = \frac{mn}{H_z} \left[\frac{\partial}{\partial \xi} \left(\frac{\nu_4 H_z m}{n} \frac{\partial \psi}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left(\frac{\nu_4 H_z n}{m} \frac{\partial \psi}{\partial \eta} \right) \right] \tag{107}$$

and is

$$- \left[\frac{\partial}{\partial \xi} \left(\frac{H_z m}{n} \frac{\partial Y}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left(\frac{H_z n}{m} \frac{\partial Y}{\partial \eta} \right) \right] \tag{108}$$

where ψ is any of u, v, T, and S. Note that u and v are treated as independent scalar quantities rather than as a vector. The complete Laplacian operator on a vector quantity $\vec{u}$ contains additional terms, including v terms in the u equation and vice versa. These extra terms were found to be small in a test problem and have been left out of the model.

4.11 Vertical mixing schemes

ROMS contains a variety of methods for setting the vertical viscous and diffusive coefficients. The choices range from simply choosing fixed values to the KPP, generic lengthscale (GLS) and Mellor-Yamada turbulence closure schemes. See Large [38] for a review of surface ocean mixing schemes. Many schemes have a background molecular value which is used when the turbulent processes are assumed to be small (such as in the interior).

4.11.1 Mellor-Yamada

One of the more popular closure schemes is that of Mellor and Yamada [53], [54]. They actually present a hierarchy of closures of increasing complexity. ROMS provides only the "Level 2.5" closure with the Galperin et al. [18] modifications as described in Allen et al. [1]. This closure scheme adds two prognostic equations, one for the turbulent kinetic energy ($\frac{1}{2}q^2$) and one for the turbulent kinetic energy times a length scale ($q^2 l$).

The turbulent kinetic energy equation is:

$$\frac{D}{Dt}\left(\frac{q^2}{2}\right) - \frac{\partial}{\partial z}\left[K_q \frac{\partial}{\partial z}\left(\frac{q^2}{2}\right)\right] = P_s + P_b - \xi_d \tag{109}$$

where P_s is the shear production, P_b is the buoyant production and ξ_d is the dissipation of turbulent kinetic energy. These terms are given by

$$P_s = K_m \left[\left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2\right], \tag{110}$$

$$P_b = K_s N^2, \tag{111}$$

$$\xi_d = \frac{q^3}{B_1 l} \tag{112}$$

where B_1 is a constant. One can also add a traditional horizontal Laplacian or biharmonic diffusion ($\mathcal{D}_q$) to the turbulent kinetic energy equation. The form of this equation in the model coordinates becomes

$$\frac{\partial}{\partial t}\left(\frac{H_z q^2}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u q^2}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v q^2}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega q^2}{mn}\right) - \frac{\partial}{\partial s}\left(\frac{K_q}{mnH_z}\frac{\partial q^2}{\partial s}\right) = \frac{2H_z K_m}{mn}\left[\left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2\right] + \frac{2H_z K_s}{mn}N^2 - \frac{2H_z q^3}{mnB_1 l} + \frac{H_z}{mn}\mathcal{D}_q. \tag{113}$$

The vertical boundary conditions are:

$$\begin{aligned}
&\text{top } (z = \zeta(x,y,t)) && \tfrac{H_z\Omega}{mn} = 0 \\
& && \tfrac{K_q}{mnH_z}\tfrac{\partial q^2}{\partial s} = \tfrac{B_1^{2/3}}{\rho_o}\left[\left(\tau_s^\xi\right)^2 + \left(\tau_s^\eta\right)^2\right] \\
& && H_z K_m\left(\tfrac{\partial u}{\partial z}, \tfrac{\partial v}{\partial z}\right) = \tfrac{1}{\rho_o}\left(\tau_s^\xi, \tau_s^\eta\right) \\
& && H_z K_s N^2 = \tfrac{Q}{\rho_o c_P} \\
&\text{and bottom } (z = -h(x,y)) && \tfrac{H_z\Omega}{mn} = 0 \\
& && \tfrac{K_q}{mnH_z}\tfrac{\partial q^2}{\partial s} = \tfrac{B_1^{2/3}}{\rho_o}\left[\left(\tau_b^\xi\right)^2 + \left(\tau_b^\eta\right)^2\right] \\
& && H_z K_m\left(\tfrac{\partial u}{\partial z}, \tfrac{\partial v}{\partial z}\right) = \tfrac{1}{\rho_o}\left(\tau_b^\xi, \tau_b^\eta\right) \\
& && H_z K_s N^2 = 0
\end{aligned}$$

The equation is timestepped much like the model tracer equations, including an implicit solve for the vertical operations and an option for using the third-order upwind advection.

There is also an equation for the turbulent length scale l:

$$\frac{D}{Dt}\left(lq^2\right) - \frac{\partial}{\partial z}\left[K_l \frac{\partial l q^2}{\partial z}\right] = l E_1 (P_s + P_b) - \frac{q^3}{B_1}\tilde{W} \tag{114}$$

where $\tilde{W}$ is the wall proximity function:

$$\tilde{W} = 1 + E_2 \left(\frac{l}{kL} \right)^2 \tag{115}$$

$$L^{-1} = \frac{1}{\zeta - z} + \frac{1}{H + z} \tag{116}$$

The form of this equation in the model coordinates becomes

$$\frac{\partial}{\partial t}\left(\frac{H_z q^2 l}{mn}\right) + \frac{\partial}{\partial \xi}\left(\frac{H_z u q^2 l}{n}\right) + \frac{\partial}{\partial \eta}\left(\frac{H_z v q^2 l}{m}\right) + \frac{\partial}{\partial s}\left(\frac{H_z \Omega q^2 l}{mn}\right) - \frac{\partial}{\partial s}\left(\frac{K_q}{mnH_z}\frac{\partial q^2 l}{\partial s}\right) = \frac{H_z}{mn} l E_1 (P_s + P_b) - \frac{H_z q^3}{mnB_1}\tilde{W} + \frac{H_z}{mn}\mathcal{D}_{ql}. \tag{117}$$

where $\mathcal{D}_{ql}$ is the horizontal diffusion of the quantity $q^2 l$.

Given these solutions for q and l, the vertical viscosity and diffusivity coefficients are:

$$K_m = qlS_m + K_{m_{\text{background}}} \tag{118}$$

$$K_s = qlS_h + K_{s_{\text{background}}} \tag{119}$$

$$K_q = qlS_q + K_{q_{\text{background}}} \tag{120}$$

and the stability coefficients S_m, S_h and S_q are found by solving

$$S_s \left[1 - (3A_2B_2 + 18A_1A_2)G_h\right] = A_2 \left[1 - 6A_1 B_1^{-1}\right] \tag{121}$$

$$S_m \left[1 - 9A_1A_2G_h\right] - S_s \left[G_h(18A_1^2 + 9A_1A_2)G_h\right] = A_1 \left[1 - 3C_1 - 6A_1B_1^{-1}\right] \tag{122}$$

$$G_h = \min(-\frac{l^2N^2}{q^2}, 0.028). \tag{123}$$

$$S_q = 0.41 S_m \tag{124}$$

The constants are set to $(A_1, A_2, B_1, B_2, C_1, E_1, E_2) = (0.92, 0.74, 16.6, 10.1, 0.08, 1.8, 1.33)$. The quantities q^2 and $q^2 l$ are both constrained to be no smaller than 10^{-8} while l is set to be no larger than $0.53q/N$.

4.11.2 The Large, McWilliams and Doney parameterization

The vertical mixing parameterization introduced by Large, McWilliams and Doney [39] is a versatile first order scheme which has been shown to perform well in open ocean settings. Its design facilitates experimentation with additional or modified representations of specific turbulent processes.

Surface boundary layer The Large, McWilliams and Doney scheme (LMD) matches separate parameterizations for vertical mixing of the surface boundary layer and the ocean interior. A formulation based on boundary layer similarity theory is applied in the water column above a calculated boundary layer depth h_{sbl}. This parameterization is then matched at the interior with schemes to account for local shear, internal wave and double diffusive mixing effects.

Viscosity and diffusivities at model levels above a calculated surface boundary layer depth (h_{sbl}) are expressed as the product of the length scale h_{sbl}, a turbulent velocity scale w_x and a non-dimensional shape function.

$$\nu_x = h_{sbl} w_x(\sigma) G_x(\sigma) \tag{125}$$

where σ is a non-dimensional coordinate ranging from 0 to 1 indicating depth within the surface boundary layer. The x subscript stands for one of momentum, temperature and salinity.

Surface Boundary layer depth The boundary layer depth h_{sbl} is calculated as the minimum of the Ekman depth, estimated as,

$$h_e = 0.7u_*/f \tag{126}$$

(where u_* is the friction velocity $u_* = \sqrt{\tau_x^2 + \tau_y^2}/\rho$), the Monin-Obukhov depth:

$$L = u_*^3/(\kappa B_f) \tag{127}$$

(where $\kappa = 0.4$ is von Karman's contant and B_f is the surface buoyancy flux), and the shallowest depth at which a critical bulk Richardson number is reached. The critical bulk Richardson number (Ri_c) is typically in the range 0.25–0.5. The bulk Richardson number (Ri_b) is calculated as:

$$Ri_b(z) = \frac{(B_r - B(d))d}{|\vec{V}_r - \vec{V}(d)|^2 + {V_t}^2(d)} \tag{128}$$

where d is distance down from the surface, B is the buoyancy, B_r is the buoyancy at a near surface reference depth, $\vec{V}$ is the mean horizontal velocity, $\vec{V}_r$ the velocity at the near surface reference depth and V_t is an estimate of the turbulent velocity contribution to velocity shear.

The turbulent velocity shear term in this equation is given by LMD as,

$$V_t^2(d) = \frac{C_v(-\beta_T)^{1/2}}{Ri_c\kappa}(c_s\epsilon)^{-1/2}dNw_s \tag{129}$$

where C_v is the ratio of interior N to N at the entrainment depth, β_T is ratio of entrainment flux to surface buoyancy flux, c_s and ϵ are constants, and w_s is the turbulent velocity scale for scalars. LMD derive (129) based on the expected behavior in the pure convective limit. The empirical rule of convection states that the ratio of the surface buoyancy flux to that at the entrainment depth be a constant. Thus the entrainment flux at the bottom of the boundary layer under such conditions should be independent of the stratification at that depth. Without a turbulent shear term in the denominator of the bulk Richardson number calculation, the estimated boundary layer depth is too shallow and the diffusivity at the entrainment depth is too low to obtain the necessary entrainment flux. Thus by adding a turbulent shear term proportional to the stratification in the denominator, the calculated boundary layer depth will be deeper and will lead to a high enough diffusivity to satisfy the empirical rule of convection.

turbulent velocity scale To estimate w_x (where x is m - momentum *or* s - any scalar) throughout the boundary layer, surface layer similarity theory is utilized. Following an argument by Troen and Mahrt [81], Large et al. estimate the velocity scale as

$$w_x = \frac{\kappa u_*}{\phi_x(\zeta)} \tag{130}$$

where ζ is the surface layer stability parameter defined as z/L. ϕ_x is a non-dimensional flux profile which varies based on the stability of the boundary layer forcing. The stability parameter used in this equation is assumed to vary over the entire depth of the boundary layer in stable and neutral conditions. In unstable conditions it is assumed only to vary through the surface layer which is defined as ϵh_{sbl} (where ϵ is set at 0.10) . Beyond this depth ζ is set equal to its value at ϵh_{sbl}.

The flux profiles are expressed as analytical fits to atmospheric surface boundary layer data. In stable conditions they vary linearly with the stability parameter ζ as

$$\phi_x = 1 + 5\zeta \tag{131}$$

In near-neutral unstable conditions common Businger-Dyer forms are used which match with the formulation for stable conditions at $\zeta = 0$. Near neutral conditions are defined as

$$-0.2 \leq \zeta < 0 \tag{132}$$

for momentum and,

$$-1.0 \leq \zeta < 0 \tag{133}$$

for scalars. The non dimensional flux profiles in this regime are,

$$\phi_m = (1 - 16\zeta)^{1/4} \tag{134}$$

$$\phi_s = (1 - 16\zeta)^{1/2} \tag{135}$$

In more unstable conditions ϕ_x is chosen to match the Businger-Dyer forms and with the free convective limit. Here the flux profiles are

$$\phi_m = (1.26 - 8.38\zeta)^{1/3} \tag{136}$$

$$\phi_s = (-28.86 - 98.96\zeta)^{1/3} \tag{137}$$

The shape function The non-dimensional shape function $G(\sigma)$ is a third order polynomial with coefficients chosen to match the interior viscosity at the bottom of the boundary layer and Monin-Obukhov similarity theory approaching the surface. This function is defined as a 3rd order polynomial.

$$G(\sigma) = a_o + a_1\sigma + a_2\sigma^2 + a_3\sigma^3 \tag{138}$$

with the coefficients specified to match surface boundary conditions and to smoothly blend with the interior,

$$a_o = 0 \tag{139}$$

$$a_1 = 1 \tag{140}$$

$$a_2 = -2 + 3\frac{\nu_x(h_{sbl})}{hw_x(1)} + \frac{\partial_x\nu_x(h)}{w_x(1)} + \frac{\nu_x(h)\partial_\sigma w_x(1)}{hw_x^2(1)} \tag{141}$$

$$a_3 = 1 - 2\frac{\nu_x(h_{sbl})}{hw_x(1)} - \frac{\partial_x\nu_x(h)}{w_x(1)} - \frac{\nu_x(h)\partial_\sigma w_x(1)}{hw_x^2(1)} \tag{142}$$

where $\nu_x(h)$ is the viscosity calculated by the interior parameterization at the boundary layer depth.

Countergradient flux term The second term of the LMD scheme's surface boundary layer formulation is the non-local transport term γ which can play a significant role in mixing during surface cooling events. This is a redistribution term included in the tracer equation separate from the diffusion term and is written as

$$-\frac{\partial}{\partial z}K\gamma. \tag{143}$$

LMD base their formulation for non-local scalar transport on a parameterization for pure free convection from Mailhôt and Benoit [46]. They extend this parameterization to cover any unstable surface forcing conditions to give

$$\gamma_T = C_s\frac{\overline{wT_0} + \overline{wT_R}}{w_T(\sigma)h} \tag{144}$$

for temperature and

$$\gamma_S = C_s\frac{\overline{wS_0}}{w_S(\sigma)h} \tag{145}$$

for salinity (other scalar quantities with surface fluxes can be treated similarly). LMD argue that although there is evidence of non-local transport of momentum as well, the form the term would take is unclear so they simply specify $\gamma_m = 0$.

The interior scheme The interior scheme of Large, McWilliams and Doney estimates the viscosity coefficient by adding the effects of several generating mechanisms: shear mixing, double-diffusive mixing and internal wave generated mixing.

$$\nu_x(d) = \nu_x^s + \nu_x^d + \nu_x^w \tag{146}$$

Shear generated mixing The shear mixing term is calculated using a gradient Richardson number formulation, with viscosity estimated as:

$$\nu_x^s = \begin{cases} \nu_0 & Ri_g < 0, \\ \nu_0[1 - (Ri_g/Ri_0)^2]^3 & 0 < Ri_g < Ri_0, \\ 0 & Ri_g > Ri_0. \end{cases} \tag{147}$$

where ν_0 is 5.0×10^{-3}, $Ri_0 = 0.7$.

Double diffusive processes The second component of the interior mixing parameterization represents double diffusive mixing. From limited sources of laboratory and field data LMD parameterize the salt fingering case ($R_\rho > 1.0$)

$$\nu_s^d(R_\rho) = \begin{cases} 1 \times 10^{-4}[1 - (\frac{(R_\rho - 1}{R_\rho^0 - 1})^2)^3 & \text{for } 1.0 < R_\rho < R_\rho^0 = 1.9, \\ 0 & \text{otherwise.} \end{cases} \tag{148}$$

$$\nu_\theta^d(R_\rho) = 0.7\nu_s^d \tag{149}$$

For diffusive convection ($0 < R_\rho < 1.0$) LMD suggest several formulations from the literature and choose the one with the most significant impact on mixing (Fedorov [11]).

$$\nu_\theta^d = (1.5^{-6})(0.909 \exp(4.6 \exp[-0.54(R_\rho^{-1} - 1)]) \tag{150}$$

for temperature. For other scalars,

$$\nu_s^d = \begin{cases} \nu_\theta^d(1.85 - 0.85R_\rho^{-1})R_\rho & \text{for } 0.5 <= R_\rho < 1.0, \\ \nu_\theta^d 0.15 R_\rho & \text{otherwise.} \end{cases} \tag{151}$$

Internal wave generated mixing Internal wave generated mixing serves as the background mixing in the LMD scheme. It is specified as a constant for both scalars and momentum. Eddy diffusivity is estimated based on the data of Ledwell et al. [41]. While Peters et al. [59] suggest eddy viscosity should be 7 to 10 times larger than diffusivity for gradient Richardson numbers below approximately 0.7. Therefore LMD use

$$\nu_m^w = 1.0 \times 10^{-4} m^2 s^{-1} \tag{152}$$

$$\nu_s^w = 1.0 \times 10^{-5} m^2 s^{-1} \tag{153}$$

4.12 Timestepping vertical viscosity and diffusion

The $\mathcal{D}_u$, $\mathcal{D}_v$, and $\mathcal{D}_C$ terms in equations (18)–(20) represent both horizontal and vertical mixing processes. The horizontal options will be covered in §4.10.1. The model has several options for computing the vertical coefficients; these will be described in §4.11. The vertical viscosity and diffusion terms have the form:

$$\frac{\partial}{\partial \sigma}\left(\frac{K}{H_z mn}\frac{\partial \phi}{\partial \sigma}\right) \tag{154}$$

where ϕ represents one of u, v, or C, and K is the corresponding vertical viscous or diffusive coefficient. This is timestepped using a semi-implicit Crank-Nicholson scheme with a weighting of 0.5 on the old timestep and 0.5 on the new timestep. Specifically, the equation of motion for ϕ can be written as:

$$\frac{\partial(H_z\phi)}{\partial t} = mnR_\phi + \frac{\partial}{\partial\sigma}\left(\frac{K}{H_z}\frac{\partial\phi}{\partial\sigma}\right) \tag{155}$$

where R_ϕ represents all of the forcing terms other than the vertical viscosity or diffusion. Since we want the diffusion term to be evaluated partly at the current timestep n and partly at the next timestep $n+1$, we introduce the parameter λ and rewrite equation (155) as:

$$\frac{\partial(H_z\phi)}{\partial t} = mnR_\phi + (1-\lambda)\frac{\partial}{\partial\sigma}\left(\frac{K}{H_z}\frac{\partial\phi^n}{\partial\sigma}\right) + \lambda\frac{\partial}{\partial\sigma}\left(\frac{K}{H_z}\frac{\partial\phi^{n+1}}{\partial\sigma}\right). \tag{156}$$

The discrete form of equation (156) is:

$$\begin{aligned}\frac{H^{n+1}_{z_k}\phi^{n+1}_k - H^n_{z_k}\phi^n_k}{\Delta t} = mnR_\phi &+ \frac{(1-\lambda)}{\Delta s^2}\left[\frac{K_k}{H_{z_k}}(\phi^n_{k+1}-\phi^n_k) - \frac{K_{k-1}}{H_{z_{k-1}}}(\phi^n_k-\phi^n_{k-1})\right] \\ &+ \frac{\lambda}{\Delta s^2}\left[\frac{K_k}{H_{z_k}}(\phi^{n+1}_{k+1}-\phi^{n+1}_k) - \frac{K_{k-1}}{H_{z_{k-1}}}(\phi^{n+1}_k-\phi^{n+1}_{k-1})\right]\end{aligned} \tag{157}$$

where k is used as the vertical level index. This can be reorganized so that all the terms involving ϕ^{n+1} are on the left and all the other terms are on the right. The equation for ϕ^{n+1}_k will contain terms involving the neighbors above and below (ϕ^{n+1}_{k+1} and ϕ^{n-1}_{k+1}) which leads to a set of coupled equations with boundary conditions for the top and bottom. The general form of these equations is:

$$A_k\phi^{n+1}_{k+1} + B_k\phi^{n+1}_k + C_k\phi^{n+1}_{k-1} = D_k \tag{158}$$

where the boundary conditions are written into the coefficients for the end points. In this case the coefficients become:

$$A(1) = 0 \tag{159}$$

$$A(2:\mathbf{N}) = -\frac{\lambda\Delta t\, K_{k-1}}{\Delta\sigma^2 H^{n+1}_{z_{k-1}}} \tag{160}$$

$$B(1) = H^{n+1}_{z_1} + \frac{\lambda\Delta t\, K_1}{\Delta\sigma^2 H^{n+1}_{z_1}} \tag{161}$$

$$B(2:\mathbf{Nm}) = H^{n+1}_{z_k} + \frac{\lambda\Delta t\, K_k}{\Delta\sigma^2 H^{n+1}_{z_k}} + \frac{\lambda\Delta t\, K_{k-1}}{\Delta\sigma^2 H^{n+1}_{z_{k-1}}} \tag{162}$$

$$B(\mathbf{N}) = H^{n+1}_{z_\mathbf{N}} + \frac{\lambda\Delta t\, K_\mathbf{Nm}}{\Delta\sigma^2 H^{n+1}_{z_\mathbf{Nm}}} \tag{163}$$

$$C(1:\mathbf{Nm}) = -\frac{\lambda\Delta t\, K_k}{\Delta\sigma^2 H^{n+1}_{z_k}} \tag{164}$$

$$C(\mathbf{N}) = 0 \tag{165}$$

$$D(1) = H^n_{z_1}\phi^n_1 + \Delta t\, mnR_{\phi_1} + \frac{\Delta t(1-\lambda)}{\Delta\sigma^2}\frac{K_1}{H^n_{z_1}}(\phi^n_2-\phi^n_1) - \frac{\Delta t}{\Delta\sigma}\tau_b \tag{166}$$

$$D(2:\mathbf{Nm}) = H^n_{z_k}\phi^n_k + \Delta t\, mnR_{\phi_k} + \tag{167}$$

$$\frac{\Delta t(1-\lambda)}{\Delta\sigma^2}\left[\frac{K_k}{H^n_{z_k}}(\phi^n_{k+1}-\phi^n_k) - \frac{K_{k-1}}{H^n_{z_{k-1}}}(\phi^n_k-\phi^n_{k-1})\right] \tag{168}$$

$$D(\mathbf{N}) = H^n_{z_\mathrm{N}}\phi^n_\mathrm{N} + \Delta t\, mnR_{\phi_\mathrm{N}} - \frac{\Delta t(1-\lambda)}{\Delta\sigma^2}\frac{K_\mathbf{Nm}}{H^n_{z_\mathbf{Nm}}}(\phi^n_\mathrm{N}-\phi^n_\mathrm{Nm}) + \frac{\Delta t}{\Delta\sigma}\tau_s \tag{169}$$

This is a standard tridiagonal system for which the solution procedure can be found in any standard reference, such as Press et al. [62].

4.13 Open boundary conditions

Need update for ROMS here...

4.13.1 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.

4.13.2 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 [56] 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 [64] have modified the scheme to account for propagation in all three directions. In ROMS, only the two horizontal directions are accounted for:

$$\frac{\partial C}{\partial t} = -\left(C_x \frac{\partial C}{\partial \xi} + C_y \frac{\partial C}{\partial \eta}) \right) \tag{170}$$

where

$$C_x = \frac{F \frac{\partial C}{\partial \xi}}{\left(\frac{\partial C}{\partial \xi}\right)^2 + \left(\frac{\partial C}{\partial \eta}\right)^2} \tag{171}$$

$$C_y = \frac{F \frac{\partial C}{\partial \eta}}{\left(\frac{\partial C}{\partial \xi}\right)^2 + \left(\frac{\partial C}{\partial \eta}\right)^2} \tag{172}$$

$$F = -\frac{\partial C}{\partial t} \tag{173}$$

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 (170), again using a consistent time stepping scheme. Raymond and Kuo give the form used for centered differencing and a leapfrog time step while ROMS 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.

5 Ice Model Formulation

The sea-ice component of ROMS is a combination of the elastic-viscous-plastic (EVP) rheology (Hunke and Dukowicz [31], Hunke [30]) and simple one-layer ice and snow thermodynamics with a molecular sublayer under the ice (Mellor and Kantha [52]). It is tightly coupled, having the same grid (Arakawa-C) and timestep as the ocean and sharing the same parallel coding structure for use with MPI or OpenMP (Budgell [5]).

5.1 Dynamics

The momentum equations describe the change in ice/snow velocity due to the combined effects of the Coriolis force, surface ocean tilt, air and water stress, and internal ice stress: (174) and (175):

$$M\frac{\partial u}{\partial t} = Mfv - Mg\frac{\partial \zeta_w}{\partial x} + \tau_a^x + \tau_w^x + \mathcal{F}_x \tag{174}$$

$$M\frac{\partial v}{\partial t} = -Mfu - Mg\frac{\partial \zeta_w}{\partial y} + \tau_a^y + \tau_w^y + \mathcal{F}_y. \tag{175}$$

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:

$$\vec{\tau}_a = \rho_a C_a |\vec{V}_{10}| \vec{V}_{10} \tag{176}$$

$$C_a = \frac{1}{2} C_d \left[1 - \cos(2\pi \min(h_i + .1, .5)\right] \tag{177}$$

$$\vec{\tau}_w = \rho_w C_w |\vec{v}_w - \vec{v}| (\vec{v}_w - \vec{v}). \tag{178}$$

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) [27]:

$$\sigma_{ij} = 2\eta\dot{\epsilon}_{ij} + (\zeta - \eta)\dot{\epsilon}_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij} \tag{179}$$

$$\dot{\epsilon}_{ij} \equiv \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{180}$$

$$P = P^* A h_i e^{-C(1-A)} \tag{181}$$

where the nonlinear viscosities are given by

$$\zeta = \frac{P}{2\left[(\epsilon_{11}^2 + \epsilon_{22}^2)(1 + 1/e^2) + 4e^{-2}\epsilon_{12}^2 + 2\epsilon_{11}\epsilon_{22}(1 - 1/e^2)\right]^{1/2}} \tag{182}$$

$$\eta = \frac{\zeta}{e^2}. \tag{183}$$

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:

$$\frac{1}{2\eta}\sigma_{ij} + \frac{\eta - \zeta}{4\eta\zeta}\sigma_{kk}\delta_{ij} + \frac{P}{4\zeta}\delta_{ij} = \dot{\epsilon}_{ij} \tag{184}$$

then add the elastic term:

$$\frac{1}{E}\frac{\partial \sigma_{ij}}{\partial t} + \frac{1}{2\eta}\sigma_{ij} + \frac{\eta - \zeta}{4\eta\zeta}\sigma_{kk}\delta_{ij} + \frac{P}{4\zeta}\delta_{ij} = \dot{\epsilon}_{ij} \tag{185}$$

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 [74]. 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 (186)–(188)) also include thermodynamic terms (S_h, S_s and S_A), which will be described in §5.2:

$$\frac{\partial Ah_i}{\partial t} = -\frac{\partial (uAh_i)}{\partial x} - \frac{\partial (vAh_i)}{\partial y} + S_h + \mathcal{D}_h \tag{186}$$

$$\frac{\partial Ah_s}{\partial t} = -\frac{\partial (uAh_s)}{\partial x} - \frac{\partial (vAh_s)}{\partial y} + S_s + \mathcal{D}_s \tag{187}$$

$$\frac{\partial A}{\partial t} = -\frac{\partial (uA)}{\partial x} - \frac{\partial (vA)}{\partial y} + S_A + \mathcal{D}_A \qquad 0 \le A \le 1. \tag{188}$$

The first two equations represent the conservation of ice and snow. Equation 188 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 [19]):

$$\frac{\partial A}{\partial t} = -\frac{\partial (uA)}{\partial x} - \frac{\partial (vA)}{\partial y} - A\alpha(A)\,\nabla \cdot \vec{v}\, H(-\nabla \cdot \vec{v}) + S_A + \mathcal{D}_A \qquad 0 \le A \le 1. \tag{189}$$

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.

Note that Hibler's h_I variable is equivalent to our Ah_i combination - his h_I is the average thickness over the whole gridbox while our h_i is the average thickness over the ice-covered fraction of the gridbox.

5.2 Thermodynamics

The thermodynamics is based on calculating how much ice grows and melts on each of the surface, bottom, and sides of the ice floes, as well as frazil ice formation (Mellor and Kantha [52]). Once the ice tracers are advected, the ice concentration and thickness are timestepped according to the terms on the right:

(186) and (188) is:

$$\frac{DAh_i}{Dt} = \frac{\rho_o}{\rho_i}\left[A(W_{io} - W_{ai}) + (1 - A)W_{ao} + W_{fr}\right] \tag{190}$$

$$\frac{DA}{Dt} = \frac{\rho_o A}{\rho_i h_i}\left[\Phi(1 - A)W_{ao} + (1 - A)W_{fr}\right] \qquad 0 \le A \le 1. \tag{191}$$

The term Ah_i is the "effective thickness", a measure of the ice volume. Its evolution equation is simply quantifying the change in the amount of ice. The ice concentration equation is more interesting in that it provides the partitioning between ice melt/growth on the sides vs. on the top and bottom. The parameter Φ controls this and has differing values for ice melt and retreat. In principle, most of the ice growth is assumed to happen at the base of the ice while rather more of the melt happens on the sides of the ice due to warming of the water in the leads.

The heat fluxes through the ice are based on a simple one layer Semtner [67] type model with snow on top. The temperature is assumed to be linear within the snow and within the ice.

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

The ice contains brine pockets for a total ice salinity of 5. The surface ocean temperature and salinity is half a dz below the surface. The water right below the surface is assumed to be at the freezing temperature; a logarithmic boundary layer is computed having the temperature and salinity matched at freezing.

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 §5.2.2—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:

$$\Phi = 4.0 \qquad W_{ao} \geq 0 \tag{192}$$

$$\Phi = 0.5 \qquad W_{ao} < 0 \tag{193}$$

$$\tag{194}$$

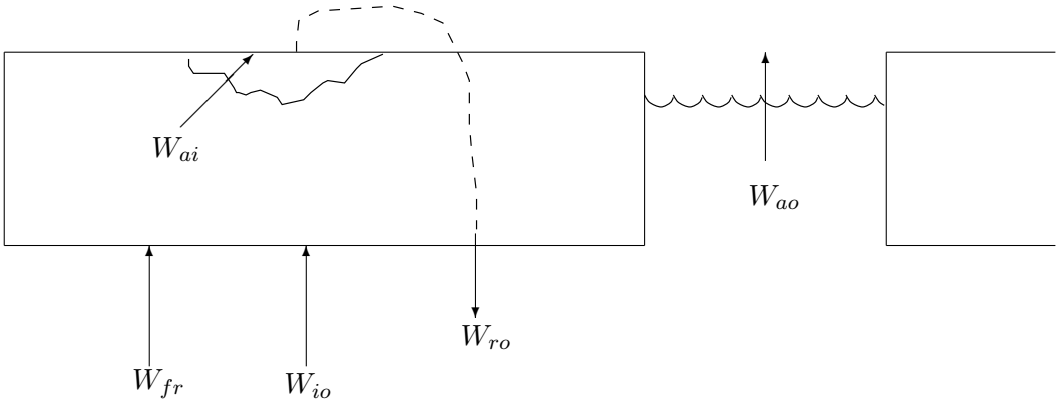

Figure 7: Diagram of the different locations where ice melting and freezing can occur.

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:

$$Q_{ai} = -H\downarrow -LE\downarrow -\epsilon_s LW\downarrow -(1-\alpha_s)SW\downarrow +\epsilon_s \sigma (T_3 + 273)^4 \tag{195}$$

The formulas for sensible heat, latent heat, and incoming longwave and shortwave radiations are the same as in Parkinson and Washington [57] and are shown in Appendix E. The sensible heat is

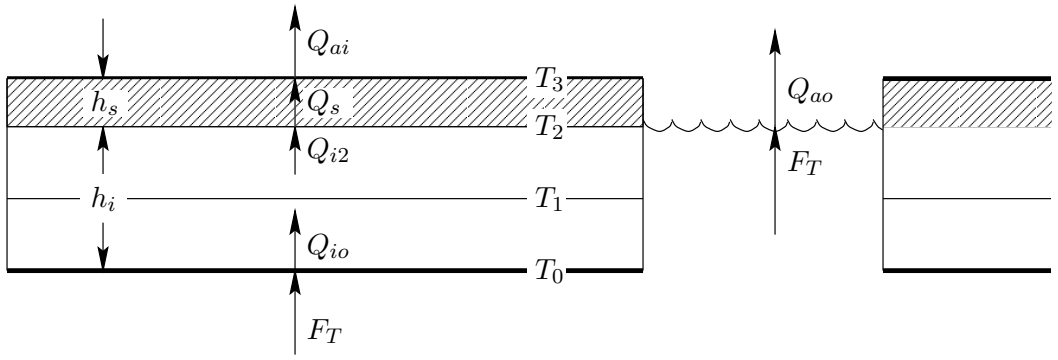

Figure 8: Diagram of internal ice temperatures and fluxes. The hashed layer is the snow.

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

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 203). As in 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:

$$W_{ai} = \frac{Q_{ai} - Q_{i2}}{\rho_o L_3} \tag{196}$$

$$L_3 \equiv [E(T_3, 1) - E(T_1, R_1)] \tag{197}$$

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

$$E(T, r) = r(L_i + C_{po}T) + (1 - r)C_{pi}T \tag{198}$$

Substituting in for r and differentiating gives:

$$\frac{\partial E}{\partial T} = -\frac{S_i m L_i}{T_1^2} + C_{pi} \tag{199}$$

Inside the snow, we have

$$Q_s = \frac{k_s}{h_s}(T_2 - T_3) \tag{200}$$

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

$$Q_{I2} = \frac{2k_i}{h_i}(T_1 - T_2) \tag{201}$$

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

$$T_2 = \frac{T_3 + C_k T_1}{1 + C_k} \tag{202}$$

where

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

Substituting into (201), we get:

$$Q_s = Q_{I2} = \frac{2k_i}{h_i}\frac{(T_1 - T_3)}{(1 + C_k)} \tag{203}$$

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

$$Q_{I0} = \frac{2k_i}{h_i}(T_0 - T_1) \tag{204}$$

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

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

$$\rho_i h_i \left[\frac{\partial E}{\partial t} + \vec{v} \cdot \nabla E\right] = Q_{I0} - Q_{I2} \tag{205}$$

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

$$\rho_i h_i \frac{\partial E}{\partial T} \left[\frac{\partial T_1}{\partial t} + \vec{v} \cdot \nabla T_1\right] = Q_{I0} - Q_{I2} \tag{206}$$

where

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

5.2.1 Ocean surface boundary conditions

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

$$K_m \frac{\partial u_w}{\partial z} = \frac{A}{\rho_o}\tau_{io}^x + \frac{1 - A}{\rho_o}\tau_{ao}^x \tag{207}$$

$$K_m \frac{\partial v_w}{\partial z} = \frac{A}{\rho_o}\tau_{io}^y + \frac{1 - A}{\rho_o}\tau_{ao}^y \tag{208}$$

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:

$$\frac{F_T}{\rho_o C_p o} = -C_{T_z}(T_0 - T) \qquad z \to 0 \tag{209}$$

where

$$C_{T_z} = \frac{u_\tau}{P_{rt} k^{-1} \ln(-z/z_0) + B_T} \tag{210}$$

$$B_T = b\left(\frac{z_0 u_\tau}{\nu}\right)^{1/2} Pr^{2/3} \tag{211}$$

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:

$$W_{io} = \frac{1}{\rho_o L_o}(Q_{io} - F_T) \tag{212}$$

$$W_{ao} = \frac{1}{\rho_o L_o}(Q_{ao} - F_T) \tag{213}$$

$$\tag{214}$$

where

$$L_o \equiv [E(T_0, 1) - E(T_1, r_1)] \tag{215}$$

The ocean model receives the following heat and salt fluxes:

$$F_T = AQ_{io} + (1 - A)Q_{ao} - W_o L_o \tag{216}$$

$$F_S = (W_o - AW_{ro})(S_i - S_0) + (1 - A)S_o(\dot{\mathrm{P}} - \dot{\mathrm{E}}) \tag{217}$$

$$W_o \equiv AW_{io} + (1 - A)W_{ao} \tag{218}$$

5.2.2 Frazil ice formation

Following Steele et al. [77], 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:

$$m_{w_1} = m_{w_2} + m_i \tag{219}$$

$$m_{w_1}(C_{pw}T_1 + L) = m_{w_2}(C_{pw}T_2 + L) + m_i C_{pi} T_2 \tag{220}$$

$$m_{w_1} S_1 = m_{w_2} S_2. \tag{221}$$

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

$$T_2 = T_1 + \gamma \left[\frac{L}{C_{pw}} + T_1 \left(1 - \frac{C_{pi}}{C_{pw}} \right) \right] \tag{222}$$

$$S_2 = S_1(1 + \gamma). \tag{223}$$

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

$$T_f = mS + nz. \tag{224}$$

We can then solve for γ:

$$\gamma = \frac{-T_1 + mS_1 + nz}{LC_{pw} + T_1\left(1 - \frac{C_{pi}}{C_{pw}}\right) - mS_1}. \tag{225}$$

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 (222) and (223) and the ice above is thickened by the amount:

$$\Delta h = \gamma_k \Delta z_k \frac{\rho_w}{\rho_i}. \tag{226}$$

5.2.3 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 added various limiters:
  - Ice concentration:$A_{\min} \leq A \leq 1.0$, $A_{\min} = 0.0$.
  - Ice thickness: $h_i \geq 0.0$.
  - Snow thickness: $h_s \geq 0.0$.
  - Brine fraction: $r \leq r_{\max}$, $r_{\max} = 0.2$
  - Surface water: $0.0 \leq W_s \leq W_{s\,\max}$, $W_{s\,\max} = 0.1$

6 Details of the Code

6.1 Directory structure

The directory structure is as shown in Fig. 9, with the ability to run the ocean alone or coupled to atmospheric and/or wave models. If running just the ocean, the model can be run forward in time (the nonlinear model) or as an adjoint, tangent linear, or representer model for data assimilation purposes. This document describes the uncoupled forward model only, specifically the version used for the Northeast Pacific domain containing sea ice and other changes from the main trunk code. Details are subject to change without notice - check your own source code for specific details as they apply to you.

The directories shown here are:

Apps This directory contains a subdirectory for each of my personal applications. The subdirectory contains files used by that application: the ROMS header file for setting cpp definitions, the analytic formulations for fields computed in the model rather than read from files (bottom heat flux of zero, for instance), and ASCII input files read by ROMS on startup to set things such as forcing file names and model time-step. Some of these applications are:

- **Bering** This is a 4 km grid of the Bering Sea, aligned with the Northeast Pacific grid but at three times the resolution.
- **Bering_10k** This is a 10 km grid of the Bering Sea, a subset of the Northeast Pacific domain, with the same extent as the 4 km grid above.
- **CGOA** This is a 3 km grid of the Gulf of Alaska. It is a subset of the Northeast Pacific grid, but at four times the resolution.
- **Circle** This is a circular domain wave propagation problem with an analytic solution used as a test problem ([37]).
- **NEP** This is the Northeast Pacific domain covering the waters off the west coast of the US, from California to the Bering Sea. It is a rectangular domain at about 11 km resolution when viewed in a conformal conic projection with standard latitudes of 40 and 60 N.

Paul Budgell's applications are also here. The application-specific files included in the main trunk ROMS are elsewhere.

Atmosphere This directory is under development, not currently supported.

Compilers This contains makefile fragments as described in §G.3.

Data Directories under here contain example forcing, grid, and initial condition NetCDF files. There is also a directory containing the headers of these files in the format produced by **ncdump** (CDL).

Lib The ARPACK and MCT libraries are needed by the data assimilation codes and by the coupled models, respectively.

makefile This is the standard ROMS makefile as described in §G.

Master The ROMS main program is here, in various forms for the forward model, coupled models and others. See §6.2.

ROMS These files are for the ocean model, as opposed to other components of the coupled system.

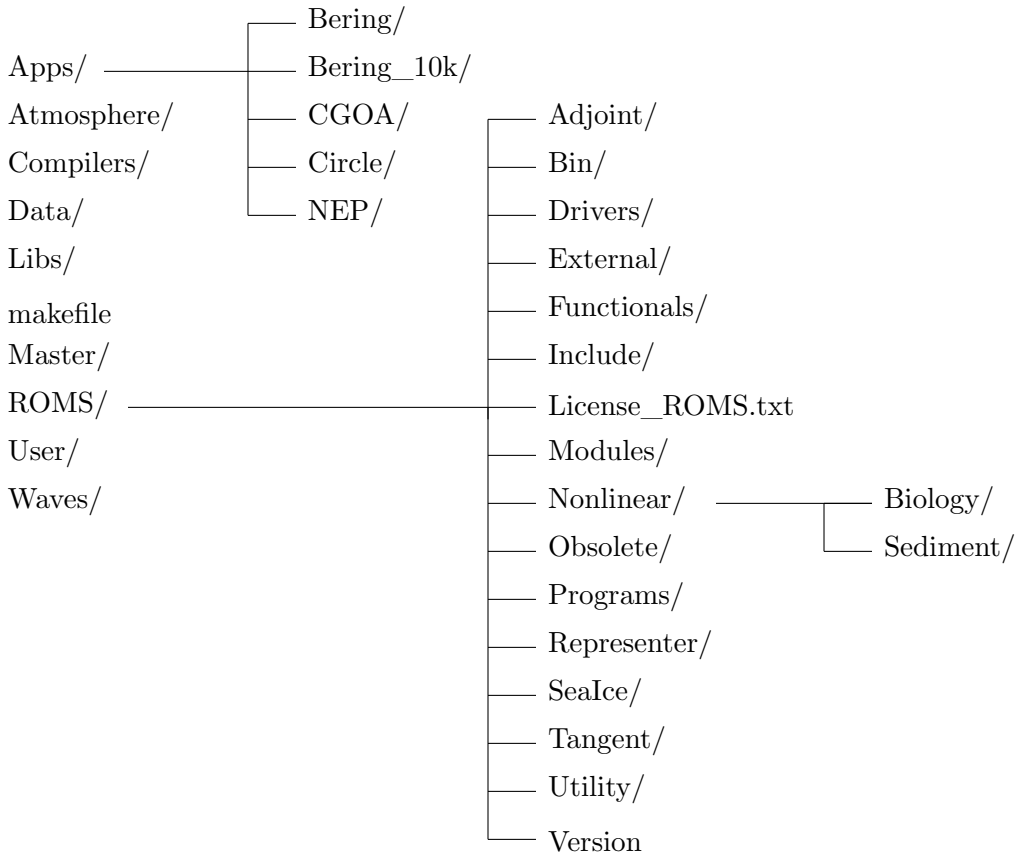

Figure 9: ROMS directory structure.

Adjoint This is the adjoint of the forward model, for data assimilation.

Bin Various shell and Perl scripts for use with the model. Note that the **.sh** files are actually **csh** scripts, not **sh** scripts—if it were up to me, I'd rename them all to **.csh**.

Drivers The main program includes one of these files, depending on how you are running the model. The forward model is in **nl_ocean.h**.

External ROMS reads an ASCII file on startup. Here are examples for various applications, also examples of the optional files for extra components such as a sediment model.

Functionals The file **analytical.F** can include one or more code bits for the analytic specification of for instance the initial conditions. Here are examples for the supported model test problems.

Include Each application has a header file with C preprocessor options for that application. For instance, the **UPWELLING** case has the include file **upwelling.h** containing C preprocessor options for its periodic channel domain. The full list of available options is in **cppdefs.h**.

License_ROMS.txt The open source license under which ROMS is copyrighted.

Modules The ROMS data structures are now in Fortran 90 module files, located here.

Nonlinear The routines used by the nonlinear forward model are here, implementing the physics described in §4.

Biology The files for the ecosystem parts of the forward model are here.

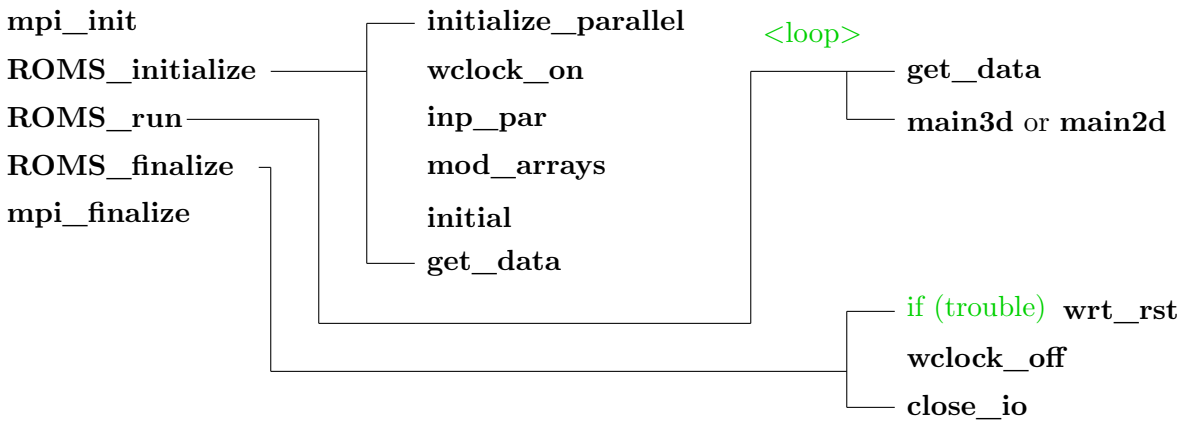

Figure 10: ROMS main structure.

Sediment The files for the sediment parts of the forward model are here.

Obsolete Long unused versions of the boundary conditions are stored here.

Programs Not all computer architectures or compilers are the same. The **types.F** program checks your compiler for the sizes of the Fortran floating point types.

Representer This is the representer of the forward model, for data assimilation.

SeaIce The sea ice model described in §5 is here.

Tangent This is the tangent linear of the forward model, for data assimilation.

Utility Here are utility functions used by the various ROMS routines, many dealing with I/O.

Version A file containing the time and date of this **svn** revision, also the **svn** URL.

User Some might choose to use this directory rather than the Apps directory. It serves the same purpose but is arranged by file type rather than by application.

Waves The SWAN wave model is here.

6.2 Main subroutines

6.2.1 master.F

The main program is in **master.F**. It is simply a shell, including one of **mct_coupler.h**, **esmf_coupler.h** or **ocean.h**. In our case, **ocean.h** contains the actual main program, which initializes MPI (if needed), calls **ROMS_initialize**, calls **ROMS_run** with arguments for how many steps to take, then **ROMS_finalize**, and finally wraps up the MPI. See Fig. 10.

6.2.2 ocean_control.F

This is again a shell which includes one of many other files to do the actual work. In this case, the worker files all contain **ROMS_initialize**, **ROMS_run** and **ROMS_finalize** and live in the **ROMS/Drivers** directory. The driver file we will be looking at is **nl_ocean.h**.

6.2.3 ROMS_initialize

This is called at the beginning of the run and therefore starts off by finding out how many parallel processes are running and which one this is, then calls the following ROMS routines:

initialize_parallel is in the **mod_parallel** module and sets up a few variables, including some for the built-in profiling.

wclock_on is in **timers.F** and initializes a timer for the built-in profiling.

inp_par reads in the ASCII input file(s) used by ROMS.

mod_arrays allocates and initializes the dynamically sized arrays in ROMS based on the grid sizes read in by **inp_par**.

initial reads in the initial conditions from a NetCDF file or computes them analytically. Likewise for the grid, plus it sets up the vertical grid spacing to be used and many other details.

get_data reads in the first record of time-varying forcing fields, boundary conditions, etc.

6.2.4 ROMS_run

This loops over all the steps from the starting iteration to the ending iteration in its argument list. The loop consists of calls to:

get_data reads in the second and subsequent records of time-varying forcing fields, boundary conditions, etc.

main3d or **main2d** solves the full equations described in §4 (**main3d**) or the depth-integrated version only (**main2d**).

6.2.5 ROMS_finalize

This is called at the end of the run, whether it was otherwise successful or not. The routines called are:

wrt_rst if the run had an error code set, it will write out a restart record of the current model fields in case they are useful in diagnosing the trouble.

wclock_off ends the built-in timers and causes them to print out a report.

close_io closes all open files so as to flush the buffers and put NetCDF files into a finished state.

6.2.6 main3d

This solves the full three-dimensional equations described in §4. It has siblings **main2d** for solving the depth-integrated equations and **main3d_offline** for reading files from a prior simulation and using them to advect the biological tracers or the Lagrangian floats. The full version is shown in Fig. 11. Note that many subroutines are optional and only get called if the appropriate C preprocessor switches have been set. The subroutines are described as follows:

set_data time interpolates between the records read in by **get_data**.

ini_zeta checks for wet/dry cells if needed and initializes all the time levels of zeta.

ini_fields initializes the 2-D velocities to match the vertical integral of the 3-D velocities, making all the time levels match.

set_massflux computes horizontal mass fluxes, $\frac{H_z u}{n}$ and $\frac{H_z v}{m}$.

rho_eos computes the nonlinear equation of state.

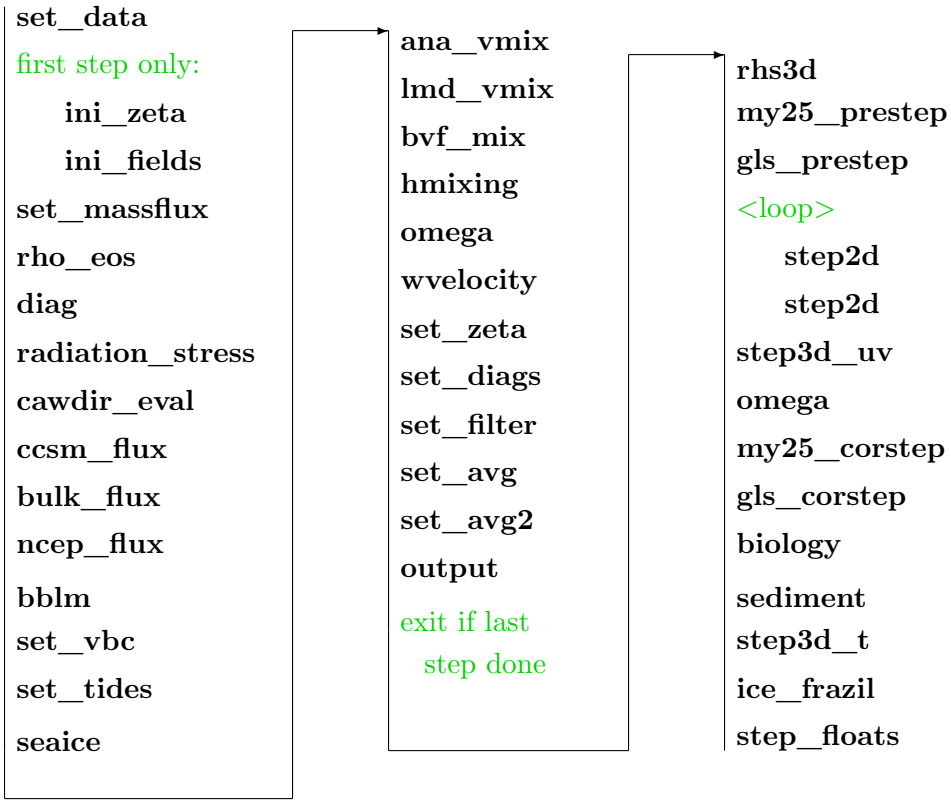

Figure 11: Flow chart of the model main program.

diag computes some global sums, prints them, and checks them to see if they are sensible. If not, it stops the model run.

radiation_stress computes the radiation stresses due to wave-current interactions ([50] and [51]).

cawdir_eval computes a 24-hour mean albedo at the marine surface. Not in the trunk code.

ccsm_flux computes the surface fluxes from the atmosphere based on a marine boundary layer. This version comes from CCSM and is reputed to do better outside of the tropics. Not in the trunk code.

bulk_flux computes the surface fluxes from the atmosphere based on a marine boundary layer. This version comes from COARE version 3.0 ([10], [79] and [55]).

ncep_flux computes the surface fluxes from the NCEP atmospheric model. Not in the trunk code.

bblm compute the bottom stresses from one of three bottom boundary layer models.

set_vbc computes the surface and bottom fluxes and stresses that aren't computed elsewhere—set vertical boundary conditions.

set_tides computes the tidal boundary conditions from the tidal constituents.

seaice runs the sea ice model described in §5. It changes the surface boundary conditions for the ocean and therefore gets called before the call to **output** or anything else that would be needing the surface boundary conditions. Not in the trunk code.

ana_vmix is called if there's an analytic profile for the vertical mixing coefficient.

lmd_vmix is called when using the K-profile parameterization of vertical mixing ([39] and [38]).

bvf_mix computes the vertical mixing as a function of the Brunt-Väisälä frequency.

hmixing computes time-dependent horizontal mixing coefficients ([73], [29], [85] and [21]).

omega computes the Ω vertical velocity from the horizontal divergences.

wvelocity computes the physical vertical velocity for the model output.

set_zeta sets the surface elevation to the time-mean over the last baroclinic time-step.

set_diags accumulates the time-average of the diagnostics fields.

set_filter accumulates a weighted sum using a Lanczos filter for detiding the most important of the output fields. Not in the trunk code.

set_avg accumulates time-averaged fields for the averages output.

set_avg2 accumulates the time-averaged surface fields for the second averages output. Not in the trunk code.

output writes to various output NetCDF files.

rhs3d computes right-hand-sides of the three-dimensional velocity and tracer fields.

my25_prestep computes the predictor step for turbulent kinetic energy prognostic variables, **tke** and **gls**.

gls_prestep computes the predictor step for turbulent kinetic energy prognostic variables, **tke** and **gls**.

step2d computes the depth-integrated time-step. It is called in a loop over all the short time-steps, first as a predictor step, then as a corrector step.

step3d_uv completes the time-step for the three-dimensional velocities.

omega computes the Ω vertical velocity.

my25_corstep performs the corrector step for turbulent kinetic energy and length scale prognostic variables, **tke** and **gls** ([54] and [18]).

my25_corstep performs the corrector step for turbulent kinetic energy and length scale prognostic variables, **tke** and **gls** ([82]).

biology computes the changes to the biological tracers due to biological activity using one of several options for the ecosystem model.

sediment computes changes to the sediment tracers ([84]).

step3d_t completes the tracer time-step.

ice_frazil computes the frazil ice growth, if any. Not in the trunk code.

step_floats time-steps the Lagrangian floats.

6.3 Initialization

checkdefs Reports on which C preprocessor variables have been **#define**d and checks their consistency.

ana_grid Computes the grid internally.

ana_mask Computes the land mask internally.

get_grid Reads in the curvilinear coordinate arrays as well as f and h from a grid NetCDF file.

set_scoord Sets and initializes relevant variables associated with the vertical transformation to nondimensional σ-coordinate described in Appendix B.

set_weights Sets the barotropic time-step average weighting function.

metrics Computes the metric term combinations which do not depend on the surface elevation and therefore remain constant in time.

ana_hmixcoef Computes the horizontal mixing coefficients.

ana_nudgcoef Computes the nudging time scales.

ana_initial Analytic initial conditions for momentum and active tracers.

ana_passive Analytic initial conditions for passive tracers.

ana_biology Analytic initial conditions for ecosystem tracers.

ana_sediment Analytic initial conditions for sediment tracers.

ana_ice Analytic initial conditions for ice variables.

get_state Reads initial fields from disk—either restart or from some other source which has been converted to the appropriate format of NetCDF file.

set_depth Compute time-evolving depths.

set_massflux Compute initial horizontal mass fluxes.

get_idata Read in time-invariant forcing data.

stiffness Compute grid stiffness.

grid_coords Convert initial float and station locations to fractional grid coordinates.

6.4 Modules

Now that we are using Fortran 90, the method of choice for managing data structures is modules. The **ROMS/Modules** directory contains all of the ROMS modules that contain globally used variables. The complete list is:

mod_arrays.F This actually has no data structures, but has the routine that calls the allocate and initialize routines for all the others.

mod_average.F If **AVERAGES** is defined, this will provide the storage for the running means of the fields you are averaging.

mod_average2.F If **AVERAGES2** is defined, this will provide the storage for the surface running means of the fields you are averaging.

mod_bbl.F If **BBL_MODEL** is defined, this will provide the storage for the bottom boundary fields.

mod_biology.F If **BIOLOGY** is defined, this will provide the storage for the biology interaction parameters.

mod_boundary.F This contains the storage for the open boundary conditions. If they aren't provided analytically, this will also provide the storage for fields read from a file that need to be time-interpolated.

mod_clima.F If one of **CLIMATOLOGY** or several other options is defined, this will provide the storage for the climatology fields.

mod_coupler.F If either **MODEL_COUPLING** or **ESMF_LIB** is defined, this will set up the requisite fields and data structures for the coupling.

mod_coupling.F If **SOLVE3D** is defined, this will provide the storage for the fields used in coupling the 2-D and 3-D components of the simulation.

mod_diags.F If **DIAGNOSTICS** is defined, this will provide the storage for the various tendency terms.

mod_eclight.F If both **BIOLOGY** and **ECOSIM** are defined, this will set up the spectral irradiance variables.

mod_eoscoef.F If **NONLIN_EOS** is defined, this will provide the polynomial expansion coefficients for the nonlinear equation of state for sea water.

mod_filter.F If **FILTERED** is defined, this will provide the storage for the weighted means used in detiding the averages.

mod_floats.F If **FLOATS** is defined, this will provide the storage for the float tracking variables.

mod_forces.F This provides the storage for the surface and bottom forcing fields.

mod_fourdvar.F If either **FOUR_DVAR** or **VERIFICATION** is defined, this will set up the variational data assimilation variables.

mod_grid.F This provides the storage for the model grid fields.

mod_ice.F If **ICE_MODEL** is defined, this will provide storage for the ice fields.

mod_iounits.F This contains a number of variables used by the I/O, including file names and file IDs.

mod_kinds.F This contains the integers associated with the various integer and real Fortran types. If you find more systems supporting 128-bit reals, let us know.

mod_mixing.F This contains the arrays for the various optional horizontal and vertical mixing parameterizations.

mod_ncparam.F This contains all sorts of parameters relating to the NetCDF I/O files, including that read from the **varinfo.dat** file. The parameters **MV** and **NV** are set here, giving the maximum number of variables that can be read [this is a change from the trunk code].

mod_nesting.F If **NESTING** is defined, this module defines generic structures used for nesting, composed, and mosaic grids. Not yet functional.

mod_netcdf.F This brings in **netcdf.mod** and defines a few type variables based upon it.

mod_obs.F If either **ASSIMILATION** or **NUDGING** is defined, this contains variables for the observed fields.

mod_ocean.F This contains the 2-D and 3-D fields of the primitive ocean variables and optionally the sediment variables.

mod_parallel.F This sets up some global variables such as **Master**, which is true for the master thread or process. It also initializes the internal ROMS profiling arrays.

mod_param.F This contains the sizes of each grid used, plus things like how many tidal constituents are being used. Many of these are read from the input files during initialization, not known at compile time.

mod_scalars.F This contains a large number of scalars, i.e. values which don't have spatial dependence. Some are fixed constants such as **itemp** referring to the temperature tracer. Others could have a different value on each grid.

mod_sediment.F If either **SEDIMENT** or **BBL_MODEL** is defined, this contains parameters for the respective model.

mod_sources.F If one of **UV_PSOURCE**, **TS_PSOURCE** or **Q_PSOURCE** is defined, this contains the variables used for point sources.

mod_stepping.F This contains the time-stepping variables used to point to the relevant time level.

mod_storage.F If **PROPAGATOR** is defined, this module defines the work space for the Generalized Stability Theory (GST) Analysis package (ARPACK).

mod_strings.F This contains strings such as a title for the run, the list of **cpp** options defined, and the names of the sections of code being profiled.

mod_tides.F If **SSH_TIDES** and/or **UV_TIDES** is defined, this will provide the storage for the tidal constituents.

6.5 Functionals

The Functionals directory contains **analytical.F** which conditionally includes code bits for computing analytic values for a wide variety of fields. Many are alternates for reading from NetCDF files, especially for idealized problems.

ana_aiobc Computes open boundary conditions for the ice concentration.

ana_biology Computes analytic initial conditions for the biology tracers.

ana_bmflux Computes analytic kinematic bottom momentum flux.

ana_btflux Computes analytic kinematic bottom flux of tracer type variables.

ana_cloud Computes analytic cloud fraction.

ana_diag Computes customized diagnostics.

ana_fsobc Computes analytic open boundary conditions for the free surface.

ana_grid Sets up an analytic grid.

ana_hiobc Computes open boundary conditions for the ice thickness.

ana_hmixcoef Computes spatially variable horizontal mixing coefficients.

ana_hsnobc Computes open boundary conditions for the snow thickness.

ana_humid Computes analytic atmospheric humidity.

ana_ice Computes analytic initial conditions for the sea ice.

ana_initial Sets up analytic initial conditions for the ocean.

ana_m2clima Sets up an analytic climatology for the two-dimensional momentum.

ana_m2obc Computes open boundary conditions for the two-dimensional momentum.

ana_m3clima Sets up an analytic climatology for the three-dimensional momentum.

ana_m3obc Computes open boundary conditions for the three-dimensional momentum.

ana_mask Sets up an analytic mask.

ana_ncep Sets up analytic fields as if they came from NCEP.

ana_nudgcoef Sets up spatially dependent nudging coefficients for nudging to a climatology.

ana_pair Computes analytic sea-level air pressure.

ana_passive Computes analytic initial conditions for passive tracers.

ana_perturb Computes analytic perturbations to the initial conditions.

ana_psource Computes analytic point source fluxes.

ana_rain Computes analytic rainfall.

ana_scope Sets adjoint sensitivity spatial scope masking arrays.

ana_sediment Computes analytic initial conditions for the sediment tracers.

ana_smflux Computes analytic kinematic surface momentum flux (wind stress).

ana_specir Sets surface solar downwelling spectral irradiance at just beneath the sea surface.

ana_spinning Sets time-variable rotation force as the sum of Coriolis and Centripetal accelerations. This is used in polar coordinate applications (annulus grid).

ana_srflux Computes analytic kinematic surface shortwave radiation.

ana_ssh Computes analytic sea surface height.

ana_sss Computes analytic sea surface salinity.

ana_sst Computes analytic sea surface temperature and dQdSST which are used in the surface heat flux correction.

ana_stflux Computes analytic kinematic surface flux of tracer type variables.

ana_tair Computes analytic air temperature.

ana_tclima Computes analytic tracer climatology fields.

ana_tobc Computes analytic open boundary conditions for all tracers (active, passive, biology, and sediment).

ana_vmix Computes analytic vertical mixing coefficients.

ana_winds Computes analytic winds.

ana_wwave Computes analytic wind-induced wave amplitude, direction and period.

6.6 Other subroutines and functions

NetCDF I/O

def_* Creates the ROMS NetCDF file of the appropriate type, including dimensions, attributes, and variables.

def_info Adds some standard scalar variables to any NetCDF file.

get_srflux Reads shortwave radiation flux

wrt_* Writes to the ROMS NetCDF file of the appropriate type.

6.7 C preprocessor variables

Before it can be compiled, the model must be run through the C preprocessor **cpp**, as described in Appendix F. The C preprocessor has its own variables, which may be defined either with an explicit **#define** command or with a command line option to **cpp**. We have chosen to define these variables in an application-specific include file, except for some machine-dependent ones, which are defined in the **makefile**. These variables allow you to conditionally compile sections of the code. For instance, if **MASKING** is not defined, the masking code will not be seen by the compiler, and the masking variables will not be declared. These **cpp** variables can be grouped into several categories:

Momentum terms The default horizontal advection is 3rd-order upstream bias for 3D momentum and 4th-order centered for 2D momentum. The default vertical advection is 4th-order centered for 3D momentum. If this is the case, no flags for momentum advection need to be activated except for **UV_ADV**.

The 3rd-order upstream split advection (**UV_U3ADV_SPLIT**) can be used to correct for the spurious mixing of the advection operator in terrain-following coordinates. If this is the case, the advection operator is split in advective and viscosity components and several internal flags are activated in **globaldefs.h**. Notice that horizontal and vertical advection of momentum is 4th-order centered plus biharmonic viscosity to correct for spurious mixing.

UV_ADV Define to compute the momentum advection terms.

CURVGRID Define to compute the extra non-linear advection terms which arise when using curvilinear coordinates.

UV_COR Define to compute the Coriolis term.

UV_U3ADV_SPLIT Define for 3rd-order upstream split momentum advection.

UV_C2ADVECTION Define for 2nd-order centered advection.

UV_C4ADVECTION Define for 4rd-order centered advection.

UV_SADVECTION Define for splines vertical advection (for shallow, vertically well-resolved domains).

UV_VIS2 Define to compute the horizontal Laplacian viscosity.

UV_VIS4 Define to compute the horizontal biharmonic viscosity.

UV_SMAGORINSKY Define for Smagorinsky-like viscosity.

UV_LOGDRAG Define for logarithmic bottom friction.

UV_LDRAG Define for linear bottom friction.

UV_QDRAG Define for quadratic bottom friction.

RDRG_GRID Define for spatially variable bottom drag.

DRAG_LIMITER Define for bottom drag limiter.

UV_PSOURCE Define for point sources/sinks.

Q_PSOURCE Define for mass point sources/sinks.

Tracers The default horizontal and vertical advection is 4th-order centered.

The 3rd-order upstream split advection (**TS_U3ADV_SPLIT**) can be used to correct for the spurious diapycnal diffusion of the advection operator in terrain-following coordinates. If this is the case, the advection operator is split in advective and diffusive components and several internal flags are activated in **globaldefs.h**. Notice that horizontal and vertical advection of tracer is 4th-order centered plus biharmonic diffusion to correct for spurious diapycnal mixing. The total time-dependent horizontal mixing coefficient are computed in **hmixing.F**. It is also recommended to use the rotated mixing tensor along geopotentials (**MIX_GEO_TS**) for the biharmonic operator.

TS_U3ADV_SPLIT Define for 3rd-order upstream split tracer advection.

TS_A4HADVECTION Define for 4nd-order Akima horizontal advection.

TS_C2HADVECTION Define for 2nd-order centered horizontal advection.

TS_C4HADVECTION Define for 4rd-order centered horizontal advection.

TS_MPDATA Define for recursive MPDATA 3D advection ([47]).

TS_U3HADVECTION Define for 3nd-order upstream horizontal advection.

TS_A4VADVECTION Define for 4nd-order Akima vertical advection.

TS_C2VADVECTION Define for 2nd-order centered vertical advection.

TS_C4VADVECTION Define for 4rd-order centered vertical advection.

TS_SADVECTION Define for splines vertical advection (for shallow, vertically well-resolved domains).

TS_DIF2 Define to compute horizontal Laplacian diffusion.

TS_DIF4 Define to compute horizontal biharmonic diffusion.

TS_SMAGORINSKY Define for Smagorinsky-like diffusion.

TS_FIXED Define for a diagnostic calculation in which the tracer fields do not change in time.

T_PASSIVE Define for passive tracers.

SALINITY Define if salinity is used as one of the active tracers.

NONLIN_EOS Define to use the nonlinear equation of state.

QCORRECTION Define to use the net heat flux correction.

SCORRECTION Define to use freshwater flux correction.

SOLAR_SOURCE Define to use solar radiation source term.

SRELAXATION Define to use salinity relaxation as a freshwater flux.

TS_PSOURCE Define for point sources/sinks.

Pressure gradient options If no option is selected, the pressure gradient term is computed using standard density Jacobian algorithm. Notice that there are two quartic pressure Jacobian options. They differ on how the WENO reconciliation step is done and in the monotonicity constraining algorithms.

DJ_GRADPS Define for splines density Jacobian ([69]).

PJ_GRADP Define for finite volume Pressure Jacobian ([43]).

PJ_GRADPSQ2 Define for quartic 2 Pressure Jacobian ([69]).

PJ_GRADPSQ4 Define for quartic 4 Pressure Jacobian ([69]).

DJ_GRADPS Define for weighted density Jacobian ([75]).

ATM_PRESS Define to impose atmospheric sea-level pressure onto the sea surface.

Atmospheric boundary layer There are three ways to provide longwave radiation in the atmospheric boundary layer: (1) Compute the net longwave radiation internally using the Berliand (1952) equation (**LONGWAVE**) as function of air temperature, sea surface temperature, relative humidity, and cloud fraction; (2) provide (read) longwave downwelling radiation only and then add outgoing longwave radiation (**LONGWAVE_OUT**) as a function of the model sea surface temperature; (3) provide net longwave radiation (default).

The shortwave radiation can be computed using the global albedo equation with a cloud correction. Alternatively, input shortwave radiation data computed from averaged data (with snapshots greater or equal than 24 hours) can be modulated by the local diurnal cycle which is a function longitude, latitude and day-of-year.

BULK_FLUXES Define for bulk flux computation.

CCSM_FLUXES Define for CCSM version of bulk flux computation.

NCEP_FLUXES Define if NCEP forcing files are used.

NL_BULK_FLUXES Define to use bulk fluxes computed by nonlinear (forward) model.

COOL_SKIN Define for cool skin correction.

LONGWAVE Define to compute net longwave radiation.

LONGWAVE_OUT Define to compute outgoing longwave radiation.

EMINUSP Define to compute evaporation minus precipitation.

COARE_TAYLOR_YELLAND Define to use Taylor and Yelland wave roughness ([79]).

COARE_OOST Define to use Oost et al. wave roughness ([55]).

DEEPWATER_WAVES Define to use deep water waves approximation.

ALBEDO Define to use albedo equation for shortwave radiation.

DIURNAL_SRFLUX Define to impose the local diurnal cycle onto the shortwave radiation.

General model configuration

SOLVE3D Define to solve the 3-D primitive equations.

MASKING Define if there is land in the domain to be masked out.

BODYFORCE Define to apply the surface stress as a body force.

PROFILE Define for time profiling.

AVERAGES Define to write out time-averaged model fields.

AVERAGES2 Define to write out secondary time-averaged model fields.

AVERAGES_AKV Define to write out time-averaged AKv.

AVERAGES_AKT Define to write out time-averaged AKt.

AVERAGES_AKS Define to write out time-averaged AKs.

AVERAGES_DETIDE Define to write out time-averaged detided fields, one method.

FILTERED Define to write out time-averaged detided fields, using a Lanczos filter.

AVERAGES_FLUXES Define to write out time-averaged surface fluxes.

AVERAGES_NEARSHORE Define to write out time-averaged nearshore stresses.

AVERAGES_QUADRATIC Define to write out time-averaged quadratic terms.

AVERAGES_BEDLOAD Define to write out time-averaged bed load.

ICESHELF Define for ice shelf cavities.

SPHERICAL Define if lat/lon coordinates rather than x/y.

SPLINES Define for conservative, parabolic spline reconstruction of vertical derivatives.

STATIONS Define to write out time-series information at specific points in the model.

STATIONS_CGRID Define if stations are on native C-grid.

FLOATS Define for simulated Lagrangian drifters.

FLOATS_VWALK Define if floats do vertical random walk.

VWALK_FORWARD Define for forward time stepping of vertical random walk.

DEBUGGING Define to suppress timestamps for easier comparisons between files.

Analytic fields

ANA_BIOLOGY Define for analytic biology initial conditions.

ANA_BPFLUX Define for an analytic bottom passive tracer flux.

ANA_BSFLUX Define for an analytic bottom salt flux.

ANA_BTFLUX Define for an analytic bottom heat flux.

ANA_CLOUD Define for an analytic cloud fraction.

ANA_DIAG Define for customized diagnostics.

ANA_FSOBC Define for analytic free-surface boundary conditions.

ANA_GRID Define for an analytic model grid set-up.

ANA_HUMIDITY Define for analytic surface air humidity.

ANA_ICE Define for analytic ice initial conditions.

ANA_INITIAL Define for analytic initial conditions.

ANA_M2CLIMA Define for an analytic 2D momentum climatology.

ANA_M2OBC Define for analytic 2D momentum boundary conditions.

ANA_M3CLIMA Define for an analytic 3D momentum climatology.

ANA_M3OBC Define for analytic 3D momentum boundary conditions.

ANA_MASK Define for an analytic mask.

ANA_PAIR Define for an analytic surface air pressure.

ANA_PASSIVE Define for analytic initial conditions for inert tracers.

ANA_PERTURB Define for analytic perturbation of initial conditions.

ANA_PSOURCE Define for analytic point sources.

ANA_RAIN Define for analytic rain fall rate.

ANA_SEDIMENT Define for analytic sediment initial fields.

ANA_SMFLUX Define for an analytic kinematic surface momentum stress.

ANA_SPFLUX Define for analytic surface passive tracers fluxes.

ANA_SPINNING Define for an analytic time-varying rotation force.

ANA_SRFLUX Define for an analytic kinematic surface shortwave radiation.

ANA_SSFLUX Define for an analytic kinematic surface freshwater flux.

ANA_SSH Define for an analytic sea surface height.

ANA_SSS Define for an analytic sea surface salinity.

ANA_SST Define for an analytic SST and $\partial Q/\partial$SST.

ANA_STFLUX Define for an analytic kinematic surface heat flux.

ANA_TAIR Define for analytic surface air temperature.

ANA_TCLIMA Define for an analytic tracer climatology.

ANA_TOBC Define for analytic tracer open boundary conditions.

ANA_VMIX Define for analytic vertical mixing coefficients.

ANA_WIND Define for analytic surface winds.

ANA_WWAVE Define for an analytic wind induced wave field.

Horizontal mixing of momentum

MIX_GEO_UV Define for viscosity along constant z (geopotential) surfaces.

MIX_S_UV Define for viscosity along constant s surfaces.

VISC_GRID Define for horizontally variable viscosity coefficient.

Horizontal mixing of tracers

DIFF_GRID Define for horizontally variable diffusion coefficient.

MIX_GEO_TS Define for diffusion along constant z (geopotential) surfaces.

MIX_ISO_TS Define for diffusion along constant potential density (epineutral) surfaces.

MIX_S_TS Define for diffusion along constant s surfaces.

CLIMA_TS_MIX Define for diffusion of tracer perturbation $T - Tclm$

Vertical mixing

BVF_MIXING Define to activate Brunt-Väisälä frequency mixing.

GLS_MIXING Define for Generic Length-Scale mixing.

CANUTO_A Define for Canuto A-stability function formulation.

CANUTO_B Define for Canuto B-stability function formulation.

CHARNOK Define for Charnok surface roughness from wind stress.

CRAIG_BANNER Define for Craig and Banner wave breaking surface flux.

KANTHA_CLAYSON Define for Kantha and Clayson stability function.

K_C2ADVECTION Define for 2th-order centered advection.

K_C4ADVECTION Define for 4th-order centered advection.

N2S2_HORAVG Define for horizontal smoothing of buoyancy/shear.

ZOS_HSIG Define for surface roughness from wave amplitude.

TKE_WAVEDISS Define for wave breaking surface flux from wave amplitude.

LMD_MIXING Define to activate Large/McWilliams/Doney interior closure.

LMD_BKPP Define to add a bottom boundary layer from a local K-Profile Parameterization (KPP).

LMD_CONVEC Define to add convective mixing due to shear instabilities.

LMD_DDMIX Define to add double-diffusive mixing.

LMD_NONLOCAL Define to add convective nonlocal transport.

LMD_RIMIX Define to add diffusivity due to shear instabilities.

LMD_SHAPIRO Define to Shapiro filtering boundary layer depths.

LMD_SKPP Define to add a surface boundary layer from a local K-Profile Parameterization (KPP).

MY25_MIXING Define to activate Mellor/Yamada Level-2.5 closure.

KANTHA_CLAYSON Define for Kantha and Clayson stability function.

K_C2ADVECTION Define for 2th-order centered advection.

K_C4ADVECTION Define for 4th-order centered advection.

N2S2_HORAVG Define for horizontal smoothing of buoyancy/shear.

PP_MIXING Define to activate Pacanowski/Philander closure.

RI_HORAVG Define for horizontal Richardson number smoothing.

RI_VERAVG Define for vertical Richardson number smoothing.

Bottom boundary layer The Options **MB_Z0BL** and **MB_Z0RIP** should be activated concurrently.

MB_BBL Define to activate Meinte Blaas BBL closure.

MB_CALC_ZNOT Define to compute bottom roughness internally.

MB_CALC_UB Define to compute bottom orbital velocity internally.

MB_Z0BIO Define for biogenic bedform roughness for ripples.

MB_Z0BL Define for bedload roughness for ripples.

MB_Z0RIP Define for bedform roughness for ripples.

SG_BBL Define to activate Styles/Glenn bottom boundary layer formulation.

SG_CALC_ZNOT Define to compute bottom roughness internally.

SG_CALC_UB Define to compute bottom orbital velocity internally.

SG_LOGINT Define for logarithmic interpolation of (Ur,Vr).

SSW_BBL Define to activate Sherwood/Signell/Warner bottom boundary layer closure.

SSW_CALC_ZNOT Define to compute bottom roughness internally.

SSW_CALC_UB Define to compute bottom orbital velocity internally.

SSW_LOGINT Define for logarithmic interpolation of (Ur,Vr).

SSW_FORM_DRAG_COR Define to activate form drag coefficient.

SSW_Z0BIO Define for biogenic bedform roughness for ripples.

SSW_Z0BL Define for bedload roughness for ripples.

SSW_Z0RIP Define for bedform roughness for ripples.

Sea ice

ICE_MODEL Define to use ice component of the model (see §5).

ICE_THERMO Define for ice thermodynamics.

ICE_MK Define for Mellor-Kantha ([52]) ice thermodynamics—this is the only choice.

ICE_SMOOTH Define to smooth some ice fields.

ICE_ALB_EC92 Define for albedo computations from Ebert and Curry ([9]).

ICE_MOMENTUM Define for momentum component of the ice.

ICE_MOM_BULK Define for alternate ice-water stress computation.

ICE_EVP Define for elastic-viscous-plastic rheology ([31] and [30]).

ICE_ADVECT Define for advection of ice tracers.

ICE_SMOLAR Define to use MPDATA for ice tracers (no other option).

ICE_UPWIND Define for upwind advection (not available).

ICE_BULK_FLUXES Define for ice part of bulk flux computation.

Boundary conditions

EW_PERIODIC Define for periodic boundaries in the ξ-direction.

SPONGE Define to use SSH climatology as 2D inflow data.

NS_PERIODIC Define for periodic boundaries in the η-direction.

RADIATION_2D Define for tangential phase speed in radiation conditions.

Detailed eastern open boundary conditions Other sides have similar. If none of these are defined, it is assumed to be a closed wall.

EAST_FSCHAPMAN Define for a Chapman condition on the free surface.

EAST_FSGRADIENT Define for a gradient condition on the free surface.

EAST_FSRADIATION Define for a radiation condition on the free surface.

EAST_FSNUDGING Define for an active/passive nudging term on the free surface.

EAST_FSCLAMPED Define for a clamped free surface.

EAST_M2FLATHER Define for a Flather condition on the 2-D momentum.

EAST_M2GRADIENT Define for a gradient condition on the 2-D momentum.

EAST_M2RADIATION Define for a radiation condition on the 2-D momentum.

EAST_M2REDUCED Define for a reduced physics condition on the 2-D momentum.

EAST_M2NUDGING Define for an active passive nudging term on the 2-D momentum.

EAST_M2CLAMPED Define for clamped 2-D momentum.

EAST_M3GRADIENT Define for a gradient condition on the 3-D momentum.

EAST_M3RADIATION Define for a radiation condition on the 3-D momentum.

EAST_M3NUDGING Define for an active passive nudging term on the 3-D momentum.

EAST_M3CLAMPED Define for clamped 3-D momentum.

EAST_KGRADIENT Define for a gradient condition on the TKE fields.

EAST_KRADIATION Define for a radiation condition on the TKE fields.

EAST_TGRADIENT Define for a gradient condition on the tracers.

EAST_TRADIATION Define for a radiation condition on the tracers.

EAST_TNUDGING Define for an active passive nudging term on the tracers.

EAST_TCLAMPED Define for clamped tracers.

EAST_VOLCONS Define for Eastern edge mass conservation enforcement.

Tides The tidal data is processed in terms of tidal constituents, classified by period. The tidal forcing is computed for the full horizontal grid. If requested, the tidal forcing is added to the processed open boundary data.

Both tidal elevation and tidal currents are required to force the model properly. However, if only the tidal elevation is available, the tidal currents at the open boundary can be estimated by reduced physics. Only the pressure gradient, Coriolis, and surface and bottom stresses terms are considered at the open boundary. See **u2dbc_im.F** or **v2dbc_im.F** for details. Notice that there is an additional option (**FSOBC_REDUCED**) for the computation of the pressure gradient term in both Flather or reduced physics conditions (***_M2FLATHER**, ***_M2REDUCED**).

SSH_TIDES Define if imposing tidal elevation.

UV_TIDES Define if imposing tidal currents.

RAMP_TIDES Define if ramping (over one day) tidal forcing from zero.

ADD_FSOBC Define to tidal elevation to processed OBC data.

ADD_M2OBC Define to tidal currents to processed OBC data.

TIDES_ASTRO Define to add contributions from the long-period tides as done by Foreman.

Climatology

M2CLIMATOLOGY Define for processing the 2-D momentum climatology arrays.

M3CLIMATOLOGY Define for processing the 3-D momentum climatology arrays.

OCLIMATOLOGY Define for processing the vertical momentum climatology arrays.

TCLIMATOLOGY Define for processing the tracer climatology arrays.

ZCLIMATOLOGY Define for processing the sea surface height climatology arrays.

M2CLM_NUDGING Define for nudging to 2-D momentum climatology.

M3CLM_NUDGING Define for nudging to 3-D momentum climatology.

TCLM_NUDGING Define for nudging to tracer climatology.

ZCLM_NUDGING Define for nudging to sea surface height climatology.

Ecosystem models

BIO_FENNEL Define for Fennel ([12]) nitrogen-based model.

BIO_SEDIMENT Define to restore fallen material to the nutrient pool.

CARBON Define to add carbon constituents.

DENITRIFICATION Define to add denitrification processes.

OXYGEN Define to add oxygen dynamics.

OCMIP_OXYGEN_SC Define if Schmidt number from Keeling et al. ([34]).

RIVER_BIOLOGY Define for river biology point-sources.

TALK_NONCONSERV Define for nonconservative computation of alkalinity.

BEST_NPZ Define for Gibson et al. (personal communication) Bering Sea model.

STATIONARY Define for extra output.

BENTHIC Define for benthic components.

ICE_BIO Define for ice algae.

JELLY Define for jellyfish.

CLIM_ICE_1D Define if 1-D with ice.

BIO_UMAINE Define for Chai et al. ([7]) model.

BIO_GOANPZ Define for Hinckley et al. ([28]) Gulf of Alaska model.

NPZD_FRANKS Define for NPZD model of Franks et al. ([16]).

NPZD_IRON Define for NPZD model with iron limitation.

NPZD_POWELL Define for NPZD model of Powell et al. ([61]).

IRON_LIMIT Define for iron limitation on phytoplankton growth.

IRON_RELAX Define for nudging to iron over the shelf.

ECOSIM Define for bio-optical EcoSim model.

NEMURO Define for Nemuro ecosystem model ([36]). Need to choose a zooplankton grazing option (**HOLLING_GRAZING** or **IVLEV_EXPLICIT**). The default implicit **IVLEV** algorithm does not work yet.

BIO_SEDIMENT Define to restore fallen material to the nutrient pool.

HOLLING_GRAZING Define for Holling-type s-shaped curve grazing (implicit).

IVLEV_EXPLICIT Define for Ivlev explicit grazing algorithm.

Sediment transport model

SEDIMENT Define to activate sediment transport model ([84]).

BEDLOAD_MPM Define to activate Meyer-Peter-Mueller bed load.

BEDLOAD_SOULSBY Define to activate Soulsby wave/current bed load.

RIVER_SEDIMENT Define to activate river sediment point-sources.

SED_DENS Define to allow sediment to affect equation of state.

SED_MORPH Define to allow bottom model elevation to evolve.

SUSPLOAD Define to activate suspended load transport.

Nearshore options

WET_DRY Define to allow wetting and drying of cells.

NEARSHORE_MELLOR Define for radiation stress terms from waves.

NetCDF input/output

DEFLATE Define to set compression of NetCDF-4/HDF5 format files.

NETCDF4 Define to create NetCDF-4/HDF5 format files.

PARALLEL_IO Define to create NetCDF-4/HDF5 format files with MPI-I/O.

NO_READ_GHOST Define to not include ghost points during read/scatter.

NO_WRITE_GRID Define to omit writing grid arrays.

PERFECT_RESTART Define to include perfect restart variables.

READ_WATER Define to only read water points.

WRITE_WATER Define to only write water points.

RST_SINGLE Define to write single precision restart fields.

OUT_DOUBLE Define to write double precision output fields.

INLINE_2DIO Define to read/write 3D fields level by level.

6.8 Important parameters

The following is a list of the important parameters in the model. These are in **mod_param.F** and many of them are read from the standard input file as described in §7.1.12.

Lm Number of interior grid points in the ξ-direction.

Mm Number of interior grid points in the η-direction.

N Number of grid points in the vertical.

NAT Number of active tracers (usually 2 for temperature and salinity).

NBT Number of biological tracers. This will depend on the ecosystem model used.

NST Number of sediment tracers.

NPT Number of passive tracers.

NT Total number of tracer fields. NT = NAT+NBT+NST+NPT

Nfloats Number of Lagrangian floats to track.

Nstation Number of stations for station output.

MTC Maximum number of tidal constituents.

NtileI Number of tiles in the ξ-direction.

NtileJ Number of tiles in the η-direction.

6.9 Domain decomposition

ROMS supports serial, OpenMP, and MPI computations, with the user choosing between them at compile time. The serial code can also take advantage of multiple small tiles which can be sized to fit in cache. All are accomplished through domain decomposition in the horizontal. All of the horizontal operations are explicit with a relatively small footprint, so the tiling is a logical choice. Some goals in the parallel design of ROMS were:

- Minimize code changes.
- Don't hard-code the number of processes.

- MPI and OpenMP share the same basic structure.
- Don't break the serial optimizations.
- Same result as serial code for any number of processes.
- Portability—able to run on any (Unix) system.

First, some **cpp** options. If we're compiling for **MPI**, the option **-DMPI** gets added to the argument list for **cpp**. Then, in **globaldefs.h**, we have:

```
#if defined MPI
# define DISTRIBUTE
#endif
```

The rest of the code uses **DISTRIBUTE** to identify distributed memory jobs. The OpenMP case is more straightforward, with **-D_OPENMP** getting passed to **cpp** and **_OPENMP** being the tag to check within ROMS.

The whole horizontal ROMS grid is shown in Fig. 12. The computations are done over the cells inside the darker line; the cells are numbered 1 to **Lm** in the ξ-direction and 1 to **Mm** in the η-direction. Those looking ahead to running in parallel would be wise to include factors of two in their choice of **Lm** and **Mm**. ROMS will run in parallel with any values of **Lm** and **Mm**, but the computations might not be load-balanced.

6.9.1 ROMS internal numbers

A domain with tiles is shown in Fig. 13. The overlap areas are known as ghost points or halo points. Each tile is an MPI process, an OpenMP thread, or a discrete unit of computation in a serial run. The tile contains enough information to time-step all the interior points, so the number of ghost points is dictated by the footprint of the algorithm using the largest number of neighbor points. In ROMS, the halo area would be two grids points wide unless the MPDATA advection scheme is used, in which case it needs three. The variable **GHOST_POINTS** is set accordingly:

```
#if defined TS_MPDATA || defined UV_VIS4
# define GHOST_POINTS 3
# if defined DISTRIBUTE || defined EW_PERIODIC || defined NS_PERIODIC
#   define THREE_GHOST
# endif
#else
# define GHOST_POINTS 2
#endif
```

The number of tiles is set in the input file as **NtileI** and **NtileJ**. For an MPI job, the product of the two must equal the number of MPI processes. For an OpenMP job, the number of threads must be a multiple of the number of tiles. For instance, for **NtileI**= 4 and **NtileJ**= 6, you must have 24 MPI processes while 2, 3, 4, 6, 8, 12 and 24 are all valid numbers of OpenMP threads. Also, a serial run could have 24 tiles and would just compute them sequentially.

Once the input file has been read, we can compute the tile sizes:

```
ChunkSizeI = (Lm+NtileI-1)/NtileI
ChunkSizeJ = (Mm+NtileJ-1)/NtileJ
MarginI = (NtileI*ChunkSizeI-Lm)/2
MarginJ = (NtileJ*ChunkSizeJ-Mm)/2
```

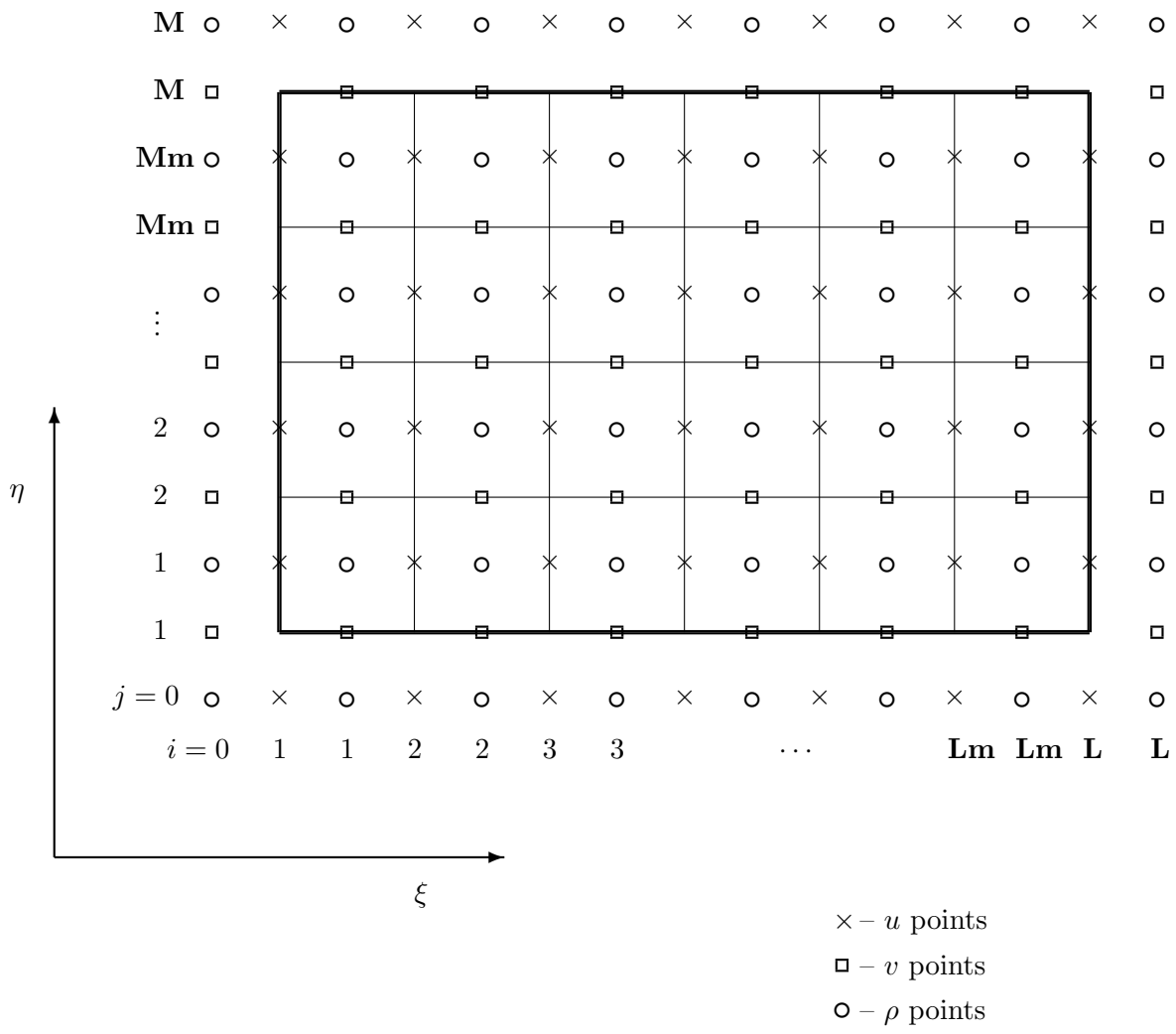

Figure 12: The whole grid. Note that there are **Lm** by **Mm** interior computational points. The points on the thick outer line and those outside it are provided by the boundary conditions.

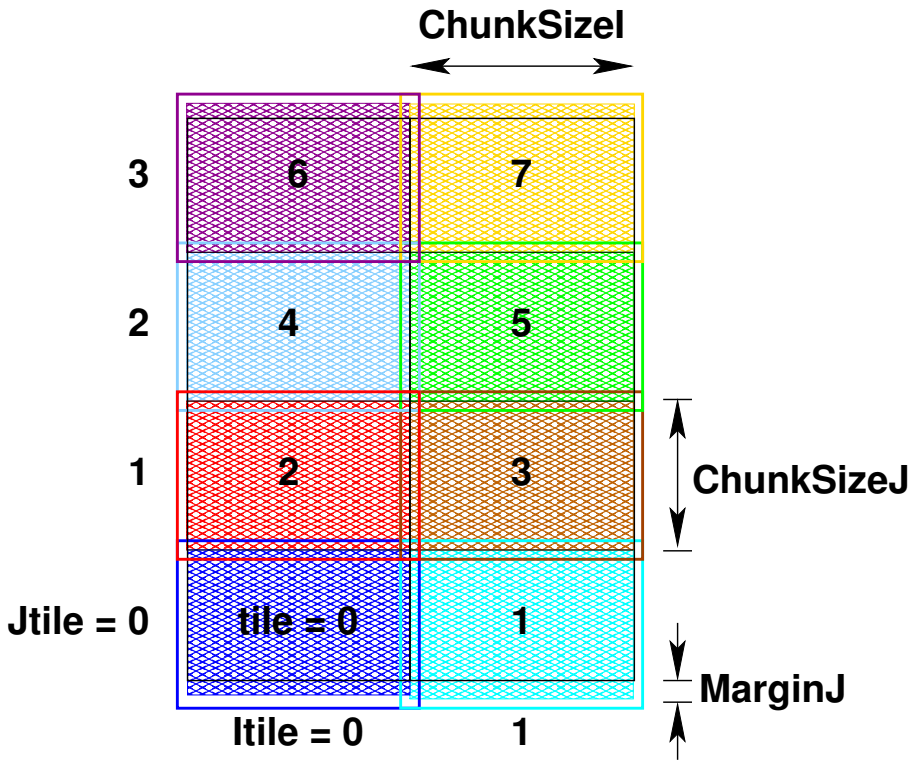

Figure 13: A tiled grid with some ROMS tile variables.

Some internal ROMS numbers are shown in Fig. 13 and are in the **BOUNDS** structure in **mod_param.F**. **MarginI** and **MarginJ** are zero if the numbers work out perfectly, i.e. **Lm/NtileI** and **Mm/NtileJ** are integers. The tile numbers match the MPI process numbers.

In picking a numbering scheme for indices within a tile, there are two common choices, as shown in Fig. 14. Each tile can be numbered from 1 to **ChunksizeI** or it can retain the numbering it would have in the whole grid. We have chosen this second option for ease when debugging features such as river inputs which apply to specific locations on the grid. It is simple to do using Fortran 90 dynamic memory allocation.

With the tile sizes known, we can assign beginning and ending indices for each tile. Some of the details depend on whether or not the domain is periodic in that direction, as shown in Fig. 15.

6.9.2 MPI exchange

For MPI jobs, the ghost points need to be updated between interior point computations. The routines **mp_exchange2d**, **mp_exchange3d** and **mp_exchange4d** can be used to update the halo points of up to four arrays at a time. Each of these routines call **tile_neighbors** to figure out which tiles are neighboring and whether or not there really is a neighboring tile on each side. The **mp_exchangexd** routines then call:

```
mpi_irecv
mpi_send
mpi_wait
```

The exchanges happen first in the east-west direction, then in the north-south direction, saving the need for diagonal exchanges. A figure with interior points colored by tile and grey halo points needing an update is shown in Fig. 16(a). The updated halo points are shown in Fig. 16(b).

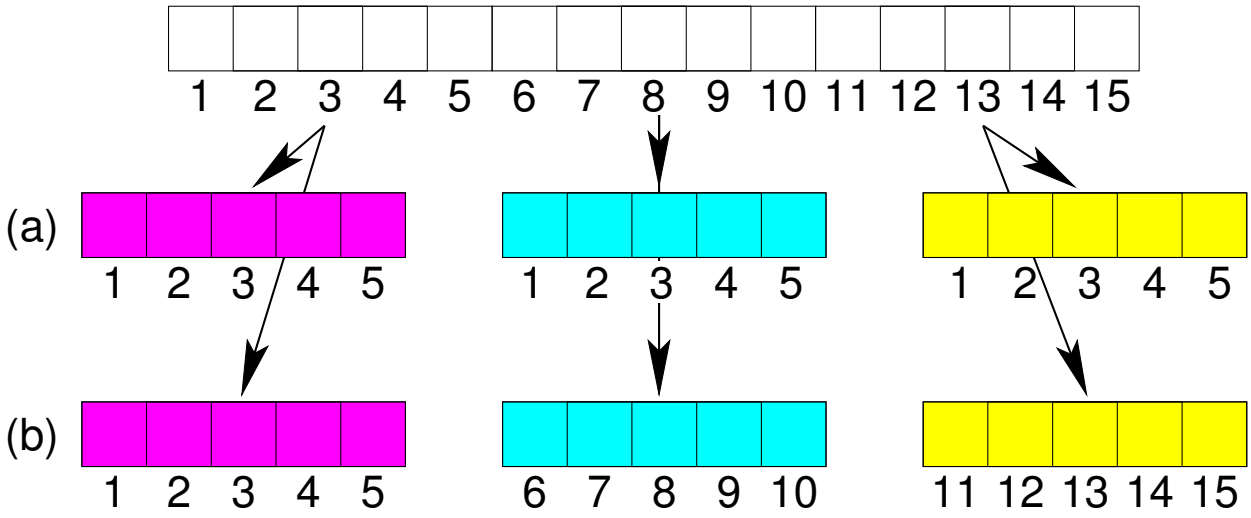

Figure 14: A choice of numbering schemes: (a) each tile is numbered the same, and (b) each tile retains the numbering of the parent domain.

6.9.3 Code syntax

In **main3d**, many function calls are surrounded by OpenMP parallel code such as:

```
!$OMP PARALLEL DO PRIVATE(thread,subs,tile) SHARED(ng,numthreads)
    DO thread=0,numthreads-1
      subs=NtileX(ng)*NtileE(ng)/numthreads
      DO tile=subs*thread,subs*(thread+1)-1,+1
        CALL set_data (ng, TILE)
      END DO
    END DO
!$OMP END PARALLEL DO
```

What isn't obvious from this is that the argument **TILE** means different things depending on if we're using OpenMP or MPI:

```
#ifdef DISTRIBUTE
# define TILE MyRank
#else
# define TILE tile
#endif
```

Likewise, **NtileX** also depends on whether or not we're using MPI:

```
#ifdef DISTRIBUTE
        NtileX(1:Ngrids)=1
#else
        NtileX(1:Ngrids)=NtileI(1:Ngrids)
#endif
```

In other words, for MPI, **TILE** becomes the process number and the loop is only executed once.

In looking at a typical routine that's called from **main3d**, the routine is usually quite short, calling a **__tile** version of itself in which the actual work happens:

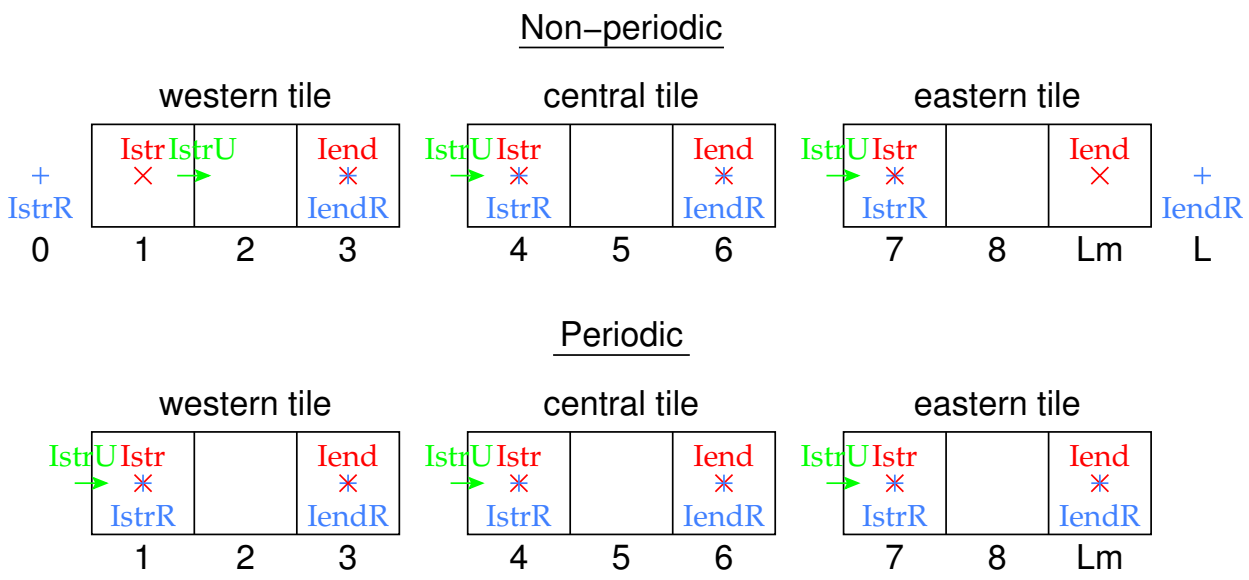

Figure 15: Some ROMS variables for tiles, for both a periodic and non-periodic case. Shown are the variables in the **i**-direction, the **j**-direction is similar.

```
SUBROUTINE set_data (ng, tile)
# include "tile.h"
  CALL set_data_tile (ng, tile,                          &
 &                    LBi, UBi, LBj, UBj,                &
 &                    IminS, ImaxS, JminS, JmaxS)
  RETURN
  END SUBROUTINE set_data
```

Here, there are two sets of array lower and upper bounds, those in the **LBi** family and those in the **IminS** family. Both depend on the **Istr** family shown in Fig. 15. The **IminS** family is for work arrays that are local to an MPI process or to an OpenMP thread, also local to a **__tile** routine. They are initialized:

```
IminS=BOUNDS(ng)%Istr(tile)-3
ImaxS=BOUNDS(ng)%Iend(tile)+3
JminS=BOUNDS(ng)%Jstr(tile)-3
JmaxS=BOUNDS(ng)%Jend(tile)+3
```

and used:

```
real(r8), dimension(IminS:ImaxS,JminS:JmaxS) ::  work1
real(r8), dimension(IminS:ImaxS,JminS:JmaxS) ::  work2
```

The **Istr** and **LBi** families are dimensioned by the number of tiles once it is known by **inp__par**:

```
DO ng=1,Ngrids
  Ntiles=NtileI(ng)*NtileJ(ng)-1
  allocate ( BOUNDS(ng) % LBi  (-1:Ntiles) )
                  :
  allocate ( BOUNDS(ng) % Jend (-1:Ntiles) )
END DO
```

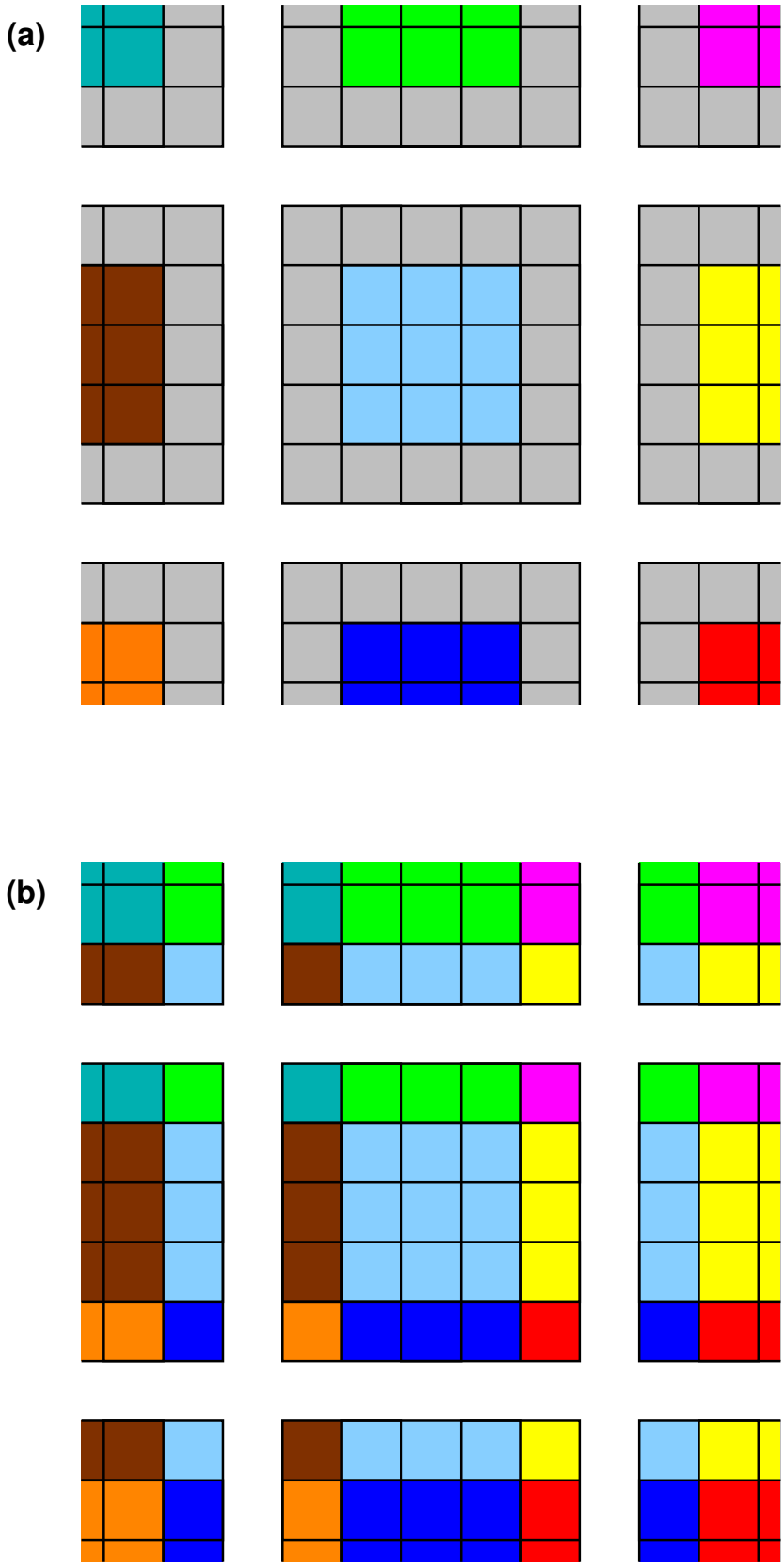

Figure 16: A tiled grid with out-of-date halo regions shown in grey and the interior points color-coded by tile: (a) before an exchange and (b) after an exchange.

They are then initialized in calls to the routines in **get_bounds.F**. If a tile is on the "western" edge (**Itile**= 0), then **UBi** is set to **LOWER_BOUND_I**.

```
Imin=LOWER_BOUND_I
        :
IF ((Itile.eq.-1).or.(Itile.eq.0)) THEN
  LBi=Imin
ELSE
  LBi=Istr-Nghost
END IF
```

Tracking the origins of **LOWER_BOUND_I**, we find it in **globaldefs.h**:

```
#ifdef EW_PERIODIC
#  define LOWER_BOUND_I -GHOST_POINTS
#else
#  define LOWER_BOUND_I 0
#endif
```

In the case of **set_data**, we are simply passing array indices for the tiled arrays. To access the tiled arrays from within **set_data_tile**, we need to **use** the relevant modules and then refer to the array with its full name:

```
  USE mod_forces
        :
  CALL set_2dfld_tile (ng, tile, iNLM, idCfra,             &
 &                     LBi, UBi, LBj, UBj,                 &
 &                     FORCES(ng)%cloudG,                  &
 &                     FORCES(ng)%cloud,                   &
 &                     update)
```

In other cases, the parent routine would have the **use**, then would pass the relevant array to the **_tile** routine:

```
  USE mod_grid
      :
  CALL prsgrd_tile (ng, tile,                              &
      :
 &                  GRID(ng) % Hz,                        &
      :
  SUBROUTINE prsgrd_tile (ng, tile,                        &
      :
 &                        Hz, z_r, z_w,                   &
      :
  real(r8), intent(in) :: Hz(LBi:,LBj:,:)
```

This allows the **_tile** routine to use **Hz** with the same syntax as the pre-parallel, pre-module code once had.

6.9.4 Input/output

In ROMS, the distributed memory I/O is all happening on the master process (0) unless you specifically ask it to use MPI-I/O, which requires both the **NETCDF4** and **PARALLEL_IO** **cpp** flags to be defined. If you do this, you will be reading and writing **HDF5** files and will need

to update your pre- and post-processing tools accordingly. I have tentatively tried the parallel I/O and found it to be exceedingly slow—I've been told since that this is the fault of the **NetCDF-4** layer sitting on top of **HDF5**—**HDF5** alone should be fast.

In the case of having all the I/O pass through the master process, we can still read and write classic NetCDF-3 files. Care must be taken though, in the event of an error. ROMS has been cleaned up so that the master process will broadcast its return state to the other processes and they can all die gracefully together when there is a problem.

An example of a routine which reads from disk is **get_grid**, called from **initial**. Each MPI process calls **get_grid**:

```
        CALL get_grid (ng, iNLM)
# ifdef DISTRIBUTE
        CALL mp_bcasti (ng, iNLM, exit_flag)
# endif
        if (exit_flag.ne.NoError) RETURN
```

If any one of the processes has trouble, it will enter into the **exit_flag** which is then shared by all.

To read in an array variable, all processes in **get_grid** uses **nf_fread2d** and friends:

```
            status=nf_fread2d(ng, model, ncname, ncGRDid(ng),      &
     &                        var_name(it), var_id(it),            &
     &                        0, gtype, Vsize,                     &
     &                        LBi, UBi, LBj, UBj,                  &
     &                        Fscl, Fmin, Fmax,                    &
     &                        GRID(ng) % rmask,                    &
     &                        GRID(ng) % rmask)
            IF (status.ne.nf90_noerr) THEN
              exit_flag=2
              ioerror=status
              EXIT
            END IF
```

Within **nf_fread2d**, we get to a call to the NetCDF library from just the master process:

```
      IF (InpThread) THEN
        status=nf90_get_var(ncid, ncvarid, wrk, start, total)
            :
      END IF
# ifdef DISTRIBUTE
      CALL mp_bcasti (ng, model, status)
# endif
      IF (status.ne.nf90_noerr) THEN
        exit_flag=2
        ioerror=status
        nf_fread2d=status
        RETURN
      END IF
```

At this point, the master process has the entire 2-D array stored in **wrk**. This then needs to be divvied out to the various tiles to their copy of the array in question (stored in the **A** argument to **nf_fread2d**):

```
# ifdef DISTRIBUTE
        CALL mp_scatter2d (ng, model, LBi, UBi, LBj, UBj,         &
     &                     Nghost, MyType, Amin, Amax,            &
#  if defined READ_WATER && defined MASKING
     &                     NWpts, SCALARS(ng)%IJwater(:,wtype),   &
#  endif
     &                     Npts, wrk, A)
```

Something similar happens when writing to output files.

7 Configuring ROMS for a Specific Application

This chapter describes the parts of ROMS for which the user is responsible when configuring it for a given application. Section 7.1 describes the process in a generic fashion while §7.2 and §7.3 step through the application of ROMS to upwelling/downwelling and wind-driven Northeast Pacific problems, respectively. As distributed, ROMS is ready to run quite a few examples, where the C preprocessor flags determine which is to be executed. Some of these examples are described in Haidvogel and Beckmann [24], some are listed here:

BASIN This is a rectangular, flat-bottomed basin with double-gyre wind forcing. When run, it produces a western boundary current flowing into a central "Gulf Stream" which goes unstable and generates eddies. The goal is to run adiabatically to study the homogenization of potential vorticity. It earned its nickname of Big Bad Basin by taking a long time to run and causing difficulties for the spectral versions of SPEM.

GRAV_ADJ The gravitational adjustment problem takes place in a long narrow domain which is initialized with dense water at one end and light water at the other. At time zero, the water is released and it generates two propagating fronts as the light water rushes to fill the top and the dense water rushes to fill the bottom. This configuration was used to test various advection schemes.

OVERFLOW This configuration is similar to the GRAV_ADJ problem, but is initialized with dense water in the shallow part of a domain with a sloping bottom.

SEAMOUNT The seamount test was used to test the pressure gradient errors. It has an idealized seamount in a periodic channel. See Beckmann and Haidvogel [4] and McCalpin [48] for more information.

UPWELLING The upwelling/downwelling example was contributed by Anthony Macks and Jason Middleton [45] and consists of a periodic channel with shelves on each side. There is along-channel wind forcing and the Coriolis term leads to upwelling on one side and downwelling on the other side. If you run it for several days without vertical mixing, you end up with dense water over light water.

Some NetCDF input files for the ROMS examples can be found under **Data/ROMS** in the ROMS distribution. The ASCII input files are under **ROMS/External**.

7.1 Configuring ROMS

The four main sections you need to change in ROMS are the **makefile** or **build.bash**, an include file with **cpp** options, any analytic functions, and the ASCII input file. If more realistic fields are desired, you will have to provide other input files as well, for instance for the grid and the wind forcing.

7.1.1 Case Name

First, you need to decide on a name for your particular application or configuration. This name is provided via the **ROMS_APPLICATION** in either the makefile or the build script. This name should be reasonably short, all uppercase, with spaces converted to underscores. For example, let's say we pick the name **WIKI_TEST**. This name gets defined during the build, so you can add code protected by **#ifdef WIKI_TEST** as needed. This would be a good time to either copy the makefile or the build Script to create one specific to this case prior to editing it.

7.1.2 Case-specific Include File

Each application has its own include file, included by **cppdefs.h**. The name of this file is the name of your application (**WIKI_TEST** here) turned into lower case, with '.h' appended (**wiki_test.h**). The location of this file is set by **MY_HEADER_DIR**, pointing to **User/Include** or some other location of your choosing.

The complete list of options to be set prior to compilation are listed in §6.7. Place those you need in the **wiki_test.h** file. These include algorithm choices (e.g. advection and turbulence closure schemes), boundary conditions, output options (averages, diagnostics, stations, floats), and application modules (biology, sediments). Each line should be of the form:

```
#define SOME_VAR
```

Note that any undefined variable need not be mentioned.

Also note that if you copy a predefined application from **ROMS/Include** as a template for your application, you must rename it. If you don't change the name, ROMS will use the one in **ROMS/Include** and your file will be ignored during the build procedure.

7.1.3 Functionals

Some of the cpp Options have names beginning with **ANA_**. For each one of these, you will be expected to provide an analytic expression for the field in question in the corresponding include file. These files are listed in §6.5 and their location is determined by **MY_ANALYTICAL_DIR**. You may chose to copy those from **User/Functionals** to some new directory and place your version of the assignments within

```
#ifdef WIKI_TEST
! Set weird and wonderful winds
   :
#endif
```

This makes it easy to search for later, if nothing else.

7.1.4 checkdefs.F

For each new **cpp** variable other than your application name, it is recommended that you also add the appropriate code to **checkdefs.F**, such as:

```
#ifdef SLEET
      IF (Master) WRITE(stdout,20) 'SLEET',                    &
     &   'Sleet falling on the ice option.'
      is=lenstr(Coptions)+1
      Coptions(is:is+7)=' SLEET,'
#endif /* ICE */
```

Note that the number "7" on the **Coptions** line must be set according to the length of the string you are adding. In this case 7 is for " SLEET,", including the comma and the space. Again, you do not need to do this for your application name (**WIKI_TEST** here), since **checkdefs** will print whatever is in **MyAppCPP**.

7.1.5 Model domain

One of the first things the user must decide is how many grid points to use, and can be afforded. There are three parameters in **ocean.in** which specify the grid size and one parameter for the number of active tracers:

Lm Number of finite-difference points in ξ.
Mm Number of finite-difference points in η.
N Number of finite-difference points in the vertical.
NAT Number of active tracers.

The number of biological tracers is set in the **biology.in** file. There are no constraints on these except **Lm** ≥ 2, **Mm** ≥ 2, **N** ≥ 2 and **NAT** ≥ 1. **Lm** and **Mm** should be at least 3 if the domain is periodic in that direction.

7.1.6 x, y grid

The subroutine **get_grid** or **ana_grid** is called by **initial** to set the grid arrays, the bathymetry, and the Coriolis parameter. Most of the simple test problems have their grid information specified in **ana_grid.h** in the directory **ROMS/Functionals**. More realistic problems require a NetCDF grid file, produced by the grid generation programs described in Wilkin and Hedstrom [86], by the Matlab **SeaGrid**, or by some other method. The variables which are read by **get_grid** are:

xl, el, spherical, f, h, pm, pn, x_rho, y_rho, lon_rho, lat_rho, angle.

If the grid is curved, **get_grid** will also read:

dndx, dmde.

Likewise, if **MASKING** is defined, it will read:

mask_rho, mask_u, mask_v, mask_psi.

7.1.7 ξ, η grid

Before providing initial conditions and boundary conditions, the user must understand the model grid. The fields are laid out on an Arakawa C grid as in Fig. 1. The overall grid is shown in Fig. 12. The thick outer line shows the position of the model boundary. The points inside this boundary are those which are advanced in time using the model physics. The points on the boundary and those on the outside must be supplied by the boundary conditions.

The three-dimensional model fields are carried in four-dimensional arrays, where the fourth array index refers to one of two or three time levels. The tracers have a fifth array index telling which tracer is being referred to. For instance, **itemp** $= 1$ refers to potential temperature while **isalt** $= 2$ refers to salinity. The integers i, j, and k are used throughout the model to index the three spatial dimensions:

i Index variable for the ξ-direction.
j Index variable for the η-direction.
k Index variable for the σ-direction. $k = 1$ refers to the bottom while $k =$ **N** refers to the surface.

7.1.8 Initial conditions

The initial values for the model fields are provided by either **ana_initial** or **get_state**. **get_state** is also used to read a restart file if the model is being restarted from a previous run.

Also in **initial**, **rho_eos** is called to initialize the density field. **rho_eos** also computes **rhoA**, the vertically averaged density, and **rhoS**, the density perturbation. Both **rhoA** and **rhoS** are used in the barotropic pressure gradient.

7.1.9 Equation of state

The equation of state is defined in the subroutine **rho_eos**. Two versions are provided in ROMS: a nonlinear $\rho = \rho(T, S, z)$ from Jackett and McDougall [32] and a linear $\rho(T, S)$. The linear form is

$$\rho = \mathbf{R0} - \mathbf{Tcoef} \cdot (T - T0) + \mathbf{Scoef} \cdot (S - S0)$$

or

$$\rho = \mathbf{R0} + \mathbf{Tcoef} \cdot (T - T0),$$

depending on whether or not **SALINITY** is defined. Specify which equation of state you would like to use with the **NONLIN_EOS** C preprocessor flag in your application include file. The linear coefficients **R0**, **T0**, **Tcoef**, **S0**, and **Scoef** are set in **ocean.in**. Note that we are computing *in situ* density from potential temperature and salinity. Some of the vertical mixing schemes require potential density and some other fields, which are computed by **rho_eos** as well.

7.1.10 Boundary conditions

The horizontal boundary conditions are provided by the subroutines in **u3dbc_im**, **v3dbc_im**, **u2dbc_im**, **v2dbc_im**, **t3dbc_im**, and **zetabc**. They are called every time-step and provide the boundary values for the fields $u, v, \overline{u}, \overline{v}$, all tracers, and ζ, respectively. They are currently configured for a closed basin, a periodic channel, a doubly periodic domain or a domain with various open boundary conditions. Each side is controlled independently with **WEST** being the **i=1** boundary, **EAST** being the **i=L** boundary, **SOUTH** being the **j=1** boundary, and **NORTH** being the **j=M** boundary. These choices are made via the **cpp** options in your include file.

Many of the choices for open boundaries require that the model have some boundary values for the field in question. These can be specified in the appropriate **ana_xxx.h** file for say **ANA_TOBC** or they can be read from a boundary NetCDF file. There is logic in **globaldefs.h** by which ROMS decides whether or not it needs to read a boundary file.

7.1.11 Model forcing

(a) Winds and thermal fluxes

There are two different ways to apply a wind forcing: as a surface momentum flux in the vertical viscosity term, or as a body force over the upper water column. Usually, we set the vertical σ-coordinate parameters to retain some resolution near the surface and apply the fluxes as boundary conditions to the vertical viscosity/diffusivity. In either case, the surface and bottom fluxes are either defined analytically, read from a forcing file, or computed inside ROMS using a bulk flux formula from the appropriate atmospheric fields (air temperature and winds, for instance). You must either edit the appropriate **ana_xxx.h** or create a NetCDF forcing file in the format expected by ROMS. Note that it is quite common to put the surface variables in the forcing file while having an analytic bottom heat flux. ROMS now has the capability of reading in a list of forcing fields—it can be convenient to have one file per field rather than stuffing tides, winds, river inputs all into one file.

In the past, our vertical resolution was relatively coarse and the vertical viscosity would have to have been unreasonably large for us to resolve the surface Ekman layer. If that is your situation, define **BODYFORCE** in **cppdefs.h** and provide a value for **levsfrc** in **ocean.in**. The forcing is applied over the levels from **levsfrc** to **N**. The above caution about vertical resolution also applies to the surface fluxes of T and S, although **BODYFORCE** only refers to wind stress, not the surface tracer fluxes.

(b) Climatology

One way to force the model is via a nudging to the tracer and/or momentum climatologies. Nudging to tracers was used in the North Atlantic simulations in sponge layers along the northern

and southern boundaries. Set the climatologies in **ana_tclima.h** or in a file read by **get_data**, set **TCLM_NUDGING** in **wiki_test.h** and also set the array **Tnudgcof** in **ana_nudgcoef.h**.

(c) Tides

There is also more than one way to force with tides. One way is to provide boundary conditions with enough temporal resolution to resolve the tides. Another is to provide ROMS with the tidal constituents at all grid points and to have ROMS reconstruct the tidal currents (**UV_TIDES**) and/or elevations (**SSH_TIDES**) for any given time. An example of such a tidal forcing file is in **Data/ROMS/Forcing/test_head_frc.nc**.

The non-trunk code with ice, etc. includes an option to add on the long-period tides (**TIDES_ASTRO**) from Foreman ([13] and [14]). There is also an option to include the tidal potential forcing term (**POT_TIDES**), requiring the tidal potential to be included in the tides forcing file.

(d) Rivers

Point sources can be used to provide river inflow to the model. These too can be specified in a forcing file if not provided via **ana_psource**.

7.1.12 ocean.in

ROMS expects to read a number of variables from an ASCII file as described in §2.5. Example input files are in **ROMS/External** with names like **ocean_grav_adj.in**, where "grav_adj" refers to the name of the application. Lines beginning with "!" are comments and will be ignored by ROMS on reading them.

The input is organized as key/value pairs, separated by one or two equals signs. It is possible for ROMS to run on more than one grid simultaneously, with the number of grids being set at compile time via the build script and/or the **makefile**. If there is one equals sign (=), ROMS will use the corresponding value for all grids. If there are two (==), ROMS will read a value for each grid. Thus far, our domains have used just one grid since the inter-grid coupling has not been released.

ROMS will ignore the parameters not needed by the current simulation, e.g., the GLS parameters will not be read if you are not using that mixing scheme. However, I believe all the example files contain all possible parameters (except the ice branch ones). The input parameters are in groups, as follows:

Header

TITLE A text string to put in the output files.

MyAppCPP The shorthand name for this application.

VARNAME The location of the **varinfo.dat** file containing information about fields to read/write from/to NetCDF files.

Grid-dimension parameters

Lm Number of i-direction INTERIOR RHO-points.

Mm Number of j-direction INTERIOR RHO-points.

N Number of vertical levels.

Nbed Number of sediment bed layers.

NAT Number of active tracers (usually 2).

NPT Number of passive tracers.

NCS Number of cohesive (mud) sediment tracers.

NNS Number of non-cohesive (sand) sediment tracers.

Domain-decomposition parameters

NtileI Number of i-direction partitions.

NtileJ Number of j-direction partitions.

Time-stepping parameters

NTIMES Number of time-steps to evolve the 3-D equations in the current run. This is actually the total number, including any previous segments of the same run. For instance, if you already did a three-month run and wish to continue for another three months, set **NTIMES** to the number of steps needed for six months.

DT Time-step in seconds for the 3-D equations.

NDTFAST Number of time-steps for the 2-D equations to be executed each **dt**.

Model iteration loops parameters

ERstr Starting ensemble run number.

ERend Ending ensemble run number.

Nouter Maximum number of 4DVAR outer loop iterations.

Ninner Maximum number of 4DVAR inner loop iterations.

Nintervals Number of time interval divisions for stochastic optimals computations.

Generalized Stability Theory (GST) parameters

NEV Number of eigenvalues.

NCV Number of eigenvectors.

Input/Output parameters

ROMS has several possible output files. The output files can include a restart file, a history file, an averages file, and a station file, for instance. The restart file often contains only two records with the older record being overwritten during the next write. The history file can contain a subset of the restart fields, for instance just the surface elevation and the surface temperature. The averages file contains time-averages of the model fields, for instance daily or monthly means, depending on **NAVG**. The station file contains timeseries for specified points, possibly quite frequently since each record is small. For some, machinery is in place to write multiple files, numbering them _0001, _0002, etc.

NRREC Record number of the restart file to read as the initial conditions. Set to 0 at the beginning of the run, -1 to read the latest record.

LcycleRST Logical, true to cycle between two records of the restart file.

NRST Number of time-steps between writing of restart fields.

NSTA Number of time-steps between writing fields into the stations file.

NFLT Number of time-steps between writing fields into the floats file.

NINFO Number of time-steps between calling **diag** to write some global information and check for NaN values.

History, average, diagnostic output parameters

LDEFOUT True for creating new output files for stations, history, floats, etc. If false, output is appended to these files.

NHIS Number of time-steps between writing history records.

NDEFHIS Number of time-steps between starting new history files.

NTSAVG Starting time-step for the accumulation of output time-averaged data. For instance, you might want to average over the last day of a thirty-day run.

NAVG Number of time-steps between writing time-averaged data into the averages file.

NDEFAVG Number of time-steps between starting new averages files.

NTSDIA Starting time-step for the accumulation of output diagnostics data. For instance, you might want to write diagnostics for the last day of a thirty-day run.

NDIA Number of time-steps between writing diagnostics data into the diagnostics file.

NDEFDIA Number of time-steps between starting new diagnostics files.

Tangent linear and adjoint output parameters

LcycleTLM Logical, true to cycle between two records

NTLM Number of time-steps between writing tangent linear data.

NDEFTLM Number of time-steps between starting new tangent linear files.

LcycleADJ Logical, true to cycle between two records of the restart file.

NADJ Number of time-steps between writing adjoint data.

NDEFADJ Number of time-steps between starting new adjoint files.

NSFF Number of time-steps between 4DVAR adjustment of surface forcing fluxes.

NOBC Number of time-steps between 4DVAR adjustment of open boundary fields.

Check-pointing GST restart parameters

LrstGST GST restart switch.

MaxIterGST Maximum number of iterations.

NGST Check-pointing interval.

Ritz GST parameter

Ritz_tol Relative accuracy of the Ritz values computed in the GST analysis.

Horizontal mixing of tracers

TNU2 Constant mixing coefficient for the horizontal Laplacian diffusion of each tracer. A value is expected for each of the **NAT+NPT** tracers.

TNU4 Constant mixing coefficient for the horizontal biharmonic diffusion of each tracer. A value is expected for each of the **NAT+NPT** tracers.

Horizontal viscosity coefficients

VISC2 Constant mixing coefficient for the horizontal Laplacian viscosity.

VISC4 Constant mixing coefficient for the horizontal biharmonic viscosity.

Vertical mixing coefficients for tracers

AKT_BAK Background vertical mixing coefficient for the tracers (**NAT+NPT** values).

Vertical mixing coefficient for momentum

AKV_BAK Background vertical mixing coefficient for momentum.

Turbulent closure parameters

AKK_BAK Background vertical mixing coefficient for turbulent kinetic energy.

AKP_BAK Background vertical mixing coefficient for turbulent kinetic energy.

TKENU2 Constant mixing coefficient for the horizontal Laplacian diffusion of turbulent kinetic energy.

TKENU4 Constant mixing coefficient for the horizontal biharmonic diffusion of turbulent kinetic energy.

Generic length-scale turbulence closure parameters

GLS_P Stability exponent.

GLS_M Turbulent kinetic energy exponent.

GLS_N Turbulent length scale exponent.

GLS_Kmin Minimum value of specific turbulent kinetic energy.

GLS_Pmin Minimum value of dissipation.

GLS_CMU0 Stability coefficient.

GLS_C1 Shear production coefficient.

GLS_C2 Dissipation coefficient.

GLS_C3M Buoyancy production coefficient (minus).

GLS_C3P Buoyancy production coefficient (plus).

GLS_SIGK Constant Schmidt number (non-dimensional) for turbulent kinetic energy diffusivity.

GLS_SIGP Constant Schmidt number (non-dimensional) for turbulent generic statistical field, "psi".

Constants used in surface TKE flux computation

CHARNOK_ALPHA Charnok surface roughness.

ZOS_HSIG_ALPHA Roughness from wave amplitude.

SZ_ALPHA Roughness from wave dissipation.

CRGBAN_CW Craig and Banner wave breaking.

Bottom drag coefficients

RDRG Linear bottom drag coefficient.

RDRG2 Quadratic bottom drag coefficient.

Zob Bottom roughness.

Zos Surface roughness (under ice shelves).

Height of atmospheric measurements for bulk flux parameterizations

BLK_ZQ Height of air humidity values.

BLK_ZT Height of air temperature values.

BLK_ZW Height of wind values.

Wetting and drying parameter

DCRIT Minimum depth for dry cells.

Various parameters

WTYPE Jerlov water type.

LEVSFRC Deepest level to apply surface momentum stresses as a body force. Used when the C-preprocessor option BODYFORCE is defined.

LEVBFRC Shallowest level to apply bottom momentum stresses as a body force. Used when the C-preprocessor option BODYFORCE is defined.

Vertical coordinate parameters

Vtransform Transformation equation, 1 for old style.

Vstretching Stretching function, 1 for old style.

Vertical σ-coordinates parameters

theta_s σ-coordinate surface control parameter, $[0 < \mathbf{theta\_s} < 20]$.

theta_b σ-coordinate bottom control parameter, $[0 < \mathbf{theta\_b} < 1]$.

Tcline Width of the surface or bottom boundary layer in which higher vertical resolution is required during stretching.

Mean Density and Brunt-Väisälä frequency

RHO0 Mean density used in the Boussinesq approximation.

BVF_BAK Background Brunt-Väisälä frequency squared.

Time parameters

DSTART Time stamp assigned to model initialization (days).

TIDE_START Time of tidal origin relative to model origin.

TIME_REF Reference time in the format yyyymmdd.dd or else a special value for specific calendars as documented in the **ocean.in** files.

Nudging time scales

TNUDG Time scale (days) of nudging towards tracer climatology at the interior and at the boundaries. A value is expected for each active tracer.

ZNUDG Time scale (days) of nudging towards free surface climatology at the interior and at the boundaries.

M2NUDG Time scale (days) of nudging towards 2-D momentum climatology at the interior and at the boundaries.

M3NUDG Time scale (days) of nudging towards 3-D momentum climatology at the interior and at the boundaries.

Open boundary factor

OBCFAC Ratio of inflow and outflow nudging time scales.

Linear equation of state parameters

R0 Background density value used in the linear equation of state.

T0 Background potential temperature constant.

S0 Background salinity constant.

TCOEF Thermal expansion coefficient in the linear equation of state.

SCOEF Saline contraction coefficient in the linear equation of state.

Slipperiness parameter

gamma2 Slipperiness variable, either 1.0 (free slip) or -1.0 (no slip).

Adjoint sensitivity time parameters

DstrS Starting day.

DendS Ending day.

Adjoint sensitivity vertical level parameters

KstrS Starting level.

KendS Ending level.

Adjoint sensitivity logical parameters

Lstate(isFsur) Free surface.

Lstate(isUbar) 2D u-momentum.

Lstate(isVbar) 2D v-momentum.

Lstate(isUvel) 3D u-momentum.

Lstate(isVvel) 3D v-momentum.

Lstate(isTvar) Tracers (NT values expected).

Stochastic optimals time scale

SO_decay Stochastic optimals time decorrelation scale (days) assumed for red noise processes.

Logicals for stochastic optimals

SOstate(isUstr) Surface u-stress.

SOstate(isVstr) Surface v-stress.

Lstate(isTsur) Surface tracer flux (NT values expected).

Stochastic optimals surface standard deviations

SO_sdev(isUstr) Surface u-stress.

SO_sdev(isVstr) Surface v-stress.

SO_sdev(isTsur) Surface tracer flux (NT values expected).

Logical switches to activate the writing of history/averages fields

Hout(idUvel) 3-D u-velocity component.

Hout(idVvel) 3-D v-velocity component.

Hout(idWvel) 3-D w-velocity component.

Hout(idOvel) 3-D Ω vertical velocity.

Hout(idUbar) 2-D u-velocity component.

Hout(idVbar) 2-D v-velocity component.

Hout(idFsur) Free-surface.

Hout(idBath) Time-dependent bathymetry.

Hout(idTvar) Tracer type variables: potential temperature, salinity, etc.

Hout(idUsms) Surface u-stress.

Hout(idVsms) Surface v-stress.

Hout(idUbms) Bottom u-stress.

Hout(idVbms) Bottom v-stress.

Hout(idUbrs) Bottom u-current stress.

Hout(idVbrs) Bottom v-current stress.

Hout(idUbws) Bottom u-wave stress.

Hout(idVbws) Bottom v-wave stress.

Hout(idUbcs) Bottom max wave-current u-stress.

Hout(idVbcs) Bottom max wave-current v-stress.

Hout(idUbot) Bed wave orbital u-velocity.

Hout(idVbot) Bed wave orbital v-velocity.

Hout(idUbur) Bottom max wave-current u-stress.

Hout(idVbur) Bottom max wave-current v-stress.

Hout(idW2xx) 2D radiation stress, S_{xx} component.

Hout(idW2xy) 2D radiation stress, S_{xy} component.

Hout(idW2yy) 2D radiation stress, S_{yy} component.

Hout(idU2rs) 2D u-radiation stress.

Hout(idV2rs) 2D v-radiation stress.

Hout(idU2Sd) 2D u-Stokes velocity.

Hout(idV2Sd) 2D v-Stokes velocity.

Hout(idW3xx) 3D radiation stress, S_{xx} component.

Hout(idW3xy) 3D radiation stress, S_{xy} component.

Hout(idW3yy) 3D radiation stress, S_{yy} component.

Hout(idW3zx) 3D radiation stress, S_{zx} component.

Hout(idW3zy) 3D radiation stress, S_{zy} component.

Hout(idU3rs) 3D u-radiation stress.

Hout(idV3rs) 3D v-radiation stress.

Hout(idU3Sd) 3D u-Stokes velocity.

Hout(idV3Sd) 3D v-Stokes velocity.

Hout(idWamp) Wave height.

Hout(idWlen) Wave length.

Hout(idWdir) Wave direction.

Hout(idTsur) Surface net heat and salt flux (NAT values).

Hout(idLhea) Latent heat flux.

Hout(idShea) Sensible heat flux.

Hout(idLrad) Longwave radiation flux.

Hout(idSrad) Shortwave radiation flux.

Hout(idEmPf) E-P flux.

Hout(idevap) Evaporation rate.

Hout(idrain) Precipitation rate.

Hout(idDano) Density anomaly.

Hout(idVvis) Vertical viscosity coefficient.

Hout(idTdif) Vertical diffusion coefficient for temperature.

Hout(idSdif) Vertical diffusion coefficient for salinity.

Hout(idHsbl) Depth of the surface boundary layer.

Hout(idHbbl) Depth of the bottom boundary layer.

Hout(idMtke) Turbulent kinetic energy.

Hout(idMtls) Turbulent length scale.

Ice fields

Hout(idUice) Ice u-velocity.

Hout(idVice) Ice v-velocity.

Hout(idAice) Ice concentration.

Hout(idHice) Ice thickness.

Hout(idTice) Ice surface temperature.

Hout(idHsno) Snow thickness.

Hout(idTimid) Ice internal temperature.

Hout(idSfwat) Surface water (melt ponds).

Hout(idTauiw) Tauiw.

Hout(idChuiw) Chuiw.

Hout(idAgeice) Ice age.

Hout(idSig11) Ice internal stress, 11 component.

Hout(idSig12) Ice internal stress, 12 component.

Hout(idSig22) Ice internal stress, 22 component.

Hout(idS0mk) Under-ice salinity.

Hout(idT0mk) under-ice temperature.

Hout(idWfr) Frazil ice growth rate.

Hout(idWai) Air-ice melt rate.

Hout(idWao) Air-ocean ice growth rate.

Hout(idWio) Ice-ocean ice melt/growth rate.

Hout(idWro) Surface water runoff rate.

Inert (passive) tracers

Hout(inert) Passive tracers (NPT values).

Sediment tracers

Hout(idBott) Sediment tracers (MBOTP values).

User parameters

NUSER Number of user parameters

USER Values of user parameters (NUSER values).

NetCDF-4/HDF5 parameters

NC_SHUFFLE If non-zero, turn on shuffle filter.

NC_DEFLATE If non-zero, turn on deflate filter.

NC_DLEVEL Deflate level (0–9).

Input NetCDF file names

GRDNAME Grid file.

ININAME Initial conditions file.

ITLNAME Initial tangent linear file.

IRPNAME Initial representer file.

IADNAME Initial adjoint file.

CLMNAME Climatology file.

BRYNAME Boundary condition file.

FWDNAME Forward model file.

ADSNAME Adjoint sensitivity functional file.

Forcing NetCDF files

NFFILES Number of forcing files.

FRCNAME Forcing files.

Output NetCDF file names

GSTNAME GST analysis restart file.

RSTNAME Restart file.

HISNAME History file.

TLMNAME Tangent linear file.

TLFNAME Impulse forcing file (for tangent linear model).

ADJNAME Adjoint file.

AVGNAME Averages file.

STANAME Stations file.

FLTNAME Lagrangian floats file.

ASCII input file names

APARNAM Assimilation parameters.

SPOSNAM Stations positions.

FPOSNAM Initial drifter positions.

IPARNAM Ice parameters.

BPARNAM Biology parameters.

SPARNAM Sediment parameters.

USRNAME User's generic input.

The bottom of the sample files contain comments describing some of these in greater detail.

7.1.13 User variables and subroutines

It is possible for the user to add new variables and functionality, though it is discouraged. The design goal is to isolate the most common features a user would change to the **cpp** switches (§6.7), the **ana_xx.h** files (§6.5) and the ASCII input file (§7.1.12). A query on the ROMS forum might be in order if you have something specific in mind.

If you do choose to add bits, know that the **makefile** fragments called **Module.mk** will attempt to compile anything with a **.F** extension in the directories already populated with such code.

7.2 Upwelling/Downwelling Example

The application which ROMS is configured to run as distributed is a wind-driven upwelling and downwelling example, described in Macks and Middleton [45]. There is a shelf on each wall of a periodic channel and an along-channel wind forcing, which drives upwelling at one wall and downwelling at the other. This problem depends on the Ekman layer, so a surface stress is used with vertical viscosity. The Ekman depth is estimated to be 9 m if $A_v = 0.01 m^2/s$, so the vertical grid spacing must resolve this. The maximum depth is 150 m and our choice of the vertical grid parameters leads to a surface Δz of 4.0 m.

7.2.1 cppdefs.h

The C preprocessor variable **UPWELLING** is used for the upwelling configuration of the model. The **makefile** will direct **cppdefs.h** to include the file **upwelling.h**:

```
#define UV_ADV
#define UV_COR
#define UV_LDRAG
#define UV_VIS2
#undef  MIX_GEO_UV
#define MIX_S_UV
#define TS_U3HADVECTION
#define TS_C4VADVECTION
#undef  TS_MPDATA
#define DJ_GRADPS
#define TS_DIF2
#undef  TS_DIF4
#undef  MIX_GEO_TS
#define MIX_S_TS
```

```
#define SALINITY
#define SOLVE3D
#define SPLINES
#define AVERAGES
#define DIAGNOSTICS_TS
#define DIAGNOSTICS_UV
#define EW_PERIODIC

#define ANA_GRID
#define ANA_INITIAL
#define ANA_SMFLUX
#define ANA_STFLUX
#define ANA_SSFLUX
#define ANA_BTFLUX
#define ANA_BSFLUX

#if defined GLS_MIXING || defined MY25_MIXING
# define KANTHA_CLAYSON
# define N2S2_HORAVG
#else
# define ANA_VMIX
#endif

#if defined BIO_FENNEL  || defined ECOSIM || \
    defined NPZD_POWELL || defined NEMURO || \
    defined BIO_UMAINE
# define ANA_BIOLOGY
# define ANA_SPFLUX
# define ANA_BPFLUX
# define ANA_SRFLUX
#endif

#if defined NEMURO
# define HOLLING_GRAZING
# undef  IVLEV_EXPLICIT
#endif

#ifdef BIO_FENNEL
# define CARBON
# define DENITRIFICATION
# define BIO_SEDIMENT
# define DIAGNOSTICS_BIO
#endif

#ifdef BIO_UMAINE
# define OXYGEN
# undef CARBON
#endif

#ifdef PERFECT_RESTART
```

```
# undef  AVERAGES
# undef  DIAGNOSTICS_BIO
# undef  DIAGNOSTICS_TS
# undef  DIAGNOSTICS_UV
# define OUT_DOUBLE
#endif
```

Here we have declared that we want a periodic channel (**EW_PERIODIC**) but no masking. There is salinity but we're using a linear equation of state. The momentum equations have advection, Coriolis force and pressure gradients. There is both horizontal viscosity and diffusion, but they are along constant σ-surfaces and if you check the input file, you find that the horizontal diffusion is set to zero.

There are ifdefs for various biology cases, none of which have been defined. Likewise, we are using the default of **ANA_VMIX** as distributed. We are asking for many other analytic functions too, including the grid. We are asking for diagnostic output with the **DIAGNOSTICS_TS** and **DIAGNOSTICS_UV**.

7.2.2 Model domain

The flow does not vary in x, so **Lm** can be small. Set the values for **Lm**, **Mm**, **N** and **NT** in the input file:

```
   Lm == 41            ! Number of I-direction INTERIOR RHO-points
   Mm == 80            ! Number of J-direction INTERIOR RHO-points
    N == 16            ! Number of vertical levels

  NAT =  2             ! Number of active tracers (usually, 2)
```

7.2.3 ana_grid

For this geometry one has a choice of using one of the external grid-generation programs or of using **ana_grid** to create the grid analytically. The code in **ana_grid.h** was modified to produce a bathymetry with a shelf on both walls of the channel when **UPWELLING** is defined. The fluid depth ranges from 27 m on the shelves to 150 m in the center of the channel. The horizontal grid spacing is uniform at 1 km and the Coriolis parameter f is set to a constant value suitable for Sydney, Australia.

7.2.4 Initial conditions and the equation of state

We would like the initial conditions to be a motionless fluid with an exponential stratification. The **UPWELLING** section of **ana_initial.h** is configured accordingly.

The stratification can be provided by either T or S, or by both T and S. For simplicity we will only have an active temperature field and we will use the linear equation of state by setting **NONLIN_EOS** to **#undef**. We want the density to be 26.35 at the bottom and 24.22 at the top with an e-folding scale of 50 meters. The initial temperature is set to **T0**$+8e^{z/50}$ in **ana_initial**. The linear equation of state parameters are set in **ocean_upwelling.in**:

```
   R0 == 1027.0d0                ! kg/m3
   T0 == 14.0d0                  ! Celsius
   S0 == 35.0d0                  ! PSU
TCOEF == 1.7d-4                  ! 1/Celsius
SCOEF == 0.0d0                   ! 1/PSU
```

Since density does not depend on salinity, we have a choice of how to handle the second tracer. The salinity is set to a uniform value of **S0**, though it could be left out entirely if we undefine **SALINITY** and set **NAT** to 1.

7.2.5 Boundary conditions

The option **EW_PERIODIC** has been chosen for the eastern and western edges. None of the open boundary options have been selected for the Northern and Southern edges; they are both then given the default wall conditions and no boundary values are required.

7.2.6 Model forcing

In this problem we want to resolve the surface Ekman layer and to use a surface wind stress rather than a body force. We want the amplitude of the wind to ramp up with time so we modify **ana_smflux.h** accordingly. The wind will build to an amplitude of 0.1 Pascals / ρ_o, or $10^{-4} m^2 s^{-2}$.

We need to edit **ana_vmix.h** to make sure that the vertical viscosity **Akv** is set to the value we want. This must be large at the surface ($10^{-2} m^2 s^{-1}$) to create a thick Ekman layer, but has been chosen to decrease with depth. We also need to check that **ana_sbflux**, **ana_stflux**, etc. are set correctly, in this case taking the default of zero rather than explicitly setting anything for **UPWELLING**. However, we do set **ana_srflux** to be non-zero in case we opt to turn on one of the biology models.

7.2.7 ocean.in

The model has been set up to run for five days with an internal time-step of 300 s and an external time-step of 10 s.

```
      NTIMES == 1440
          DT == 300.0d0
     NDTFAST == 30
```

We will write history, averages, and diagnostics records every 1/4 day, restart records once a day.

```
       NRREC == 0
  LcycleRST == T
        NRST == 288
    LDEFOUT == T
       NHIS == 72
    NDEFHIS == 0
     NTSAVG == 1
       NAVG == 72
    NDEFAVG == 0
     NTSDIA == 1
       NDIA == 72
    NDEFDIA == 0
```

The value of the linear bottom friction coefficient **rdrg** is set to 3.0×10^{-4} and the channel walls are set to be free-slip:

```
       RDRG == 3.0d-04                  ! m/s
     GAMMA2 == 1.0d0
```

The vertical stretching is set to a modest value of **theta_s**= 3:

```
 Vtransform == 1                         ! transformation equation
Vstretching == 1                         ! stretching function
    THETA_S == 3.0d0                     ! surface stretching parameter
    THETA_B == 0.0d0                     ! bottom  stretching parameter
     TCLINE == 50.0d0                    ! critical depth (m)
```

7.2.8 Output

The model writes some information to standard out, after setting **ninfo** to 72:

```
 Process Information:

 Thread #  0 (pid=   32022) is active.

 Model Input Parameters:  ROMS/TOMS version 3.2
                          Friday - October 30, 2009 -  9:11:20 AM
 ------------------------------------------------------------------------------

 Wind-Driven Upwelling/Downwelling over a Periodic Channel

 Operating system : Linux
 CPU/hardware     : x86_64
 Compiler system  : pgi
 Compiler command : /usr/local/pkg/pgi/current/linux86-64/6.1/bin/pgf90
 Compiler flags   :  -O3 -tp k8-64 -Mfree

 SVN Root URL  : https://www.myroms.org/svn/omlab/branches/kate/trunk
 SVN Revision  :

 Local Root    : /export/staffdata/kate/roms/kate_svn
 Header Dir    : /export/staffdata/kate/roms/kate_svn/ROMS/Include
 Header file   : upwelling.h
 Analytical Dir: /export/staffdata/kate/roms/kate_svn/ROMS/Functionals

 Resolution, Grid 01: 0041x0080x016,  Parallel Threads:  1,  Tiling: 001x001


 Physical Parameters, Grid: 01
 =============================

       1440  ntimes          Number of timesteps for 3-D equations.
    300.000  dt              Timestep size (s) for 3-D equations.
         30  ndtfast         Number of timesteps for 2-D equations between
                               each 3D timestep.
          1  ERstr           Starting ensemble/perturbation run number.
          1  ERend           Ending ensemble/perturbation run number.
          0  nrrec           Number of restart records to read from disk.
          T  LcycleRST       Switch to recycle time-records in restart file.
        288  nRST            Number of timesteps between the writing of data
                               into restart fields.
          1  ninfo           Number of timesteps between print of information
```

```
                        to standard output.
         T  ldefout         Switch to create a new output NetCDF file(s).
        72  nHIS            Number of timesteps between the writing fields
                             into history file.
         1  ntsAVG          Starting timestep for the accumulation of output
                             time-averaged data.
        72  nAVG            Number of timesteps between the writing of
                             time-averaged data into averages file.
         1  ntsDIA          Starting timestep for the accumulation of output
                             time-averaged diagnostics data.
        72  nDIA            Number of timesteps between the writing of
                             time-averaged data into diagnostics file.
0.0000E+00  tnu2(01)        Horizontal, harmonic mixing coefficient (m2/s)
                             for tracer 01: temp
0.0000E+00  tnu2(02)        Horizontal, harmonic mixing coefficient (m2/s)
                             for tracer 02: salt
5.0000E+00  visc2           Horizontal, harmonic mixing coefficient (m2/s)
                             for momentum.
1.0000E-06  Akt_bak(01)     Background vertical mixing coefficient (m2/s)
                             for tracer 01: temp
1.0000E-06  Akt_bak(02)     Background vertical mixing coefficient (m2/s)
                             for tracer 02: salt
1.0000E-05  Akv_bak         Background vertical mixing coefficient (m2/s)
                             for momentum.
3.0000E-04  rdrg            Linear bottom drag coefficient (m/s).
3.0000E-03  rdrg2           Quadratic bottom drag coefficient.
2.0000E-02  Zob             Bottom roughness (m).
         1  Vtransform      S-coordinate transformation equation.
         1  Vstretching     S-coordinate stretching function.
3.0000E+00  theta_s         S-coordinate surface control parameter.
0.0000E+00  theta_b         S-coordinate bottom  control parameter.
    50.000  Tcline          S-coordinate surface/bottom layer width (m) used
                             in vertical coordinate stretching.
  1025.000  rho0            Mean density (kg/m3) for Boussinesq approximation.
     0.000  dstart          Time-stamp assigned to model initialization (days).
      0.00  time_ref        Reference time for units attribute (yyyymmdd.dd)
0.0000E+00  Tnudg(01)       Nudging/relaxation time scale (days)
                             for tracer 01: temp
0.0000E+00  Tnudg(02)       Nudging/relaxation time scale (days)
                             for tracer 02: salt
0.0000E+00  Znudg           Nudging/relaxation time scale (days)
                             for free-surface.
0.0000E+00  M2nudg          Nudging/relaxation time scale (days)
                             for 2D momentum.
0.0000E+00  M3nudg          Nudging/relaxation time scale (days)
                             for 3D momentum.
0.0000E+00  obcfac          Factor between passive and active
                             open boundary conditions.
    14.000  T0              Background potential temperature (C) constant.
    35.000  S0              Background salinity (PSU) constant.
```

```
  1027.000  R0              Background density (kg/m3) used in linear Equation
                               of State.
1.7000E-04  Tcoef           Thermal expansion coefficient (1/Celsius).
0.0000E+00  Scoef           Saline contraction coefficient (1/PSU).
     1.000  gamma2          Slipperiness variable: free-slip (1.0) or
                                                   no-slip (-1.0).
         T  Hout(idFsur)    Write out free-surface.
         T  Hout(idUbar)    Write out 2D U-momentum component.
         T  Hout(idVbar)    Write out 2D V-momentum component.
         T  Hout(idUvel)    Write out 3D U-momentum component.
         T  Hout(idVvel)    Write out 3D V-momentum component.
         T  Hout(idWvel)    Write out W-momentum component.
         T  Hout(idOvel)    Write out omega vertical velocity.
         T  Hout(idTvar)    Write out tracer 01: temp
         T  Hout(idTvar)    Write out tracer 02: salt

Output/Input Files:

            Output Restart File:  ocean_rst.nc
            Output History File:  ocean_his.nc
          Output Averages File:  ocean_avg.nc
       Output Diagnostics File:  ocean_dia.nc
   IO Variable Information File:  ROMS/External/varinfo.dat

Tile partition information for Grid 01:  0041x0080x0016  tiling: 001x001

    tile     Istr     Iend     Jstr     Jend     Npts

Number of tracers:              2
       0        1       41        1       80    52480

Tile minimum and maximum fractional grid coordinates:
  (interior points only)

    tile     Xmin     Xmax     Ymin     Ymax     grid

       0    -2.50    43.50    -0.50    82.50  RHO-points

       0    -2.50    43.50    -0.50    82.50    U-points

       0    -2.50    43.50    -0.50    82.50    V-points

Activated C-preprocessing Options:

UPWELLING           Wind-Driven Upwelling/Downwelling over a Periodic Channel
ANA_BSFLUX          Analytical kinematic bottom salinity flux.
ANA_BTFLUX          Analytical kinematic bottom temperature flux.
ANA_GRID            Analytical grid set-up.
ANA_INITIAL         Analytical initial conditions.
ANA_SMFLUX          Analytical kinematic surface momentum flux.
```

```
ANA_SSFLUX          Analytical kinematic surface salinity flux.
ANA_STFLUX          Analytical kinematic surface temperature flux.
ANA_VMIX            Analytical vertical mixing coefficients.
ASSUMED_SHAPE       Using assumed-shape arrays.
AVERAGES            Writing out time-averaged fields.
DIAGNOSTICS_TS      Computing and writing tracer diagnostic terms.
DIAGNOSTICS_UV      Computing and writing momentum diagnostic terms.
DJ_GRADPS           Parabolic Splines density Jacobian (Shchepetkin, 2002).
DOUBLE_PRECISION    Double precision arithmetic.
EW_PERIODIC         East-West periodic boundaries.
MIX_S_TS            Mixing of tracers along constant S-surfaces.
MIX_S_UV            Mixing of momentum along constant S-surfaces.
NONLINEAR           Nonlinear Model.
!NONLIN_EOS         Linear Equation of State for seawater.
POWER_LAW           Power-law shape time-averaging barotropic filter.
PROFILE             Time profiling activated .
!RST_SINGLE         Double precision fields in restart NetCDF file.
SALINITY            Using salinity.
SOLVE3D             Solving 3D Primitive Equations.
SPLINES             Conservative parabolic spline reconstruction.
TS_U3HADVECTION     Third-order upstream horizontal advection of tracers.
TS_C4VADVECTION     Fourth-order centered vertical advection of tracers.
TS_DIF2             Harmonic mixing of tracers.
UV_ADV              Advection of momentum.
UV_COR              Coriolis term.
UV_U3HADVECTION     Third-order upstream horizontal advection of 3D momentum.
UV_C4VADVECTION     Fourth-order centered vertical advection of momentum.
UV_LDRAG            Linear bottom stress.
UV_VIS2             Harmonic mixing of momentum.
VAR_RHO_2D          Variable density barotropic mode.

INITIAL: Configuring and initializing forward nonlinear model ...


Vertical S-coordinate System:

level   S-coord     Cs-curve          at_hmin  over_slope   at_hmax

   16   0.0000000   0.0000000           0.000       0.000     0.000
   15  -0.0625000  -0.0188264          -1.575      -2.750    -3.925
   14  -0.1250000  -0.0383166          -3.150      -5.541    -7.932
   13  -0.1875000  -0.0591578          -4.725      -8.417   -12.108
   12  -0.2500000  -0.0820849          -6.300     -11.422   -16.544
   11  -0.3125000  -0.1079063          -7.875     -14.608   -21.342
   10  -0.3750000  -0.1375324          -9.450     -18.032   -26.614
    9  -0.4375000  -0.1720078         -11.025     -21.758   -32.492
    8  -0.5000000  -0.2125480         -12.600     -25.863   -39.126
    7  -0.5625000  -0.2605826         -14.175     -30.436   -46.696
    6  -0.6250000  -0.3178051         -15.750     -35.581   -55.412
    5  -0.6875000  -0.3862333         -17.325     -41.426   -65.527
```

```
    4  -0.7500000  -0.4682798         -18.900     -48.121   -77.341
    3  -0.8125000  -0.5668375         -20.475     -55.846   -91.216
    2  -0.8750000  -0.6853816         -22.050     -64.818  -107.586
    1  -0.9375000  -0.8280918         -23.625     -75.298  -126.971
    0  -1.0000000  -1.0000000         -25.200     -87.600  -150.000

Time Splitting Weights: ndtfast =  30    nfast =  42

   Primary            Secondary            Accumulated to Current Step

 1-0.0008094437383769 0.0333333333333333-0.0008094437383769 0.0333333333333333
 2-0.0014053566728197 0.0333603147912792-0.0022148004111966 0.0666936481246126
 3-0.0017877524645903 0.0334071600137066-0.0040025528757869 0.1001008081383191
 4-0.0019566842408176 0.0334667517625262-0.0059592371166046 0.1335675599008453
 5-0.0019122901320372 0.0335319745705535-0.0078715272486418 0.1670995344713988
 6-0.0016548570247459 0.0335957175749547-0.0095263842733877 0.2006952520463535
 7-0.0011849025289723 0.0336508794757796-0.0107112868023601 0.2343461315221331
 8-0.0005032751608632 0.0336903762267453-0.0112145619632232 0.2680365077488784
 9 0.0003887272597151 0.0337071520654408-0.0108258347035082 0.3017436598143192
10 0.0014892209965583 0.0336941944901169-0.0093366137069498 0.3354378543044362
11 0.0027955815694920 0.0336445537902317-0.0065410321374578 0.3690824080946679
12 0.0043042707117221 0.0335513677379153-0.0022367614257357 0.4026337758325830
13 0.0060106451121704 0.0334078920475245 0.0037738836864347 0.4360416678801077
14 0.0079087469427945 0.0332075372104522 0.0116826306292293 0.4692492050905598
15 0.0099910761708919 0.0329439123123590 0.0216737068001212 0.5021931174029188
16 0.0122483446563884 0.0326108764399960 0.0339220514565096 0.5348039938429148
17 0.0146692120341107 0.0322025982847830 0.0485912634906203 0.5670065921276978
18 0.0172400033810439 0.0317136245503127 0.0658312668716642 0.5987202166780104
19 0.0199444086685725 0.0311389577709445 0.0857756755402367 0.6298591744489550
20 0.0227631639997064 0.0304741441486588 0.1085388395399431 0.6603333185976138
21 0.0256737146312911 0.0297153720153352 0.1342125541712341 0.6900486906129490
22 0.0286498597812016 0.0288595815276255 0.1628624139524358 0.7189082721405745
23 0.0316613792205220 0.0279045862015855 0.1945237931729577 0.7468128583421599
24 0.0346736416507075 0.0268492068942347 0.2291974348236652 0.7736620652363947
25 0.0376471948657328 0.0256934188392112 0.2668446296893980 0.7993554840756058
26 0.0405373376992232 0.0244385123436867 0.3073819673886213 0.8237939964192925
27 0.0432936737565710 0.0230872677537126 0.3506756411451923 0.8468812641730052
28 0.0458596469320356 0.0216441452951603 0.3965352880772279 0.8685254094681655
29 0.0481720587108285 0.0201154903974257 0.4447073467880564 0.8886408998655913
30 0.0501605672561820 0.0185097551070648 0.4948679140442383 0.9071506549726561
31 0.0517471682814031 0.0168377361985254 0.5466150823256414 0.9239883911711814
32 0.0528456577069106 0.0151128305891453 0.5994607400325521 0.9391012217603267
33 0.0533610761022577 0.0133513086655816 0.6528218161348097 0.9524525304259083
34 0.0531891349131380 0.0115726061288397 0.7060109510479476 0.9640251365547480
35 0.0522156244733761 0.0097996349650684 0.7582265755213236 0.9738247715198163
36 0.0503158038019030 0.0080591141492892 0.8085423793232266 0.9818838856691055
37 0.0473537721847154 0.0063819206892258 0.8558961515079421 0.9882658063583314
38 0.0431818225418188 0.0048034616164019 0.8990779740497608 0.9930692679747333
39 0.0376397765791564 0.0033640675316746 0.9367177506289172 0.9964333355064079
40 0.0305543017255206 0.0021094083123694 0.9672720523544379 0.9985427438187774
```

```
 41 0.0217382098544504 0.0010909315881854 0.9890102622088882 0.9996336754069628
 42 0.0109897377911118 0.0003663245930371 1.0000000000000000 0.9999999999999998

 ndtfast, nfast =   30  42   nfast/ndtfast = 1.40000

 Centers of gravity and integrals (values must be 1, 1, approx 1/2, 1, 1):

    1.000000000000 1.047601458608 0.523800729304 1.000000000000 1.000000000000

 Power filter parameters, Fgamma, gamma =  0.28400   0.18933

 Minimum X-grid spacing, DXmin =  1.00000000E+00 km
 Maximum X-grid spacing, DXmax =  1.00000000E+00 km
 Minimum Y-grid spacing, DYmin =  1.00000000E+00 km
 Maximum Y-grid spacing, DYmax =  1.00000000E+00 km
 Minimum Z-grid spacing, DZmin =  1.57503054E+00 m
 Maximum Z-grid spacing, DZmax =  2.30290891E+01 m

 Minimum barotropic Courant Number =  2.22358627E-01
 Maximum barotropic Courant Number =  5.42494240E-01
 Maximum Coriolis   Courant Number =  2.47800000E-02


 Maximum grid stiffness ratios:  rx0 =   6.931666E-02 (Beckmann and Haidvogel)
                                 rx1 =   1.188435E+00 (Haney)


 Initial basin volumes: TotVolume =  3.8843755884E+11 m3
                        MinVolume =  1.6168290365E+06 m3
                        MaxVolume =  2.3029089059E+07 m3
                          Max/Min =  1.4243366824E+01

NL ROMS/TOMS: started time-stepping: (Grid: 01 TimeSteps: 00000001 - 00001440)

   STEP   Day HH:MM:SS  KINETIC_ENRG   POTEN_ENRG    TOTAL_ENRG   NET_VOLUME
      0     0 00:00:00  0.000000E+00  6.579497E+02  6.579497E+02 3.884376E+11
      DEF_HIS   - creating history file: ocean_his.nc
      WRT_HIS   - wrote history  fields (Index=1,1) into time record = 0000001
      DEF_AVG   - creating average file: ocean_avg.nc
      DEF_DIAGS - creating diagnostics file: ocean_dia.nc
     72     0 06:00:00  8.366194E-06  6.579497E+02  6.579497E+02 3.884376E+11
      WRT_HIS   - wrote history  fields (Index=1,1) into time record = 0000002
      WRT_AVG   - wrote averaged fields into time record = 0000001
      WRT_DIAGS - wrote diagnostics fields into time record = 0000001
    144     0 12:00:00  7.416156E-05  6.579497E+02  6.579498E+02 3.884376E+11
      WRT_HIS   - wrote history  fields (Index=1,1) into time record = 0000003
      WRT_AVG   - wrote averaged fields into time record = 0000002
      WRT_DIAGS - wrote diagnostics fields into time record = 0000002
    216     0 18:00:00  2.317054E-04  6.579497E+02  6.579499E+02 3.884376E+11
      WRT_HIS   - wrote history  fields (Index=1,1) into time record = 0000004
```

```
      WRT_AVG   - wrote averaged fields into time record = 0000003
      WRT_DIAGS - wrote diagnostics fields into time record = 0000003
    288     1 00:00:00  5.382882E-04  6.579497E+02  6.579503E+02 3.884376E+11
      WRT_HIS   - wrote history  fields (Index=1,1) into time record = 0000005
      WRT_AVG   - wrote averaged fields into time record = 0000004
      WRT_DIAGS - wrote diagnostics fields into time record = 0000004
      WRT_RST   - wrote re-start fields (Index=1,1) into time record = 0000001
            :
   1440     5 00:00:00  2.476731E-02  6.579533E+02  6.579781E+02
3.884376E+11
      WRT_HIS   - wrote history  fields (Index=1,1) into time record
= 0000021
      WRT_AVG   - wrote averaged fields into time record =
0000020
      WRT_DIAGS - wrote diagnostics fields into time record =
0000020
      WRT_RST   - wrote re-start fields (Index=1,1) into time record
= 0000001

 Elapsed CPU time (seconds):

 Thread #  0 CPU:     557.021
 Total:               557.021

 Nonlinear model elapsed time profile:

  Initialization ................................        0.015  ( 0.0027 %)
  Reading of input data .........................        0.002  ( 0.0004 %)
  Processing of input data ......................        0.127  ( 0.0229 %)
  Processing of output time averaged data .......       55.303  ( 9.9284 %)
  Computation of vertical boundary conditions ...        0.419  ( 0.0752 %)
  Computation of global information integrals ....        2.316  ( 0.4159 %)
  Writing of output data ........................        1.923  ( 0.3453 %)
  Model 2D kernel ...............................      365.560  (65.6278 %)
  2D/3D coupling, vertical metrics ..............        1.950  ( 0.3500 %)
  Omega vertical velocity .......................        3.135  ( 0.5628 %)
  Equation of state for seawater ................        1.645  ( 0.2954 %)
  3D equations right-side terms .................       14.268  ( 2.5615 %)
  3D equations predictor step ...................       31.518  ( 5.6583 %)
  Pressure gradient .............................       10.157  ( 1.8235 %)
  Harmonic mixing of tracers, S-surfaces ........        4.130  ( 0.7414 %)
  Harmonic stress tensor, S-surfaces ............        8.211  ( 1.4741 %)
  Corrector time-step for 3D momentum ...........       27.976  ( 5.0225 %)
  Corrector time-step for tracers ...............       19.558  ( 3.5112 %)
                                          Total:      548.216   98.4193

 All percentages are with respect to total time =         557.021

 ROMS/TOMS - Output NetCDF summary for Grid 01:
             number of time records written in HISTORY file = 00000021
```

```
            number of time records written in RESTART file = 00000002
            number of time records written in AVERAGE file = 00000020

 Analytical header files used:

     ROMS/Functionals/ana_btflux.h
     ROMS/Functionals/ana_grid.h
     ROMS/Functionals/ana_initial.h
     ROMS/Functionals/ana_smflux.h
     ROMS/Functionals/ana_stflux.h
     ROMS/Functionals/ana_vmix.h

 ROMS/TOMS: DONE... Friday - October 30, 2009 -  9:20:38 AM
```

Note that it ends by printing out a profile of where the time was used—the ratios will vary depending on the application.

NetCDF history and restart files are also created, containing the model fields at the requested times. We have asked it to save restart records every day. In this case, the restart file has been told to "cycle", or to write over the second last record. The restart file at the end of the run contains the fields at day 4 and day 5. The history file contains records for 0, 1/4, 1/2, 3/4, and 1 day, etc., while the averages and diagnostics files are at 1/8, 3/8, 5/8, and 7/8 day, etc.. Plots can be made from any one of these files, using the plotting software described in §8. Selected frames from such plots are shown in Fig. 17 to 20.

7.3 Northeast Pacific example

The upwelling/downwelling examples is one in which all the start-up fields are defined analytically. The other extreme is one in which everything is read from files, as in our Northeast Pacific simulations. Figure 21 shows the bathymetry and the extent of this domain, which is rectangular in a conic map projection.

7.3.1 nep5.h

The C preprocessor variable **NEP5** has been chosen to label our Northeast Pacific domain, using the fifth generation grid file. The header include file then becomes **nep5.h**:

```
/*
**  Options for Northeast Pacific (NEP5) simulation
*/

#undef NETCDF4
#undef PARALLEL_IO
#undef OFFLINE_FLOATS

/* general */

#define CURVGRID
#define MASKING
#define NONLIN_EOS
#define SOLVE3D
#define SALINITY
#ifdef SOLVE3D
```

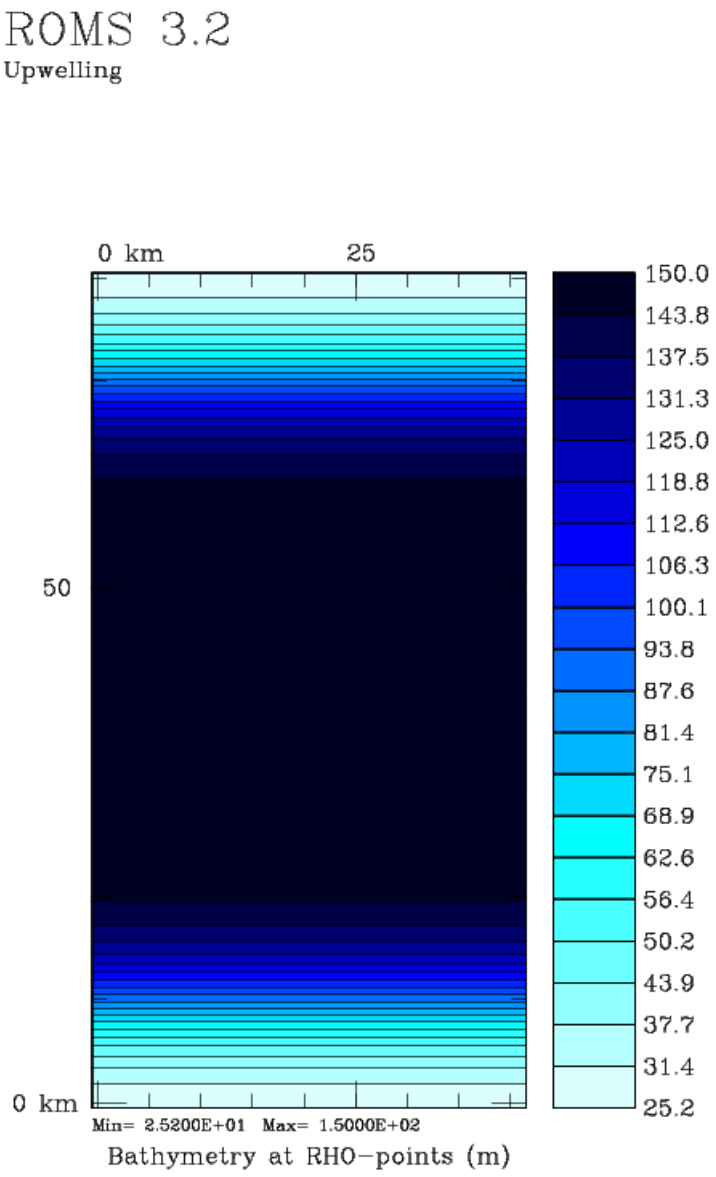

Figure 17: The upwelling/downwelling bathymetry.

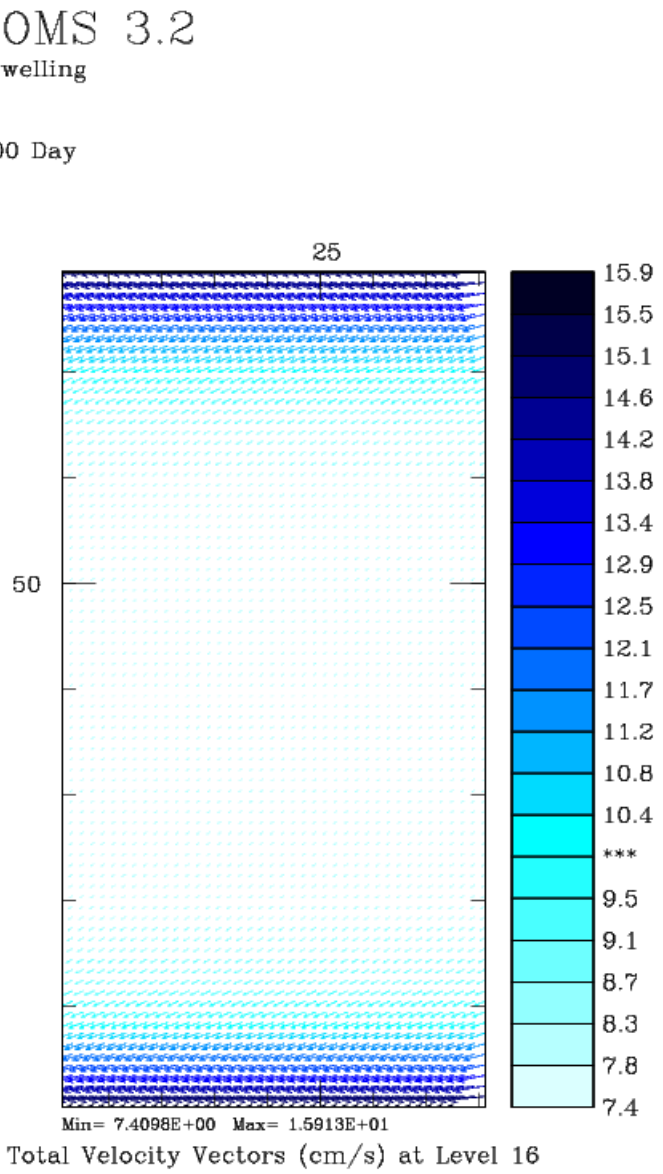

Figure 18: Surface velocities after one day, showing the flow to the left of the wind (southern hemisphere).

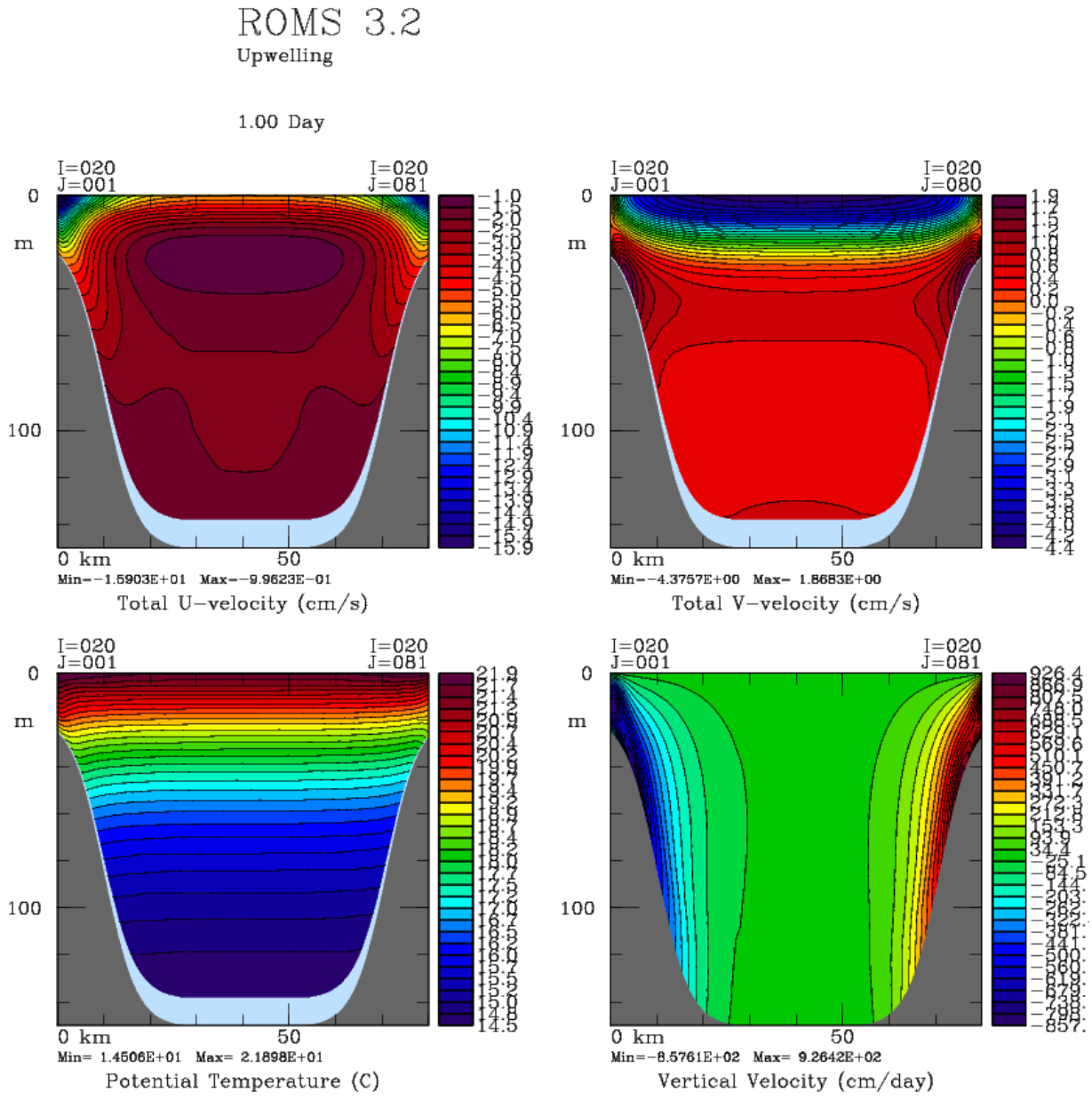

Figure 19: Constant ξ slices of the u, v, T and w fields at day 1.

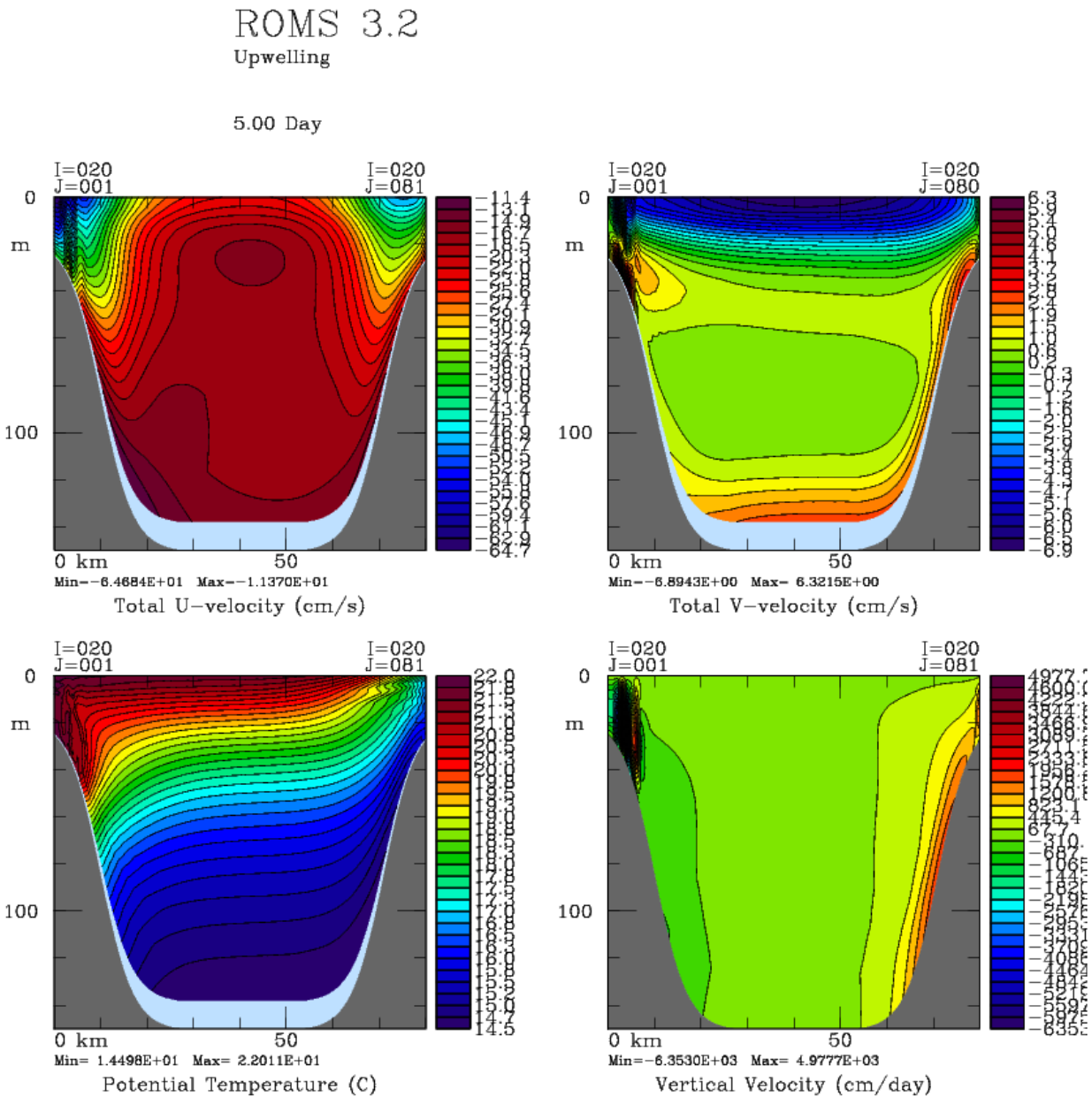

Figure 20: Constant ξ slices of the u, v, T, and w fields at day 5.

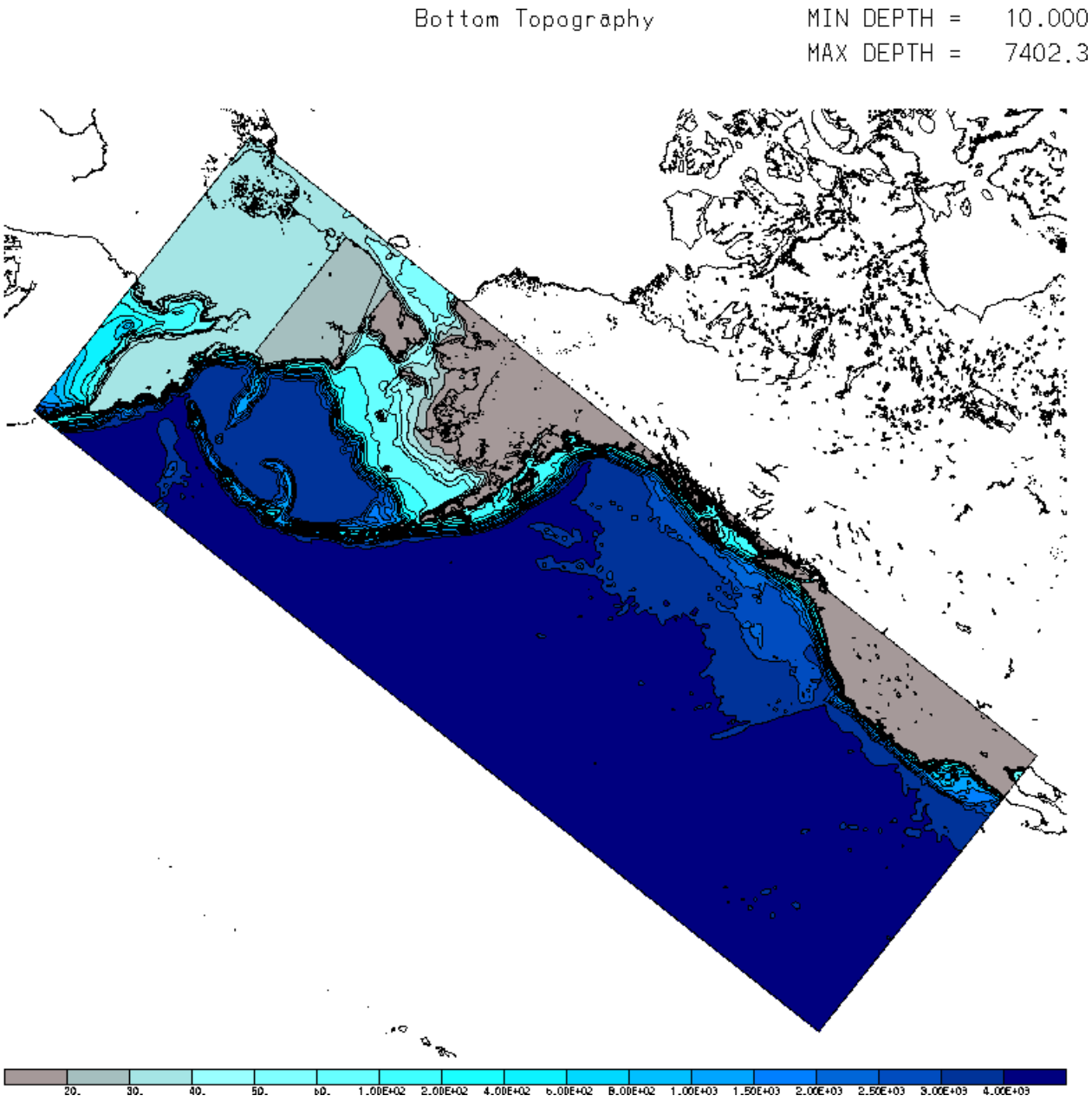

Figure 21: Bathymetry of the Northeast Pacific domain (NEP5).

```
# undef SPLINES
#endif
#define FLOATS
#define STATIONS
#undef WET_DRY

#undef T_PASSIVE
#ifdef T_PASSIVE
# define ANA_PASSIVE
#endif

/* ice */

#ifdef SOLVE3D
# define  ICE_MODEL
# ifdef ICE_MODEL
#  define  ICE_THERMO
#  define  ICE_MK
#  undef   ICE_ALB_EC92
#  undef   ICE_SMOOTH
#  define  ICE_MOMENTUM
#  define  ICE_MOM_BULK
#  define  ICE_EVP
#  define  ICE_ADVECT
#  define  ICE_SMOLAR
#  define  ICE_UPWIND
#  define  ICE_BULK_FLUXES
#  define  ANA_AIOBC
#  define  ANA_HIOBC
#  define  ANA_HSNOBC
# endif
#endif

/* output stuff */

#define NO_WRITE_GRID
#undef OUT_DOUBLE
#define RST_SINGLE
#define AVERAGES
#undef AVERAGES2
#ifdef SOLVE3D
# undef AVERAGES_DETIDE
# define AVERAGES_AKT
# define AVERAGES_AKS
# define AVERAGES_AKV
# define AVERAGES_FLUXES
# undef AVERAGES_QUADRATIC
# undef DIAGNOSTICS_TS
#endif
#undef DIAGNOSTICS_UV
```

```
/* advection, dissipation, pressure grad, etc. */

#ifdef SOLVE3D
# define DJ_GRADPS
#endif

#define UV_ADV
#define UV_COR
#define UV_SADVECTION

#ifdef SOLVE3D
# define TS_U3HADVECTION
# define TS_C4VADVECTION
# undef TS_MPDATA
#endif

#define UV_VIS2
#define UV_SMAGORINSKY
#define VISC_3DCOEF
#define MIX_S_UV
#define VISC_GRID
#define SPONGE

#ifdef SOLVE3D
# define TS_DIF2
# define MIX_GEO_TS
# define DIFF_GRID
#endif


/* vertical mixing */

#ifdef SOLVE3D
# define SOLAR_SOURCE

# define LMD_MIXING
# ifdef LMD_MIXING
#  define LMD_RIMIX
#  define LMD_CONVEC
#  define LMD_SKPP
#  undef LMD_BKPP
#  define LMD_NONLOCAL
#  define LMD_SHAPIRO
#  undef LMD_DDMIX
# endif

# undef GLS_MIXING
# undef MY25_MIXING
```

```
# if defined GLS_MIXING || defined MY25_MIXING
#  define KANTHA_CLAYSON
#  define N2S2_HORAVG
# endif
#endif

/* surface forcing */

#ifdef SOLVE3D
# define CORE_FORCING
# define BULK_FLUXES
# define CCSM_FLUXES
# if defined BULK_FLUXES || defined CCSM_FLUXES
#  define LONGWAVE_OUT
#  define DIURNAL_SRFLUX
#  define EMINUSP
#  undef ANA_SRFLUX
#  undef ALBEDO
#  undef LONGWAVE
# endif
#endif

/* surface and side corrections */

#ifdef SOLVE3D
# define SRELAXATION
# undef QCORRECTION
#endif

/* point sources (rivers, line sources) */

/* Using Runoff instead now */
#ifdef SOLVE3D
#define RUNOFF
# define UV_PSOURCE
# define ANA_PSOURCE
# undef TS_PSOURCE
#endif

/* tides */

#define LTIDES
#ifdef LTIDES
# define FILTERED
# define SSH_TIDES
# define UV_TIDES
# define ADD_FSOBC
# define ADD_M2OBC
# undef RAMP_TIDES
# define TIDES_ASTRO
```

```
# define POT_TIDES

# define UV_LDRAG
# define RDRG_GRID
# define DRAG_LIMITER
# undef UV_QDRAG
#else
# define UV_QDRAG
#endif

/* Boundary conditions...careful with grid orientation */

#define EASTERN_WALL
#define NORTHERN_WALL
#undef WESTERN_WALL
#undef SOUTHERN_WALL

#define RADIATION_2D

#ifndef NORTHERN_WALL
# define NORTH_FSCHAPMAN
# define NORTH_M2FLATHER
# ifdef SOLVE3D
#  define NORTH_M3RADIATION
#  define NORTH_M3NUDGING
#  define NORTH_TRADIATION
#  define NORTH_TNUDGING
#  define NORTH_MIGRADIENT
# endif
#endif

#ifndef WESTERN_WALL
# define WEST_FSCHAPMAN
# define WEST_M2FLATHER
# ifdef SOLVE3D
#  define WEST_M3RADIATION
#  define WEST_M3NUDGING
#  define WEST_TRADIATION
#  define WEST_TNUDGING
#  define WEST_MIGRADIENT
# endif
#endif

#ifndef SOUTHERN_WALL
# define SOUTH_FSCHAPMAN
# define SOUTH_M2FLATHER
# ifdef SOLVE3D
#  define SOUTH_M3RADIATION
#  define SOUTH_M3NUDGING
#  define SOUTH_TRADIATION
```

```
#  define SOUTH_TNUDGING
#  define SOUTH_MIGRADIENT
# endif
#endif

#ifndef EASTERN_WALL
# define EAST_FSCHAPMAN
# define EAST_M2FLATHER
# ifdef SOLVE3D
#  define EAST_M3RADIATION
#  define EAST_M3NUDGING
#  define EAST_TRADIATION
#  define EAST_TNUDGING
#  define EAST_MIGRADIENT
# endif
#endif

/* roms quirks */

#ifdef SOLVE3D
# define ANA_BSFLUX
# define ANA_BTFLUX
#else
# define ANA_SMFLUX
#endif

/*
**  Biological model options.
*/
#define NEMURO
#define LIMIT_BIO_AKT
#undef BIO_GOANPZ        /* Sarah Hinckley's 11 box model */
#undef BEST_NPZ         /* Georgina Gibsons BEST NPZ model  */

#if defined BEST_NPZ || defined BIO_GOANPZ || defined PASSIVE_TRACERS
# undef  BIOFLUX           /* sum Nitrogen fluxes between boxes */
# define ANA_BIOLOGY       /* analytical biology initial conditions */
# define ANA_BPFLUX        /* analytical bottom passive tracers fluxes */
# define ANA_SPFLUX        /* analytical surface passive tracers fluxes */
# define DIAPAUSE          /* Enable Neocalanus seasonal vertical migration */
# undef FLOAT_VWALK
# define IRON_LIMIT        /* Add iron as passive 11th tracer */
# undef TCLM_NUDGING      /* Nudging of tracer climatology for iron */
#endif

#if defined NEMURO
# undef ANA_BIOLOGY       /* analytical biology initial conditions */
# define ANA_BPFLUX        /* analytical bottom passive tracers fluxes */
# define ANA_SPFLUX        /* analytical surface passive tracers fluxes */
# define IRON_LIMIT        /* Add iron as passive 11th tracer */
```

```
# define IRON_RELAX
# undef  IRON_RSIN
# define BIO_SEDIMENT
# define HOLLING_GRAZING
# undef  IVLEV_EXPLICIT
# undef  ANA_BIOSWRAD
# undef  DIAGNOSTICS_BIO
# undef  BIO_SEDIMENT
#endif
```

Here, we have declared that we want two open sides and two closed walls. For each open side, we're using a Chapman/Flather combination on the 2D flow and a radiation/nudging combination on the 3D flow. We've got masking, salinity, and the non-linear equation of state. We want Laplacian viscosity on σ-surfaces, diffusion along constant z-surfaces and the full non-linear, curvilinear momentum equations. Sea ice is turned on, as is the **NEMURO** ecosystem (for some runs at least).

Notice that comments are allowed in this file as long as they are in the C/C++ syntax. Also, conditionals are fine—some parts are only wanted when **SOLVE3D** is on. There is evidence of options having been tried and rejected, such as the quadratic bottom drag in the presence of tides. The "Eastern" side is actually open to the North, up in the Arctic, but it is set to be a closed boundary. Instead, there is an imposed flow through Bering Strait by using **UV_PSOURCE** and **ANA_PSOURCE**.

7.3.2 NEP5 code chunks

As mentioned above, we are specifying point sources in **ana_psource.h**. The **NEP5** section of the copy in **Apps/NEP** is:

```
#if defined NEP5
        Nsrc=149
        DO is=1,Nsrc
          Dsrc(is)=0.0_r8
          Isrc(is)=225
          Jsrc(is)=472+is
          Lsrc(is,itemp)=.FALSE.
          Lsrc(is,isalt)=.FALSE.
        END DO
#elif defined BERING_10K
            :

#if defined UV_PSOURCE || defined Q_PSOURCE
# ifdef SOLVE3D
!
!  If appropriate, set-up nondimensional shape function to
distribute
!  mass point sources/sinks vertically.  It must add to unity!!.
!
#  if defined NEP5
        DO k=1,N(ng)
          DO is=1,Nsrc
            Qshape(is,k)=1.0_r8/REAL(N(ng),r8)
          END DO
```

```
        END DO
#  endif
# endif
# if defined NEP5
      cff = 800000._r8/Nsrc
      DO is=1,Nsrc
        Qbar(is)=cff
      END DO
# else
      ana_psource.h: No values provided for Qbar.
# endif
```

The total transport is set to 0.8 Sv and is divided among the **Nsrc** sources.

Doing a search for **NEP5** shows up a few other instances. One is in **output.F**, forcing more digits in the averages file names for handling one file per record, one record per day for fifty years:

```
#ifdef NEP5
  20          FORMAT (a,'_',i5.5,'.nc')
#else
  20          FORMAT (a,'_',i4.4,'.nc')
#endif
```

Also in **output.F** is code to ask for a new stations file for each run rather than appending to the old one:

```
#ifdef NEP5
          CALL def_station (ng, .true.)
#else
          CALL def_station (ng, ldefout(ng))
#endif
```

A better fix (someday) would be to allow the station files to be numbered just like the averages files. Why not set **ldefout** to be true? It is shared by many outputs, including the floats, and you can only restart floats with **ldefout** set to false.

One feature which got turned on after the two decade "run 35" is **SRELAXATION**, in which I'm nudging the sea surface salinity to a value from a forcing file. If you check the nudging code, you will find that the timescale for nudging is the same as that used for the open boundary conditions. That isn't quite what I want, so there's this change to **set_vbc.F**:

```
    #   elif defined SRELAXATION
              stflx(i,j,isalt)=-Tnudg(isalt,ng)*Hz(i,j,N(ng))*      &
    #ifdef NEP5
    !  60 days from Tnudg of 360 days
         &                     6.0_r8*                               &
    #endif
         &                     (t(i,j,N(ng),nrhs,isalt)-sss(i,j))
```

There is also a kludge in **step_floats.F** for a yet-untested float reuse strategy. Having a handy tag like **NEP5** allows me to search for these little hacks later. The file **Apps/NEP/ana_hmixcoef.h** tried to sneak by without any **NEP5**, but the location means that it will be used to set up the sponge layers for this domain:

```
    !
```

```
! Northeast Pacific sponge areas
!
      Iwrk = 10
#  if defined UV_VIS2
      DO i=IstrR,IendR
        DO j=JstrR,MIN(Iwrk,JendR)
          cff = 250.*0.5_r8*(1.0_r8+COS(pi*REAL(j,r8)/REAL(Iwrk,r8)))
          visc2_r(i,j) = max(cff, visc2_r(i,j))
          visc2_p(i,j) = max(cff, visc2_p(i,j))
        END DO
      END DO
      DO i=IstrR,MIN(Iwrk,IendR)
        DO j=MAX(JstrR,i),JendR
          cff = 250.*0.5_r8*(1.0_r8+COS(pi*REAL(i,r8)/REAL(Iwrk,r8)))
          visc2_r(i,j) = max(cff, visc2_r(i,j))
          visc2_p(i,j) = max(cff, visc2_p(i,j))
        END DO
      END DO
        :
#   if defined TS_DIF2
      DO itrc=1,NT(ng)
        DO j=JstrR,MIN(Iwrk,JendR)
          cff = 100. * (1.0_r8+COS(pi*REAL(j,r8)/REAL(Iwrk,r8)))
          DO i=IstrR,IendR
            diff2(i,j,itrc)=max(cff, diff2(i,j,itrc))
          END DO
        END DO
        DO i=IstrR,MIN(Iwrk,IendR)
          DO j=MAX(JstrR,i),JendR
            cff = 100. * (1.0_r8+COS(pi*REAL(i,r8)/REAL(Iwrk,r8)))
            diff2(i,j,itrc) = max(cff, diff2(i,j,itrc))
          END DO
        END DO
```

7.3.3 Model domain

A number of horizontal grid points was chosen to resolve the domain at roughly 10 km. Values for **Lm**, **Mm**, and **N** are:

```
        Lm == 224           ! Number of I-direction INTERIOR RHO-points
        Mm == 640           ! Number of J-direction INTERIOR RHO-points
         N == 60            ! Number of vertical levels
```

The vertical structure is the same as we've been using for a good long time now, though the minimum depth keeps creeping shallower:

```
  Vtransform == 1                       ! transformation equation
 Vstretching == 1                       ! stretching function

    THETA_S == 5.0d0                    ! surface stretching parameter
    THETA_B == 0.4d0                    ! bottom  stretching parameter
     TCLINE == 10.0d0                   ! critical depth (m)
```

7.3.4 Initial and boundary conditions

The best fields we know of at the moment are known as SODA, a reanalysis from 1958 through 2005 by Carton et al. ([6]). There are **Matlab** scripts to create initial and boundary conditions from the SODA files.

7.3.5 Forcing

We are using the CORE forcing files ([40]) and computing the momentum, heat and salt fluxes from the atmospheric conditions and the model's surface temperature. There are two options, one provided by **bulk_flux.F** and the other provided by **ccsm_flux.F**. We are opting to use the second. With minimal fussing, the CORE files can be used as is, on their native grid, then interpolated by ROMS internally to the domain at hand.

7.3.6 ocean.in

We use an internal time-step of 200 s and an external time-step of 10 s. The horizontal viscosity and diffusion is:

```
 TNU2 == 5.0d0  5.0d0                    ! m2/s
VISC2 == 25.0d0                          ! m2/s
```

plus the sponge layers as shown above along the SE and SW boundaries.

7.3.7 Output

The model writes voluminous information to standard out, as shown for the **UPWELLING** case. It also writes out NetCDF files for restart, history, daily averages, stations and floats. Plots can be made from any of these files; some examples are shown in Fig. 22–24 .

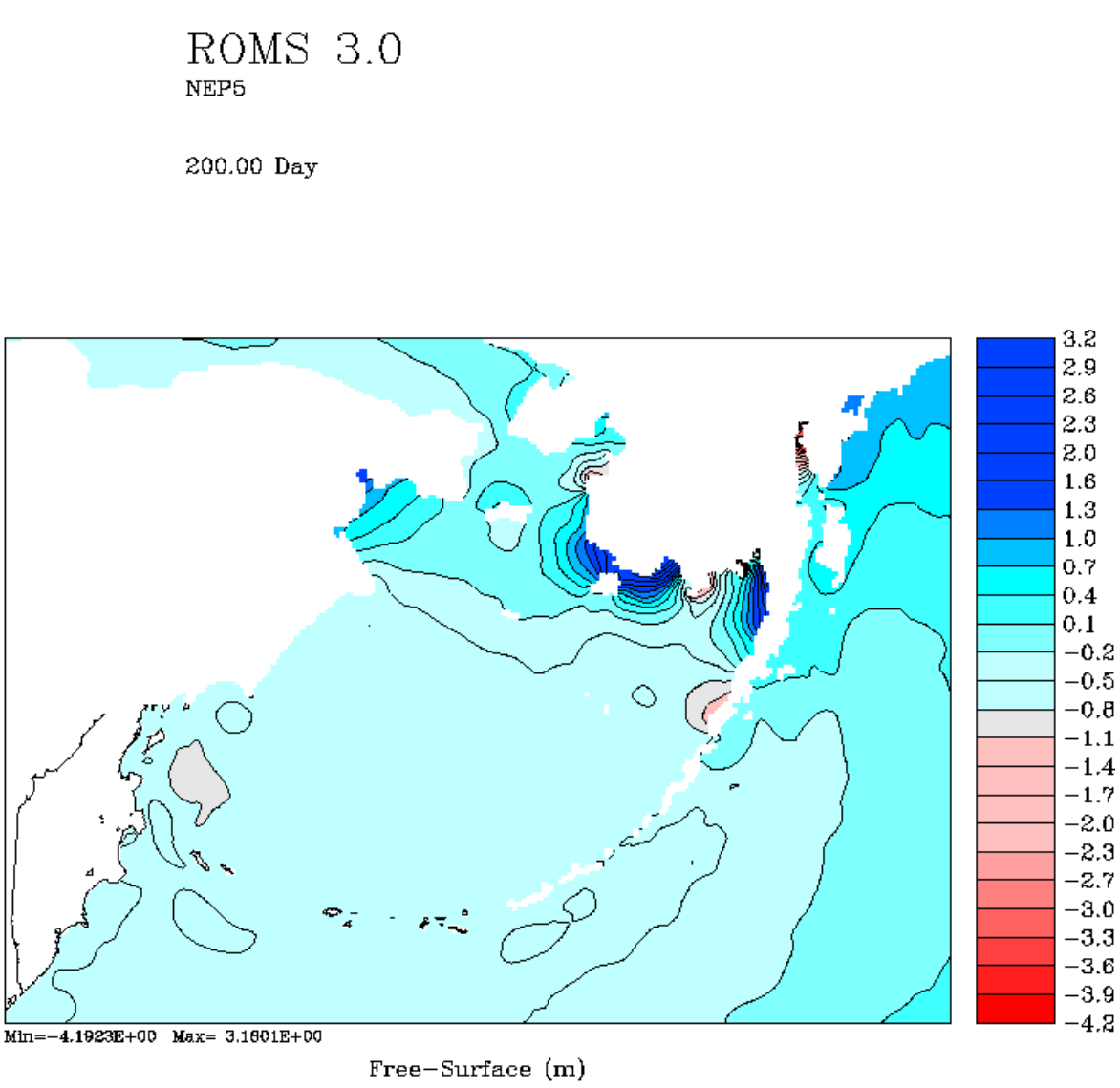

Figure 22: Surface elevation after 200 days showing tides. This is from a snapshot in a history file—the averages files have been detided.

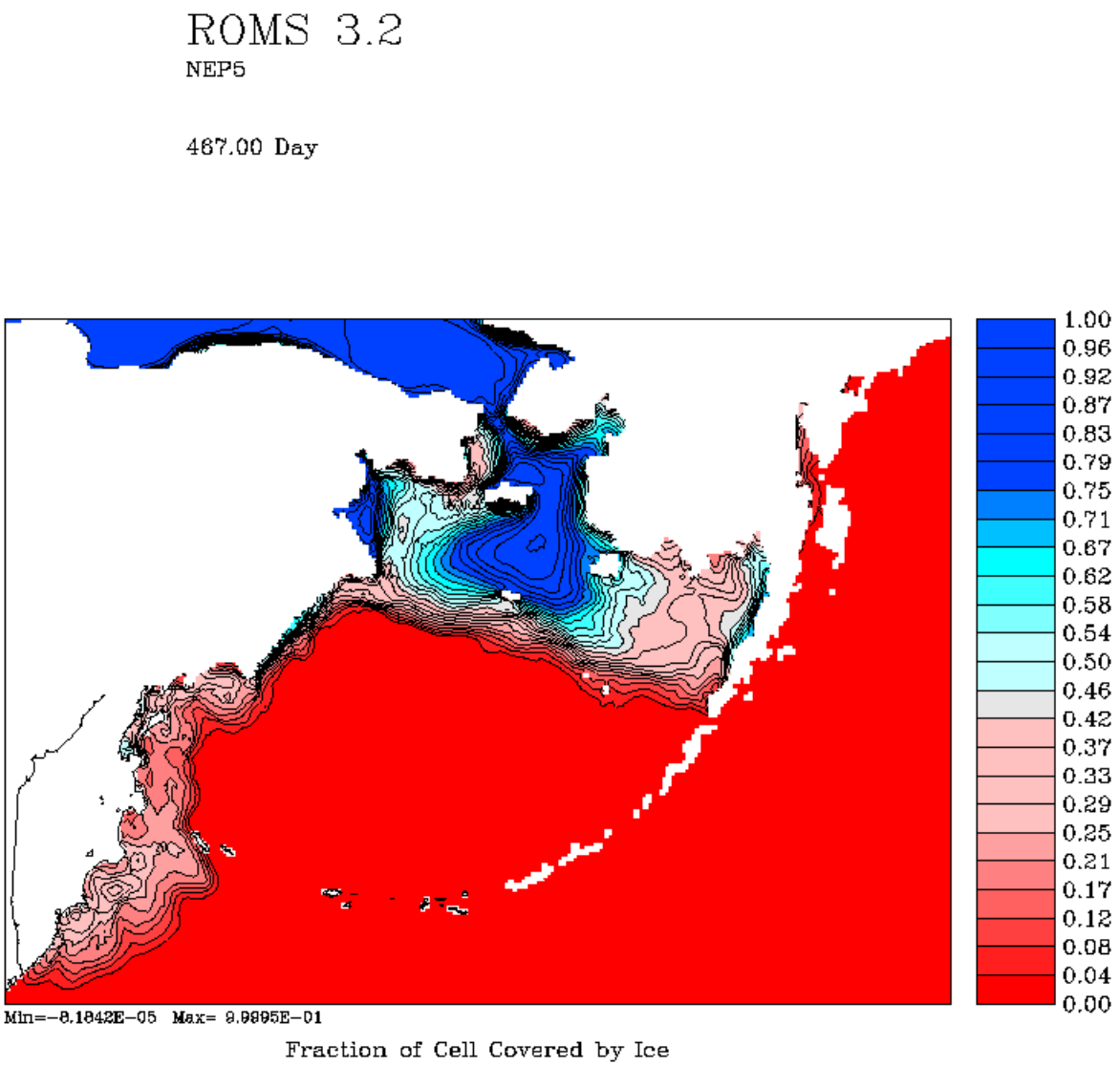

Figure 23: Ice concentration averaged over the month of April, 1959.

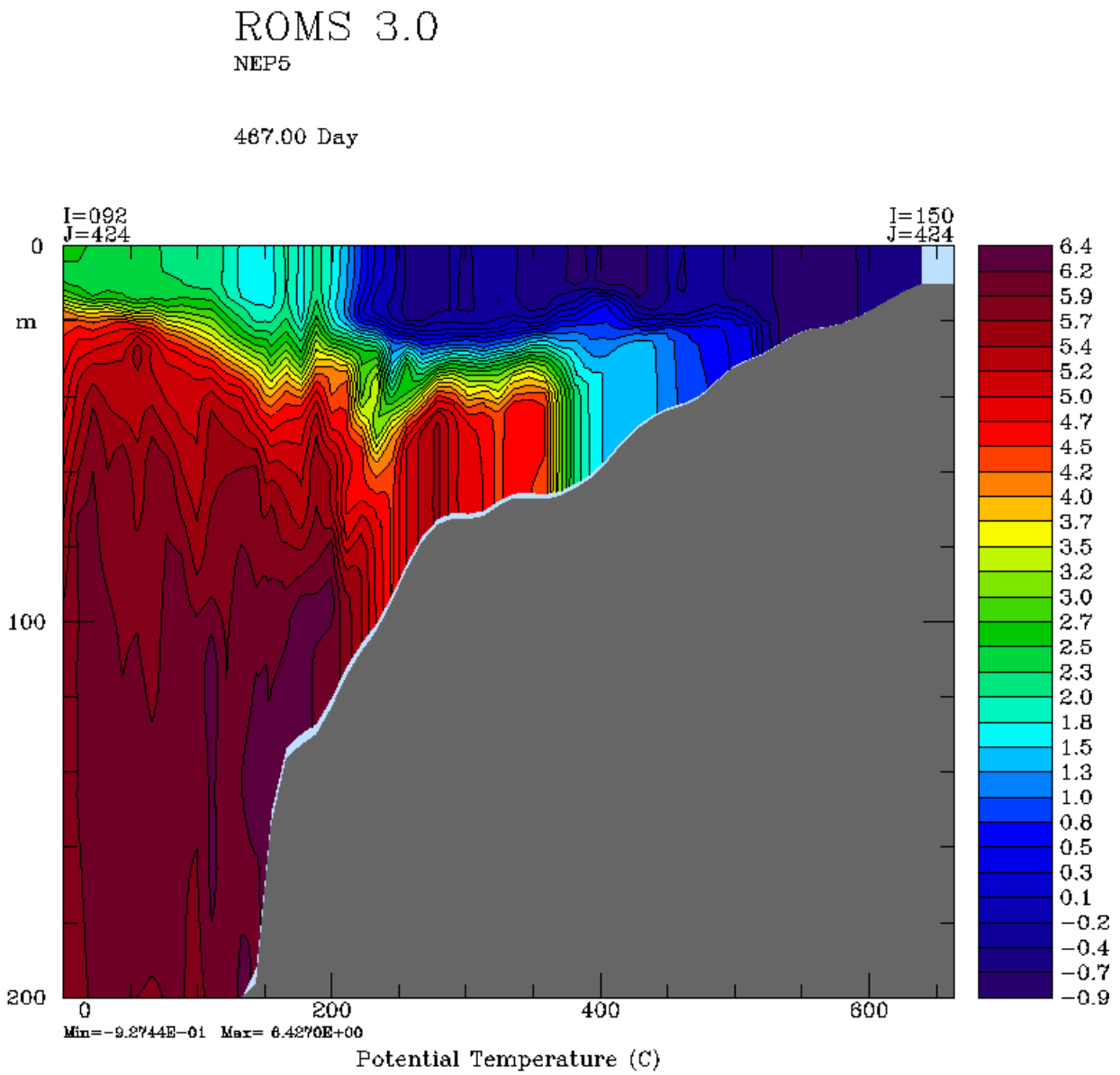

Figure 24: Vertical slice if temperature, averaged over the month of April, 1959. The slice is across the Bering Sea shelf, showing the transition from vertically mixed at the coast, a two layer system at mid-shelf, then a thermocline over the shelf-break.

8 Plotting Programs for Postprocessing

Hernan Arango has provided ROMS with some **Fortran** programs for creating plots from the NetCDF history and restart files. Some example plots are shown in §7. Other plotting options are available via **Matlab**, **Python** and **NCL**. Here we describe only the **Fortran** programs available via **svn** from **myroms.org**.

There are four plotting programs:

cnt creates black-and-white plots of the horizontal fields, including constant depth plots of the 3-D fields.

ccnt creates color plots of the horizontal fields, including constant depth plots of the 3-D fields.

sec creates black-and-white plots of vertical slices through the 3-D fields. It includes on option of finding equal-spaced points along a straight line through the curvilinear grid.

csec creates color plots of vertical slices through the 3-D fields. It includes on option of finding equal-spaced points along a straight line through the curvilinear grid.

All of these program come with example input files. For instance, the input file for **ccnt** is called **ccnt.in** and is as follows:

```
1996 -1  :  year and starting year-day (use yearday<0, for no time label)
ROMS 3.2
Coarse Arctic ocean with Budgell ice dynamics
ice thermodynamics

8     NFIELDS: number of fields to plot. Line below, field(s) types:
42,45,46,47,48,49,50,121,122,123,124  field identification:
FLDID(1:NFIELDS)
1     NLEVELS: number of levels and/or depths to plot (0 for all levels)
20           levels (>0) or depths (<0) to plot: FLDLEV(1:NLEVELS)
2     NFIELDS: number of fields to plot. Line below, field(s) types:
1,2            field identification: FLDID(1:NFIELDS)
1     NLEVELS: number of levels and/or depths to plot (0 for all levels)
1,2,3,4,5      levels (>0) or depths (<0) to plot: FLDLEV(1:NLEVELS)
0     FRSTD  : first day to plot
0     LASTD  : last day to plot
0     DSKIP  : plot every other DSKIP days (0.0 plot at its own time frequency)
0     VINTRP : vertical interpolation scheme: 0=linear, 1:cubic splines
0.0   PMIN   : field minimum value for color palette (0.0 for default)
0.0   PMAX   : field maximum value for color palette (0.0 for default)
1     ICNT   : draw contours between color bands: 0=no, 1=yes
0.0   ISOVAL : iso-surface value to process (see below)
1.2   VLWD   : vector line width (1.0 for default)
2.0   VLSCL  : vector length scale (1.0 for default)
1     IVINC  : vector grid sampling in the X-direction (1 for default)
1     JVINC  : vector grid sampling in the Y-direction (1 for default)
0     IREF   : secondary or reference field option (see below)
25    IDOVER : overlay field identification (for IREF=1,2 only)
1     LEVOVER: level of the overlay field (set to 0 if same as current FLDLEV)
0.0   RMIN   : overlay field minimum value to consider (0.0 for default)
0.0   RMAX   : overlay field maximum value to consider (0.0 for default)
```

```
10.0   LGRID : Desired longitude/latitude grid spacing (degrees)
4     IPROJ  : map projection (see below).
-60.0   PLON : projection Pole longitude (west values are negative).
90.0   PLAT  : projection Pole latitude (south values are negative).
0.0   ROTA   : projection rotation angle (clockwise; degrees).
0     LMSK   : flag to color mask land: [0] no, [1] yes
-1     NPAGE : number of plots per page (currently 1, 2, or 4)
T     READGRD: logical switch to read in positions from grid NetCDF file.
F     PLTLOGO: logical switch draw Logo.
T     WRTHDR : logical switch to write out the plot header titles.
T     WRTBLAB: logical switch to write out the plot bottom title.
T     WRTRANG: logical switch to write out data range values and CI.

T     WRTFNAM: logical switch to write out input primary filename.
T     WRTDATE: logical switch to write out current date.
T     CST    : logical switch to read and plot coastlines and islands.
50.0 50.0    : bottom and top map latitudes (south values are negative).
-110.0 80.0  : left and right map longitudes (west values are negative).
/d2/kate/plot/varid.dat
/d1/arango/scrum3.0/plot/Palettes/gs1.pal
/d2/kate/plot/default.cnt
ice_rst.nc
scrum_rst_1.nc
/d2/kate/arctic/gridpak/arctic_grid_2.nc
/u1/coasts/coast.dat


c
c======================================================================
c  Copyright (c) 1996 Rutgers University                             ===
c======================================================================
c

 *** Above FILENAMES:

            1st line: input; variables ID file.
            2nd line: input; color palette file.
            3rd line: input; contour parameters.
            4th line: input; primary NetCDF file.
            5th line: input; secondary NetCDF file.
            6th line: input; grid NetCDF file.
            7th line: input; coastlines file.

 *** IREF:  Secondary or reference field option:
           -1   Overlay horizontal grid
            0   no secondary or reference field to plot
            1   plot field overlay from primary file
            2   plot field overlay from secondary file
            3   primary - secondary file (field subtraction)
            4   Day0 - DayN (field subtraction)
```

```
*** IPROJ: Map Projections option.  Some of the values for the
           projection Pole and rotation angle are suggested.
           Check the NCAR manual for details.

          [1] Cylindrical equidistant: PLON=0, PLAT=0, ROTA=0
          [2] Mercator: PLON=?, PLAT=0, ROTA=0
          [3] Lambert conformal conic: PLON=?, PLAT=?, ROTA=?
          [4] Stereographic azimuthal:  PLON=?, PLAT=90 or -90, ROTA=0

*** IVINC, JVINC: vector grid sampling.  If either value is negative,
                  the velocity vectors at drawn at PSI-points.  Otherwise,
                  if both values are positive, the vectors are drawn at
                  interior RHO-points.

*** LGRID: Longitude/latitude grid spacing.  The grid is drawn at
           LGRID spacing starting from specified map lower corner.
           If LGRID is negative, the latitude labels in the map are
           rotated 90 degrees to avoid label congestion, if any.

*** NPAGE: Number of plots per page.  Set this parameter to a negative
           value (-1, -2, or -4) to activate preservation of the plot
           aspect ratio.


 Plotting Fields classification: (* derived fields)

 [  1]  IDUTOT  total velocity component in the XI-direction (cm/s).
 [  2]  IDVTOT  total velocity component in the ETA-direction (cm/s).
*[  3]  IDTVEC  total velocity vectors (cm/s).
*[  4]  IDTMAG  total velocity vector magnitude (cm/s).
                :
                :
```

As you can see, there are comments describing what needs to be done. Please see the variable ID file for the complete list of fields which can be plotted—this list changes as Hernan adds the ability to plot new fields. Also, check your **default.cnt** file for other vector and contour parameters. The palette file includes two number systems, one in the scale from 0 to 255 and the other from 0 to 1. The ROMS plotting uses the first while the SEOM plotting uses the second.

A Model Time-stepping Schemes

Numerical time stepping uses a discrete approximation to:

$$\frac{\partial \phi(t)}{\partial t} = \mathcal{F}(t) \tag{227}$$

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 ([72]). 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 4.3, 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:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \mathcal{F}(t) \tag{228}$$

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 $\mathrm{O}(\Delta t^2)$:

$$\frac{\phi(t+\Delta t) - \phi(t-\Delta t)}{2\Delta t} = \mathcal{F}(t). \tag{229}$$

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 $\mathcal{F}^*(t+\Delta t)$:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \frac{1}{2}\left[\mathcal{F}(t) + \mathcal{F}^*(t+\Delta t)\right]. \tag{230}$$

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:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \alpha\mathcal{F}(t) + \beta\mathcal{F}(t-\Delta t) + \gamma\mathcal{F}(t-2\Delta t) \tag{231}$$

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:

$$\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} = \phi' + \frac{\Delta t}{2}\phi'' + \frac{\Delta t^2}{6}\phi''' + \cdots \tag{232}$$

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 227 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:

$$\begin{aligned}\frac{\partial \zeta}{\partial t} &= \mathcal{F}(u) \\ \frac{\partial u}{\partial t} &= \mathcal{G}(\zeta)\end{aligned} \tag{233}$$

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

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \mathcal{F}(u^n)\Delta t \\ u^{n+1} &= u^n + \mathcal{G}(\zeta^{n+1})\Delta t\end{aligned} \tag{234}$$

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 233 is a predictor-corrector with predictor step:

$$\begin{aligned}\zeta^{n+1,\star} &= \zeta^n + \mathcal{F}(u^n)\Delta t \\ u^{n+1,\star} &= u^n + \left[\beta\mathcal{G}(\zeta^{n+1,\star}) + (1-\beta)\mathcal{G}(\zeta^n)\right]\Delta t\end{aligned} \tag{235}$$

and corrector step:

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \frac{1}{2}\left[\mathcal{F}(u^{n+1,\star}) + \mathcal{F}(u^n)\right]\Delta t \\ u^{n+1} &= u^n + \frac{1}{2}\left[\epsilon\mathcal{G}(\zeta^{n+1}) + (1-\epsilon)\mathcal{G}(\zeta^{n+1,\star}) + \mathcal{G}(\zeta^n)\right]\Delta t\end{aligned} \tag{236}$$

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 $\alpha_{\max} = 2.14093$ ([72]).

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

Another possibility is a leapfrog predictor:

$$\begin{aligned}\zeta^{n+1,\star} &= \zeta^{n-1} + 2\mathcal{F}(u^n)\Delta t \\ u^{n+1,\star} &= u^{n-1} + 2\left\{(1-2\beta)\mathcal{G}(\zeta^n) + \beta\left[\mathcal{G}(\zeta^{n+1,\star}) + \mathcal{G}(\zeta^{n-1})\right]\right\}\Delta t\end{aligned} \tag{237}$$

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

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \left[\left(\frac{1}{2}-\gamma\right)\mathcal{F}(u^{n+1,\star}) + \left(\frac{1}{2}+2\gamma\right)\mathcal{F}(u^n) - \gamma\mathcal{F}(u^{n-1})\right]\Delta t \\ u^{n+1} &= u^n + \left\{\left(\frac{1}{2}-\gamma\right)\left[\epsilon\mathcal{G}(\zeta^{n+1}) + (1-\epsilon)\mathcal{G}(\zeta^{n+1,\star})\right] + \left(\frac{1}{2}+2\gamma\right)\mathcal{G}(\zeta^n) - \gamma\mathcal{G}(\zeta^{n-1})\right\}\Delta t\end{aligned} \tag{238}$$

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 $\alpha_{\max} = 1.851640$ ([72]).

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:

$$\begin{aligned}\zeta^{n+1} &= \zeta^n + \left[\left(\frac{3}{2}+\beta\right)\mathcal{F}(u^n) - \left(\frac{1}{2}+2\beta\right)\mathcal{F}(u^{n-1}) + \beta\mathcal{F}(u^{n-2})\right]\Delta t \\ u^{n+1} &= u^n + \left[\left(\frac{1}{2}+\gamma+2\epsilon\right)\mathcal{G}(\zeta^{n+1}) + \left(\frac{1}{2}-2\gamma-3\epsilon\right)\mathcal{G}(\zeta^n) + \gamma\mathcal{G}(\zeta^{n-1}) + \epsilon\mathcal{G}(\zeta^{n-2})\right]\Delta t\end{aligned} \tag{239}$$

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 ([72]) 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 [76] but modified by Shchepetkin and McWilliams [70], the vertical coordinate has been chosen to be:

$$z = \zeta(1+\sigma) + h_c\sigma + (h-h_c)C(\sigma), \qquad -1 \le \sigma \le 0 \tag{240}$$

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:

$$C(\sigma) = (1-b)\frac{\sinh(\theta\sigma)}{\sinh\theta} + b\frac{\tanh[\theta(\sigma+\frac{1}{2})] - \tanh(\frac{1}{2}\theta)}{2\tanh(\frac{1}{2}\theta)} \tag{241}$$

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 (240) 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:
$$z = (\zeta + h)(1+\sigma) - h \tag{242}$$
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 25 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 §2.1).

We find it convenient to define:

$$H_z \equiv \frac{\partial z}{\partial \sigma} = (\zeta + h) + (h - h_c)\frac{\partial C(\sigma)}{\partial \sigma}. \tag{243}$$

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

$$\frac{\partial C(\sigma)}{\partial \sigma} = (1-b)\frac{\cosh(\theta\sigma)}{\sinh\theta}\theta + b\frac{\coth(\frac{1}{2}\theta)}{2\cosh^2[\theta(\sigma+\frac{1}{2})]}\theta. \tag{244}$$

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:

$$z(k) = z^{(0)}(k) + \zeta\left(1 + \frac{z^{(0)}(k)}{h}\right) \tag{245}$$

or

$$\Delta z(k) = \Delta z^{(0)}(k)\left(1 + \frac{\zeta}{h}\right) \tag{246}$$

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

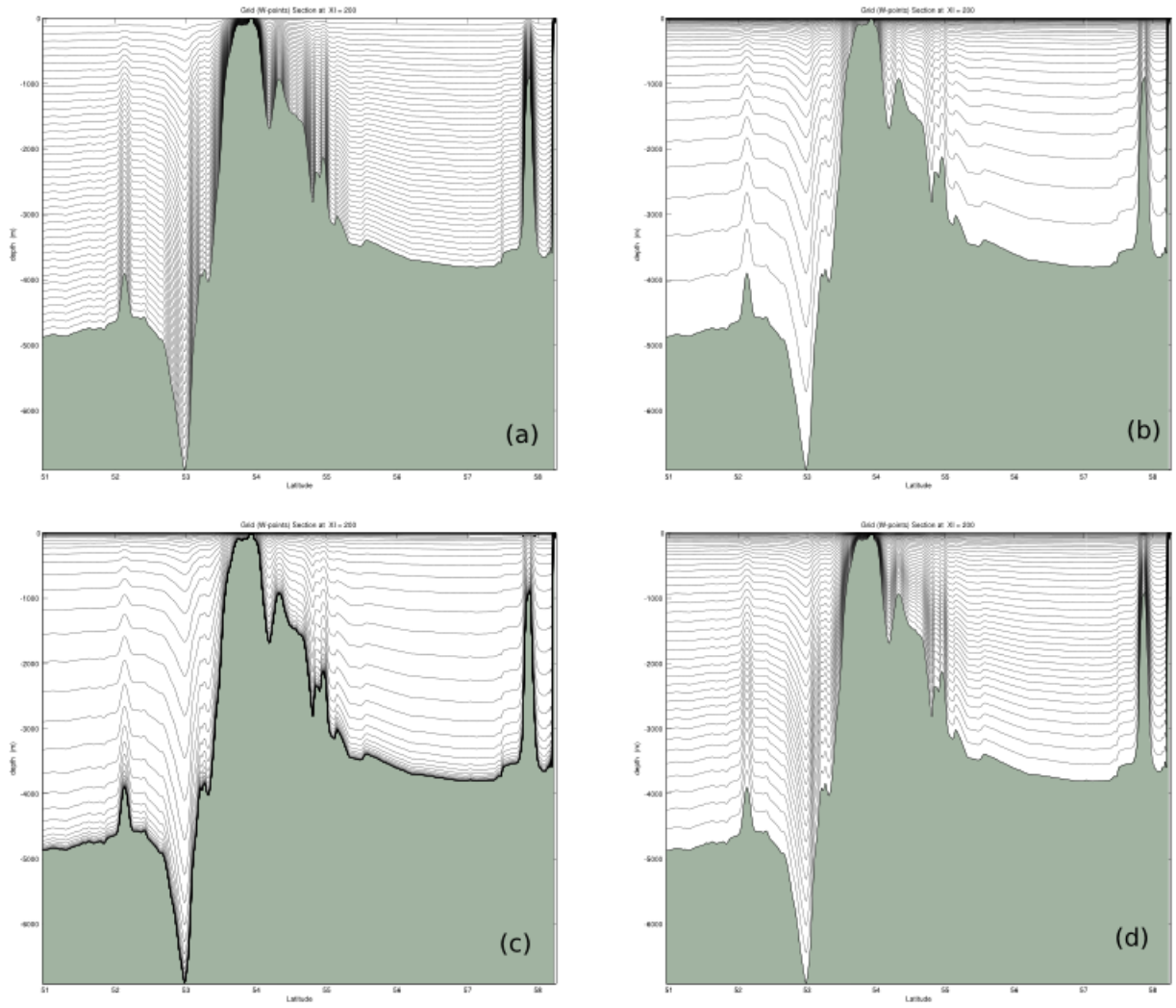

Figure 25: The σ-surfaces for the North Atlantic with (a) $\theta = 0.0001$ and $b = 0$, (b) $\theta = 8$ and $b = 0$, (c) $\theta = 8$ and $b = 1$. (d) The actual values used in this domain were $\theta = 5$ and $b = 0.4$.

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:

$$(ds)_\xi = \left(\frac{1}{m}\right) d\xi \tag{247}$$

$$(ds)_\eta = \left(\frac{1}{n}\right) d\eta \tag{248}$$

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]):

$$\nabla\phi = \hat{\xi} m \frac{\partial\phi}{\partial\xi} + \hat{\eta} n \frac{\partial\phi}{\partial\eta} \tag{249}$$

$$\nabla \cdot \vec{a} = mn \left[\frac{\partial}{\partial\xi}\left(\frac{a}{n}\right) + \frac{\partial}{\partial\eta}\left(\frac{b}{m}\right) \right] \tag{250}$$

$$\nabla \times \vec{a} = mn \begin{vmatrix} \frac{\hat{\xi}_1}{m} & \frac{\hat{\xi}_2}{n} & \hat{k} \\ \frac{\partial}{\partial\xi} & \frac{\partial}{\partial\eta} & \frac{\partial}{\partial z} \\ \frac{a}{m} & \frac{b}{n} & c \end{vmatrix} \tag{251}$$

$$\nabla^2\phi = \nabla \cdot \nabla\phi = mn \left[\frac{\partial}{\partial\xi}\left(\frac{m}{n}\frac{\partial\phi}{\partial\xi}\right) + \frac{\partial}{\partial\eta}\left(\frac{n}{m}\frac{\partial\phi}{\partial\eta}\right) \right] \tag{252}$$

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

$$\Delta t < t_{damp} = \frac{\Delta x^2}{\pi^2\nu_2} \quad \text{or} \quad \frac{\Delta x^4}{\pi^4\nu_4} \tag{253}$$

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

$$\Delta x < L_{BL} = \left(\frac{\nu_2}{\beta}\right)^{\frac{1}{3}} \quad \text{and} \quad \left(\frac{\nu_4}{\beta}\right)^{\frac{1}{5}} \tag{254}$$

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 §4.11.

E Radiant heat fluxes

As was seen in §5.2, the model thermodynamics requires fluxes of latent and sensible heat and longwave and shortwave radiation. We follow the lead of Parkinson and Washington [57] in computing these terms.

E.1 Shortwave radiation

The Zillman equation for radiation under cloudless skies is:

$$Q_o = \frac{S\cos^2 Z}{(\cos Z + 2.7)e \times 10^{-5} + 1.085\cos Z + 0.10} \tag{255}$$

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

$$\cos Z = \sin\phi \sin\delta + \cos\phi\cos\delta\cos HA. \tag{256}$$

The declination is

$$\delta = 23.44^\circ \times \cos\left[(172 - \text{day of year}) \times 2\pi/365\right] \tag{257}$$

and the hour angle is

$$HA = (12\,\text{hours} - \text{solar time}) \times \pi/12. \tag{258}$$

The correction for cloudiness is given by

$$SW\downarrow = Q_o(1 - 0.6c^3). \tag{259}$$

An estimate of the cloud fraction c will be provided by Jennifer Francis ([15]).

E.2 Longwave radiation

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

$$F\downarrow = \sigma T_a^4 \left\{1 - 0.261\exp\left[-7.77 \times 10^{-4}(273 - T_a)^2\right]\right\} \tag{260}$$

while the cloud correction is given by:

$$LW\downarrow = (1 + 0.275c)\,F\downarrow\,. \tag{261}$$

E.3 Sensible heat

The sensible heat is given by the standard aerodynamic formula:

$$H\downarrow = \rho_a c_p C_H V_{wg}(T_a - T_{sfc}). \tag{262}$$

E.4 Latent heat

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

$$e = 611 \times 10^{a(T_d - 273.16)/(T_d - b)} \tag{263}$$

$$e_s = 611 \times 10^{a(T_{sfc} - 273.16)/(T_{sfc} - b)} \tag{264}$$

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

$$q_{10\text{m}} = \frac{\epsilon e}{p - (1 - \epsilon)e} \tag{265}$$

$$q_s = \frac{\epsilon e_s}{p - (1 - \epsilon)e_s} \tag{266}$$

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:

$$LE\downarrow = \rho_a L C_E V_{wg}(q_{10\mathrm{m}} - q_s). \tag{267}$$

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 [35]. More practical advice on using **cpp** is given by Hazard [25].

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:

```
#ifdef SOLITON
        real(r8) :: g = 1.0_r8                  ! non-dimensional
# elif defined WBC_1 || defined WBC_2 || defined WBC_3
        real(r8) :: g = 9.8_r8                  ! m/s2
# elif defined CIRCLE
        real(r8) :: g = 3.92e-2_r8                  ! m/s2
#else
        real(r8) :: g = 9.81_r8                 ! m/s2
#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 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 [49]). 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.

G.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.

G.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 #.

G.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
```

G.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 §H 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.

G.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**.

G.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.

G.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.

G.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[49]:

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 §G.3.3.

G.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 §2.4.1.

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.

G.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.

G.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.

G.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 (§2.4.1), 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.

G.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-binary-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.

G.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.

G.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.

G.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[49] 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.

G.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. 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.

H sfmakedepend

The other **Perl** script I use with Fortran modifies the **Makefile** to include dependency information, much like the X11 program **makedepend**. I originally wrote **fmakedepend** which was used with traditional Fortran include statements. I later wrote a variant of it for use with the C preprocessor, called **sfmakedepend**. The latest version of **sfmakedepend** does the job of both programs and also searches for the dependencies introduced by Fortran 90 modules. It is used by the **Makefiles** described in §G.

It recursively searches for Fortran style includes, for instance if **file.f** has the statement:

```
include 'commons.h'
```

the line

```
file.o: commons.h
```

will be added to the bottom of the **Makefile**. This tells **make** that **file.o** depends on **commons.h** as well as **file.f**, and to recompile **file.f** whenever **commons.h** is modified. It likewise searches source files for C style includes such as

```
#include "commons.h"
```

and adds the corresponding dependencies to the **Makefile**. It has several options, including **-s**, required for Fortran compilers which will not invoke the C preprocessor for you. In this case the above dependency line would become

```
file.o: commons.h
file.f: commons.h
```

letting **make** know that the C preprocessor must be rerun on **file.F** whenever **commons.h** is updated.

When using the C preprocessor, you can ask it to search directories other than the current directory. Likewise, **sfmakedepend** can be instructed to search other directories with **-I dir** options. Note that it is legal to have more than one **-I dir** option as in:

```
sfmakedepend -I /usr/local/include -I /home/me/include *.F
```

Fortran 90 introduces some interesting dependencies. Two compilers I have access to (NAG **f90** and IBM **xlf**) produce a private **my_module.mod** file if you define **module My_Module** in file **mod.f90**. This file is used by the compiler when you use the module as a consistency check (type-safe programming). If **foo.f90** uses that module, you will need the following dependency information:

```
foo.o: my_module.mod
my_module.mod: mod.o
```

This says that before compiling **foo.f90** we need to have the file **my_module.mod**. This file in turn depends on **mod.o**, so that **mod.f90** must be compiled before **foo.f90**. The sgi is similar except that it uses the file **MY_MODULE.kmo** to store the private module information. Use **sfmakedepend -g** on the SGI.

Rather than creating extra module files, the Cray and Parasoft compilers store the module information in the object file and then files which use the modules need to be compiled with extra flags pointing to the module object files. For instance, if **foo.f90** uses **My_Module** which was defined in **mod.f90**, then you will need to compile **mod.f90** first and provide the Cray compiler with the extra option **-p mod.o** when compiling **foo.f90**. When using the Cray, use **sfmakedepend -c** to get the dependency information:

```
foo.o: mod.o
        $(CFT) $(FFLAGS) -c -p mod.o foo.f90
```

$(CFT) and **$(FFLAGS)** are assumed to be previously defined as the name of the compiler and the compiler options, respectively.

Note: These f90 module dependencies can confuse some versions of **make**, especially of the System V variety. We use gnu **make** because it can follow these chained dependencies and do the right thing.

sfmakedepend assumes that all the files using and defining modules are in the same directory and are all in the list of files to be searched. It seems that the industry has not settled on a practical way to deal with a separate modules directory, anyway.

I sometimes include non-existent files as a compile time consistency check:

```
#ifndef PLOTS
#include "must_define_PLOTS"       /* bogus include */
#endif
```

This program warns about include files it can't find, but not if there is a "bogus" on the same line.

See the comments at the top of **sfmakedepend** for up-to-date information on the options. I may someday get inspired to use a newer version of the **getopt** routine and rename the options to have names like **-SGI** and **-Cray**.

I Subversion

The ROMS source code is distributed using Subversion (svn). There are svn clients available for nearly every operating system and many resources available, including an O'Reilly book [8]. I'll just cover a few basics here, taken in part from the ROMS wiki.

If you wish to maintain your own copy of ROMS in a source code repository, you may want to investigate git, which has the ability to download from an svn server. It too has an O'Reilly book ([44]), plus I enjoyed Swicegood's book ([78]).

I.1 Overview

Subversion is a tool for managing software development that keeps track of who modified what and allows the return to a previous version if changes don't work as expected. All the ROMS/TOMS files are stored in a SVN repository on www.myroms.org with access controlled by requiring authentication with the same ROMS Username/Password combination assigned to registered users of the ROMS Forum.

This svn repository is the official version of the code which only the developers are allowed to change. Users should download the ROMS code to their local machines using an svn client. Don't attempt to use a regular web browser to browse or download files from the svn repository - there are much better tools for interacting with the code repository.

We strongly recommend that users always check out the current *trunk* version since this has the most recent updates and bug fixes. The tags version is kept largely as an historical record of stable releases at the conclusion of major code upgrades.

Below is a general description of how subversion works. Please look at the svn book ([8]) for more detailed information and the wiki for more on some available GUI clients.

I.2 Checking out the code

WARNING: It is strongly suggested that you checkout the ROMS source code using the same operating system you wish to compile and run ROMS on. If you download the code on a Windows machine and wish to run it on a non-Windows machine you will need convert the line endings with a utility like dos2unix or recode. Even with these utilities you may still have problems compiling ROMS.

In order download source code from a Subversion repository, svn client software must be installed on your local machine. If you are compiling subversion on your own be sure to build it with SSL support or you will not be able to download the ROMS source code. Most Linux distributions come with subversion (the command name is svn), so shell commands may be used without installing additional software. If your username on your local computer is not the same as your ROMS username you will need to pass the –username <username> option to svn; an example is given below. The general form of subversion commands is:

```
svn action from to {optional_qualifiers}
```

To check-out the files from the ROMS repository trunk:

```
svn checkout https://www.myroms.org/svn/src/trunk MyDir
```

where MyDir is the destination directory on your local computer. Note the https rather than http. If your username on your local computer is not the same as your ROMS username you will need to pass the –username option to svn:

```
svn checkout --username joe_roms https://www.myroms.org/svn/src/trunk MyDir
```

You only check out once, after that, a hidden directory called .svn exists to keep track of the source, destination and a bunch of other information. Your username and password will also be saved.

I.3 Updates

Now and again, you might feel the urge to get up to speed with the latest changes that have been made to the ROMS repository. When that happens, simply go to the directory that was "MyDir" above and type:

```
svn update
```

Subversion will remember where you checked out from before and see if a newer revision exists. If so, it will download and apply all the relevant changes.

I.4 Code changes

As you use ROMS, you may find yourself adding new files and new chunks of code to existing files. Unless you are a developer with write access to the repository at www.myroms.org, you have no easy way to save your changes within the svn framework, since any one directory can only point to one repository. To keep getting updates from the trunk, you must keep using the svn server at myroms.org. At the very least, saving a tarball before fetching major updates is a prudent step.

I.5 Conflicts

Sometimes when you make changes to your copy of the ROMS code, those changes will conflict with changes made to the repository. This means that the changes from the server overlapped with your own, and now you have to manually choose between them.

Whenever a conflict occurs, three things typically occur to assist you in resolving that conflict:

- Subversion halts during the update, offering you several choices, and remembers that the file is in a state of conflict if you don't clear it right then.
- If Subversion considers the file to be mergeable, it places conflict markers (special strings of text which delimit the âĂIJsidesâĂİ of the conflict, usually "<" and ">" characters) into the file to visibly demonstrate the overlapping areas.
- For every conflicted file, Subversion places three extra unversioned (not under Subversion control) files in your working copy:

filename.mine : This is your file as it existed in your working copy (local copy) before you updated your working copy. This file has only your latest changes in it. (If Subversion considers the file to be unmergeable, then the .mine file isn't created, since it would be identical to the working file.)

filename.rOLDREV : This is the file that was the BASE revision before you updated your working copy. That is, the file that you checked out before you made your latest edits.

filename.rNEWREV : This is the file that your Subversion client just received from the server when you updated your working copy. This file corresponds to the HEAD (latest) revision of the repository.

For example, let's say you checked out revision 280 and made some changes to **User/Functionals/ana_hmixcoef** Now you want to update your ROMS source code to take advantage of a new algorithm but when you run svn update your **ana_hmixcoef.h** is now in conflict with the new version in the repository.

```
> svn update
U Version
U ROMS/Modules/mod_mixing.F
U ROMS/Functionals/ana_hmixcoef.h
```

```
C User/Functionals/ana_hmixcoef.h
Updated to revision 291.

>
```

The above is with an older version of svn. The latest, greatest does this:

```
Conflict discovered in 'ROMS/Utility/inp_par.F'.
Select: (p) postpone, (df) diff-full, (e) edit,
(mc) mine-conflict, (tc) theirs-conflict,
(s) show all options:
```

Selecting (p) will behave as the old version.

If you get a conflict, you need to do one of three things:

- Merge the conflicted text âĂIJby handâĂİ by examining and editing the conflict markers within the file.
- Copy one of the temporary files on top of your working file.
- Run svn revert <filename> to throw away all of your local changes.

Once you've resolved the conflict, you need to let Subversion know by running "svn resolved". This removes the three temporary files and Subversion no longer considers the file to be in a state of conflict. More on this below.

I.5.1 Merging conflicts by hand

To merge your changes with those from the latest revision in the repository, open **User/Functionals/ana_hmixc** in your favorite editor and look for a string of "<" characters. You should see something like this:

```
<<<<<<< .mine
#ifndef DISTRIBUTE
IF (Lanafile.and.(tile.eq.0)) THEN
#else
IF (Lanafile) THEN
#endif
=======
#ifdef DISTRIBUTE
IF (Lanafile) THEN
#else
IF (Lanafile.and.(tile.eq.0)) THEN
#endif
>>>>>>> .r291
```

After comparing the two code blocks (separated by the "=" symbols), you decide that you prefer the logic from the repository so you remove the conflict markers and your code so the section now looks like this:

```
#ifdef DISTRIBUTE
IF (Lanafile) THEN
#else
IF (Lanafile.and.(tile.eq.0)) THEN
#endif
```

Now you can save the file and let Subversion know that you have resolved the conflict:

```
> svn resolved User/Functionals/ana_hmixcoef.h
Resolved conflicted state of 'User/Functionals/ana_hmixcoef.h'
```

I.5.2 Copying a file onto your working file

If you get a conflict and decide that you want to throw out your changes, you can merely copy one of the temporary files created by Subversion over the file in your working copy. Let's say you want to use the new revision from the repository:

```
> cd User/Functionals
> ls ana_hmixcoef.h*
ana_hmixcoef.h ana_hmixcoef.h.mine ana_hmixcoef.h.r280
ana_hmixcoef.h.r291

> cp ana_hmixcoef.h.r291 ana_hmixcoef.h
> svn resolved ana_hmixcoef.h
Resolved conflicted state of 'ana_hmixcoef.h'
```

I.5.3 Punting: Using svn revert

If you get a conflict, and upon examination decide that you want to throw out your changes and start your edits again, just revert your changes:

```
> cd User/Functionals
> svn revert ana_hmixcoef.h
Reverted 'ana_hmixcoef.h'
```

Note that when you revert a conflicted file, you don't have to run "svn resolved".

References

[1] J. S. Allen, P. A. Newberger, and J. Federiuk. Upwelling circulation on the oregon continental shelf. part i: Response to idealized forcing. *J. Phys. Oceanogr.*, 25:1843–1866, 1995.

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

[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. Carton, B. S. Giese, and S. A. Grodsky. Sea level rise and the warming of the oceans in the soda ocean reanalysis. *J. Geophys. Res.*, 110, 2005. doi:10.1029/2004JC002817.

[7] F. Chai, R. C. Dugdale, T.-H. Peng, F. P. Wilkerson, and R. T. Barber. One dimensional ecosystem model of the equatorial pacific upwelling system, part i: model development and silicon and nitrogen cycle. *Deep Sea Res. II*, 49:2713–2745, 2002.

[8] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, second edition, 2008. http://svnbook.red-bean.com/.

[9] E. E. Ebert and J. A. Curry. An intermediate one-dimensional thermodynamic sea ice model for investigating ice-atmosphere interactions. *J. Geophys. Res.*, 98:10085–10109, 1993.

[10] C. W. Fairall, E. F. Bradley, J. E. Hare, A. A. Grachev, and J. B. Edson. Bulk parameterization of air-sea fluxes: Updates and verification for the coare algorithm. *J. Climate*, 16:571–591, 2003.

[11] K. N. Fedorov. Layer thicknesses and effective diffusivities in the diffusive thermocline convection in the ocean. In J. C. J. Nihoul and B. M. Jamart, editors, *Small-scale turbulence and mixing in the ocean*, pages 471–479. Elsevier, New York, 1988.

[12] K. Fennel, J. Wilkin, J. Levin, J. Moisan, J. O'Reilly, and D. Haidvogel. Nitrogen cycling in the mid atlantic bight and implications for the north atlantic nitrogen budget: Results from a three-dimensional model. *Global Biogeochem. Cycles*, 20, 2006. doi:10.1029/2005GB002456.

[13] M. G. G. Foreman. Manual for tidal heights analysis and prediction. Technical Report 77-10, Institute of Ocean Sciences, Patricia Bay, 1977 (revised 1996). Pacific Marine Science Report.

[14] M. G. G. Foreman. Manual for tidal currents analysis and prediction. Technical Report 77-10, Institute of Ocean Sciences, Patricia Bay, 1978 (revised 1996). Pacific Marine Science Report.

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

[16] P. J. S. Franks, J. S. Wroblewski, and G. R. Flierl. Behavior of a simple plankton model with food-level acclimation by herbivores. *Mar. Biol.*, 91:121–129, 1986.

[17] 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.

[18] B. Galperin, L. H. Kantha, S. Hassid, and A. Rosati. A quasi-equilibrium turbulent energy model for geophysical flows. *J. Atmos. Sci.*, 45:55–62, 1988.

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

[20] S.M. Griffies, A. Gnanadesikan, R.C. Pacanowski, V. Larichev, J.K. Dukowicz, and R.D. Smith. Isoneutral diffusion in a z-coordinate ocean model. *J. Phys. Oceanogr.*, 28:805–830, 1998.

[21] S.M. Griffies and R.W. Hallberg. Biharmonic friction with a smagorinsky-like viscosity for use in large-scale eddy-permitting ocean models. *Mon. Wea. Rev.*, 128:2935–2946, 2000.

[22] 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.

[23] 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.

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

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

[26] 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.

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

[28] S. Hinckley, K. O. Coyle, G. Gibson, A. J. Hermann, and E. L. Dobbins. A biophysical npz model with iron for the gulf of alaska: Reproducing the differences between an oceanic hnlc ecosystem and a classical northern temperate shelf ecosystem. *Deep Sea Res. II*, 2009. in press.

[29] W. R. Holland, J. C. Chow, and F. O. Bryan. Application of a third-order upwind scheme in the ncar ocean model. *J. Climate*, 11:1487–1493, 1998.

[30] E. C. Hunke. Viscous-plastic sea ice dynamics with the evp model: linearization issues. *J. Comp. Phys.*, 170:18–38, 2001.

[31] E. C. Hunke and J. K. Dukowicz. An elastic-viscous-plastic model for sea ice dynamics. *J. Phys. Oceanogr.*, 27:1849–1868, 1997.

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

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

[34] R. F. Keeling, B. B. Stephens, R. G. Najjar, S. C. Doney, D. Archer, and M. Heimann. Seasonal variations in the atmospherice o2/n2 ratio in relation to kinetics of air-sea gas exchange. *Global Biogeochem. Cycles*, 12:141–163, 1998.

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

[36] M. J. Kishi, M. Kashiwai, D. M. Ware, B. A. Megrey, D. L. Eslinger, F. E. Werner, M. Noguchi-Aita, T. Azumaya, M. Fujii, S. Hashimoto, D. Huang, H. Iizumi, Y. Ishida, S. Kang, G. A. Kantakov, H.-C. Kim, K. Komatsu, V. V. Navrotsky, S. Lan Smith, K. Tadokoro, A. Tsuda, O. Yamamura, Y. Yamanaka, K. Yokouchi, N. Yoshie, J. Zhang, Y. I. Zuenko, and V. I. Zvalinsky. Nemuro—a lower trophic level model for the north pacific marine ecosystem. *Ecological Modelling*, 2002:12–25, 2007.

[37] H. Lamb. *Hydrodynamcis*. Cambridge University Press, 6 edition, 1932. (1945 Dover Publications reproduction).

[38] W. G. Large. Modeling and parameterization ocean planetary boundary layers. In E. P. Chassignet and J. Verron, editors, *Ocean Modeling and Parameterization*, pages 81–120. Kluwer Academic Publishers, 1998.

[39] 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.

[40] W. G. Large and S. G. Yeager. The global climatology of an interannually varying air-sea flux data set. *Clim. Dyn.*, 33:341–364, 2008. DOI 10.1007/s00382-0008-0441-3.

[41] J. R. Ledwell, A. J. Wilson, and C. S. Low. Evidence for slow mixing across the pycnocline from an open-ocean tracer-release experiment. *Nature*, 364:701–703, 1993.

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

[43] S.-J. Lin. A finite volume integration method for computing pressure gradient force in general vertical coordinates. *Quart. J. R. Met. Soc.*, 123:1749–1762, 1997.

[44] Jon Loeliger. *Version Control with Git*. O'Reilly & Associates, Inc., Sebastopol, CA, first edition, 2009.

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

[46] J. Mailhôt and R. Benoit. A finite-element model of the atmospheric boundary layer suitable for use with numerical weather prediction models. *J. Atmos. Sci.*, 39:2249–2266, 1982.

[47] L. Margolin and P. K. Smolarkievicz. Antidiffusive velocities for multipass donor cell advection. *SIAM J. Sci. Comput.*, pages 907–929, 1998.

[48] 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.

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

[50] G. L. Mellor. The three-dimensional current and surface wave equations. *J. Phys. Oceanogr.*, 33:1978–1989, 2003.

[51] G. L. Mellor. Some consequences of the three-dimensional currents and surface wave equation. *J. Phys. Oceanogr.*, 35:2291–2298, 2005.

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

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

[54] G. L. Mellor and T. Yamada. Development of a turbulence closure model for geophysical fluid problems. *Rev. Geophys. Space Phys.*, 20:851–875, 1982.

[55] W. A. Oost, G. J. Komen, C. M. J. Jacobs, and C. van Oort. New evidence for a relation between wind stress and wave age from measurements during asgamage. *Bound.-Layer Meteor.*, 103:409–438, 2002.

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

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

[58] 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.

[59] H. Peters, M. C. Gregg, and J. M. Toole. On the parameterization of equatorial turbulence. *J. Geophys. Res.*, 93:1199–1218, 1988.

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

[61] T. M. Powell, C. V. W. Lewis, E. N. Curchitser, D. B. Haidvogel, A. J. Hermann, and E. L. Dobbins. Results from a three-dimensional, nested biological-physical model of the california current system and comparisons with statistics from satellite imagery. *J. Geophys. Res.*, 111(C07018), 2006. doi:10.1029/2004JC002506.

[62] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes, The Art of Scientific Computing*. Cambridge University Press, 1986.

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

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

[65] 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.

[66] R. Sadourny and K. Maynard. Formulations of lateral diffusion in geophysical fluid dynamics models. In C.A. Lin, R. Laprise, and H. Ritchie, editors, *Numerical Methods of Atmospheric and Oceanic Modelling*, pages 547–556. NRC Research Press, 1997.

[67] A. J. Semtner, Jr. A model for the thermodynamic growth of sea ice in numerical investigations of climate. *J. Phys. Oceanogr.*, 6:379–389, 1976.

[68] A. F. Shchepetkin and J. C. McWilliams. Quasi-monotone advection schemes based on explicit locally adaptive dissipation. *Mon. Wea. Rev.*, 126:1541–1580, 1998.

[69] A. F. Shchepetkin and J. C. McWilliams. A method for computing horizontal pressure-gradient force in an oceanic model with a nonaligned vertical coordinate. *J. Geophys. Res.*, 108:1–34, 2003.

[70] 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.

[71] 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.

[72] 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, 2009. in press.

[73] J. Smagorinsky. General circulation experiments with the primitive equations. i. the basic experiment. *Mon. Wea. Rev.*, 91:99–164, 1963.

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

[75] Y. Song. A general pressure gradient formulation for ocean models. part i: Scheme design and diagnostic analysis. *Mon. Wea. Rev.*, 126:3213–3230, 1998.

[76] 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.

[77] 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.

[78] Travis Swicegood. *Pragmatic Version Control Using Git*. Pragmatic Bookshelf, Raleigh, North Carolina and Dallas, Texas, first edition, 2008.

[79] P. K. Taylor and M. A. Yelland. The dependence of sea surface roughness on the height and steepness of the waves. *J. Phys. Oceanogr.*, 31:572–590, 2001.

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

[81] I. B. Troen and L. Mahrt. A simple model of the atmospheric boundary layer; sensitivity to surface evaporation. *Boundary-Layer Meteor.*, 37:129–148, 1986.

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

[83] R. C. Wajsowicz. A consistent formulation of the anisotropic stress tensor for use in models of the large-scale ocean circulation. *J. Comp. Phys.*, 105:333–338, 1993.

[84] J. C. Warner, C. R. Sherwood, R. P. Signell, C. K. Harris, and H. G. Arango. Development of a three-dimensional, regional, coupled wave, current, and sediment-transport model. *Computers & Geosciences*, 34:1284–1306, 2008.

[85] D. J. Webb, B. A. De Cuevas, and C. S. Richmond. Improved advection schemes for ocean models. *J. Atmos. Ocean. Tech.*, 15:1171–1187, 1998.

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

U.S. Department of the Interior
Minerals Management Service
Alaska OCS Region

The Department of the Interior Mission

As the Nation's principal conservation agency, the Department of the Interior has responsibility for most of our nationally owned public lands and natural resources. This includes fostering sound use of our land and water resources; protecting our fish, wildlife, and biological diversity; preserving the environmental and cultural values of our national parks and historical places; and providing for the enjoyment of life through outdoor recreation. The Department assesses our energy and mineral resources and works to ensure that their development is in the best interests of all our people by encouraging stewardship and citizen participation in their care. The Department also has a major responsibility for American Indian reservation communities and for people who live in island territories under U.S. administration.

The Minerals Management Service Mission

As a bureau of the Department of the Interior, the Minerals Management Service's (MMS) primary responsibilities are to manage the mineral resources located on the Nation's Outer Continental Shelf (OCS), collect revenue from the Federal OCS and onshore Federal and Indian lands, and distribute those revenues.

Moreover, in working to meet its responsibilities, the **Offshore Minerals Management Program** administers the OCS competitive leasing program and oversees the safe and environmentally sound exploration and production of our Nation's offshore natural gas, oil and other mineral resources. The MMS **Minerals Revenue Management Program** meets its responsibilities by ensuring the efficient, timely and accurate collection and disbursement of revenue from mineral leasing and production due to Indian tribes and allottees, States and the U.S. Treasury.

The MMS strives to fulfill its responsibilities through the general guiding principles of: (1) being responsive to the public's concerns and interests by maintaining a dialogue with all potentially affected parties and (2) carrying out its programs with an emphasis on working to enhance the quality of life for all Americans by lending MMS assistance and expertise to economic development and environmental protection.