Naval Research Laboratory

Stennis Space Center, MS 39529-5004

NRL/MR/7320--09-9166

Software Design Description for the HYbrid Coordinate Ocean Model (HYCOM) Version 2.2

A.J. WALLCRAFT
E.J. METZGER

Ocean Dynamics and Prediction Branch
Oceanography Division

S.N. CARROLL

QinetIQ North America
Planning Systems, Inc.
Slidell, LA

February 12, 2009

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
12-02-2009	Memorandum Report	

4. TITLE AND SUBTITLE

Software Design Description for the HYbrid Coordinate Ocean Model (HYCOM) Version 2.2

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0603207N

6. AUTHOR(S)

A.J. Wallcraft, E.J. Metzger, and S.N. Carroll*

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
73-5094-19-5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Oceanography Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/7320--09-9166

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Space & Naval Warfare Systems Command
2451 Crystal Drive
Arlington, VA 22245-5200

10. SPONSOR / MONITOR'S ACRONYM(S)

SPAWAR

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

*QinetIQ North America, Planning Systems, Inc., Slidell, LA

14. ABSTRACT

This document describes the software design and code of the HYbrid Coordinate Ocean Model (HYCOM) version 2.2. It includes the mathematical formulation and solution procedures for HYCOM 2.2 as well as flow charts and descriptions of the programs, modules, and subroutines.

15. SUBJECT TERMS

Numerical ocean modeling
HYCOM

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			E. Joseph Metzger
Unclassified	Unclassified	Unclassified	UL	155	**19b. TELEPHONE NUMBER** *(include area code)* (228) 688-4762

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

TABLES AND FIGURES

1.0 SCOPE

1.1 Introduction

The HYbrid Coordinate Ocean Model (HYCOM; (Bleck, 2002) is a primitive equation, general circulation model. The vertical coordinates are isopycnal in the open, stratified ocean, but use the layered continuity equation to make a dynamically smooth transition to terrain-following coordinates in shallow coastal regions, and to *z*-level coordinates in the mixed layer and/or unstratified seas. The hybrid coordinate extends the geographic range of applicability of traditional isopycnic coordinate circulation models toward shallow coastal seas and unstratified parts of the world ocean. It maintains the significant advantages of an isopycnal model in stratified regions while allowing more vertical resolution near the surface and in shallow coastal areas, hence providing a better representation of the upper ocean physics. HYCOM is designed to provide a major advance over the existing operational global ocean prediction systems, since it overcomes design limitations of the present systems as well as limitations in vertical and horizontal resolution. The result should be a more streamlined system with improved performance and an extended range of applicability (e.g., the present systems are seriously limited in shallow water and in handling the transition from deep to shallow water).

Global HYCOM with 1/12° horizontal resolution at the equator (~7 km at mid-latitudes) is the ocean model component of an eddy-resolving operational nowcast/forecast system. It will provide nowcasts and forecasts of the three dimensional global ocean environment. HYCOM is initially delivered with a thermodynamic "energy loan" ice model, but later will be coupled to the Polar Ice Prediction System 3.0 (Posey *et al*, 2008a, 2008b, 2009) via the Earth System Modeling Framework (ESMF) (Hill *et al*., 2004). Coupling between the ocean and ice models will more properly account for the momentum, heat and salt fluxes at the ocean/ice interface. The final component of the nowcast/forecast system is the Navy Coupled Ocean Data Assimilation (NCODA) which is a multivariate optimal interpolation scheme that will be used to assimilate surface observations from satellites, including altimeter and Multi-Channel Sea Surface Temperature (MCSST) data, sea ice concentration and profile data such as XBTs (expendable bathythermographs), CTDs (conductivity temperature depth) and ARGO floats (Cummings, 2006). By combining these observations via data assimilation and using the dynamical interpolation skill of the model, the three dimensional ocean state can be more accurately nowcast and forecast.

The HYCOM user has control over setting up the model domain, generating the forcing fields, and ingesting either the climatology or output fields from other model simulations to use for interior relaxation and boundary. The model is fully parallelized and designed to be portable among a variety of systems. HYCOM 2.2 is equipped with an improved inert tracer code that serves as a full-fledged prognostic scalar carried at both leapfrog time steps.

There are five primary vertical mixing algorithms, of which three are ‘‘continuous’’ differential models and two are bulk (slab) models (See Halliwell (2003) for more details). The three differential models are the nonlocal K-Profile Parameterization (KPP; Large *et al*., 1994], the NASA Goddard Institute for Space Studies level 2 turbulence closure (GISS; Canuto *et al*., 2001, 2002), and the Mellor–Yamada level 2.5 turbulence closure (MY; Mellor and Yamada,

Manuscript approved December 29, 2008.

1982). These models govern vertical mixing throughout the water column. As of this writing, the system that was transitioned uses KPP. The bulk models include the dynamical instability model of Price *et al*. (1986) (PWP) and the Kraus–Turner model.

HYCOM 2.2 is a result of collaborative efforts between the University of Miami, Florida State University, the Los Alamos National Laboratory, the Naval Research Laboratory (NRL) and other institutions that make up the HYCOM consortium. Ongoing HYCOM research has been funded under the National Oceanographic Partnership Program (NOPP) and the Office of Naval Research (ONR).

1.2 Document Overview

The purpose of this Software Design Description (SDD) is to describe the software design and code of the HYbrid COordinate Model (HYCOM) Version 2.2. The SDD includes the mathematical formulation and solution procedures for HYCOM 2.2 as well as flow charts and descriptions of the programs, modules, and subroutines. This document, along with a User's Manual (Metzger *et al*., 2009), forms the HYCOM 2.2 documentation package. More information on HYCOM 2.2, including presentations, articles, and previous version documentation packages is available at http://hycom.rsmas.miami.edu.

In general, subroutine names are shown in *italics* and file names are in **boldface** in this document. Source code references are given in typeface, or `Courier`, font. Symbols used in the code are typewritten, while corresponding symbols in this document are in the *math* font which, granted, is similar to *italic*.

2.0 REFERENCED DOCUMENTS

2.1 Government Documentation

Cummings, J. and Carroll, S., (2006). Software User's Manual for the Navy Coupled Ocean Data Assimilation (NCODA) System, *NRL Tech. Rpt.,* MRY-001-06, Naval Research Laboratory, Monterey, CA.

Metzger E.J., O.M. Smedstad and S.N. Carroll, (2009). User's Manual for Global Ocean Forecast System (GOFS) Version 3.0 (V3.0). *NRL Memo. Rpt.,* NRL/MR/7320—09-9175 Ocean Modeling Division, Naval Research Laboratory, Stennis Space Center, MS.

Posey, P.G., L.F. Smedstad, R.H. Preller, E.J. Metzger and S.N. Carroll. (2008a). "Software Design Description for the Polar Ice Prediction System (PIPS) Version 3.0" *NRL Memo. Rpt.,* NRL/MR/7320--08-9150, Ocean Modeling Division, Naval Research Laboratory, Stennis Space Center, MS.

Posey P.G., L.F. Smedstad, R.H. Preller, E.J. Metzger and S.N. Carroll. (2008b). "User's Manual for the Polar Ice Prediction System (PIPS) Version 3.0", *NRL Memo. Rpt.,* NRL/MR/7320--08-9154, Ocean Modeling Division, Naval Research Laboratory, Stennis Space Center, MS.

Posey P.G., L.F. Smedstad, R.H. Preller, E.J. Metzger, C.N. Barron, M. Phelps and S.N. Carroll. (2009). "Validation Test Report for the Polar Ice Prediction System (PIPS) Version 3.0", *NRL Memo. Rpt.,* NRL/MR/7320--09-xxxx, Ocean Modeling Division, Naval Research Laboratory, Stennis Space Center, MS (in preparation).

2.2 General Technical Documentation

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

Adcroft, A., and Hallberg R., (2006). On methods for solving the oceanic equations of motion in generalized vertical coordinates. *Ocean Modelling*, 11: 224-233.

Baraille, R. and Filatoff N., (1995). Modèle shallow-water multicouches isopycnal de Miami. *Rapport d'Etude*, CMO/RE No 003/95.

Barron, C.N. and Smedstad L.F., (2002). Global river inflow within the Navy Coastal Ocean Model. Proceedings of the Marine Technology Society, MTS/IEEE Oceans 2002 Conference, 1472-1479.

Bleck, R., (2002). An oceanic general circulation model framed in hybrid isopycnic-Cartesian coordinates. *Ocean Modelling*, 4: 55-88.

Bleck, R. and, Smith L.T., (1990). A wind-driven isopycnic coordinate model of the north and equatorial Atlantic ocean. I-model development and supporting experiments. *J. Geophys. Res*., 95(C): 3273-3285.

Bleck, R., Rooth C., Hu D., and Smith L.T., (1992). Salinity-driven thermocline transients in a wind- and thermohaline-forced isopycnic coordinate model of the North Atlantic. *J. Phys. Oceanogr*., 22: 1486-1505.

Bleck, R., Rooth C., Hu D., and Smith L.T., (1992). Ventilation patterns and mode water formation in a wind- and thermodynamically driven isopycnic coordinate model of the north Atlantic. *J. Phys. Oceanogr*., 22: 1486-1505.

Browning, G. L. and Kreiss, H.-O., (1982). Initialization of the shallow water equations with open boundaries by the bounded derivative method. *Tellus*, 34: 334-351.

Browning, G. L. and Kreiss, H.-O., (1986). Scaling and computation of smooth atmospheric motions. *Tellus*, 38A: 295-313.

Brydon, D., Sun S., and Bleck R., (1999). A new approximation of the equation of state for seawater, suitable for numerical models. *J. Geophys. Res.-Ocns*. 104: 1537.

Canuto, V.M., Howard, A., Cheng, Y., Dubovikov, M.S., (2001). Ocean turbulence. Part I: One-point closure model. Momentum and heat vertical diffusivities. *J. Phys. Oceanogr*. 31: 1413–1426.

Canuto, V.M., Howard, A., Cheng, Y., Dubovikov, M.S., (2002). Ocean turbulence. Part II: Vertical diffusivities of momentum, heat, salt, mass, and passive scalars. *J. Phys. Oceanogr*. 32: 240–264.

Carnes, M., (2002). Data base description for the Generalized Digital Environmental Model (GDEM-V) (U), version 3.0, U.S. Naval Oceanographic Office technical report, 27 pp., Oceanogr. Data Bases Div., Stennis Space Cent., Miss.

Cummings, J.A., (2005). Operational multivariate ocean data assimilation. *Quart. J. Royal Met. Soc*., 131: 3583-3604.

Ezer, T. and Mellor G. L., (2004). A generalized coordinate ocean model and a comparison of the bottom boundary layer dynamics in terrain-following and in *z*-level models. *Ocean Modelling*, 6: 379-403.

Garraffo, Z.D., Mariano A. J., Griffa A., Veneziani C., and Chassignet E. P., (2001a). Lagrangian data in a high resolution numerical simulation of the North Atlantic. I: Comparison with *in-situ* float data. *J. Mar. Sys*., 29: 157-176.

Garraffo, Z. D., Griffa A., Mariano A. J., and Chassignet E. P., (2001b). Lagrangian data in a high resolution numerical simulation of the North Atlantic. II: On the pseudo-Eulerian averaging of Lagrangian data. *J. Mar. Sys*., 29: 177-200.

Griffa, A., (1996). Applications of stochastic particles models to oceanographic problems. In: Stochastic Modeling in Physical Oceanography, Adler, Muller, and Rozovoskii, editors, pp. 114-140.

Halliwell Jr., G.R., (2003). Evaluation of Vertical Coordinate and Vertical Mixing Algorithms in the HYbrid Coordinate Ocean Model (HYCOM). *Ocean Modelling,* 7: 285-322.

Halliwell Jr., G.R., Bleck, R., and Chassignet, E.P., (1998). Atlantic ocean simulations performed using a new HYbrid Coordinate Ocean Model (HYCOM). *EOS, Fall AGU Meeting*.

Halliwell Jr., G.R., Bleck, R., Chassignet, E.P., and Smith, L.T., (2000). Mixed layer model validation in Atlantic ocean simulations using the HYbrid Coordinate Ocean Model (HYCOM). *EOS,* 80: OS304.

Hill, C., DeLuca C., Balaji V., Suarez M., da Silva A., (2004). The Architecture of the Earth System Modeling Framework, *Computing in Science and Engineering,* 6 (1).

Jerlov, N. G., (1976). Marine Optics. Elsevier, New York.

Kara, A. B., Rochford P A., and Hurlburt H. E., (2000). Efficient and accurate bulk parameterizations of air-sea fluxes for use in general circulation models. *J. Atmos. Ocean Tech*., 17: 1421-1438.

Large, W.G., McWilliams, J.C. and Doney, S.C., (1994). Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization. *Rev. Geophys.*, 32: 363-403.

Large, W.G., Danabasoglu, G., Doney, S.C., and McWilliams, J.C., (1997). Sensitivity to surface forcing and boundary layer mixing in a global ocean model: Annual-mean climatology. *J. Phys. Oceanogr.*, 27: 2418-2447.

Mariano, A. J. and Brown O. B., (1992). Efficient objective analysis of dynamically heterogeneous and nonstationary fields via the parameter matrix. *Deep Sea Res*., 39: 1255-1292.

McDougall, T. J. and Dewar W. K., (1998). Vertical mixing and cabbeling in layered models. *J. Phys. Oceanogr.*, 28: 1458-1480.

Mellor, G.L. and Yamada, T., (1982). Development of a turbulence closure model for geophysical fluid problems. *Rev. Geophys. Space Phys.,* 20: 851-875.

Mellor, G.L., (1998). User's guide for a three dimensional, primitive equation numerical ocean model. *AOS Program Report*, Princeton University, Princeton, NJ. 34 pp.

Oliger, J. and Sundstrom A., (1978). Theoretical and practical aspects of some initial boundary value problems in fluid mechanics. *SIAM Appl. Math*., 35: 419-446.

Perry, G.D., Duffy P.B., and Miller, N.L., (1996). An extended data set of river discharges for validation of general circulation models. *J. Geophys. Res*., 101: 21339-21349.

Peters, H., Gregg M. C., and Toole J. M., (1988). On the parameterization of equatorial turbulence. *J. Geophys. Res.*, 93: 1199-1218.

Price, J.F., Weller, R.A., and Pinkel, R., (1986). Diurnal cycling: Observations and models of the upper ocean response to diurnal heating, cooling and wind mixing. *J. Geophys. Res.*, 91: 8411-8427.

Semtner, A.J. Jr., (1976). A model for the thermodynamic growth of sea ice in numerical simulations of climate. *J. Phys. Oceanogr.*, 6: 379-389.

Smolarkiewicz, P.K. and Clark, T.L., (1986). The multi-dimensional positive definite advection transport algorithm: further development and applications. *J. Comput. Phys*., 67: 396-438.

Smolarkiewicz, P.K., and Grabowski, W.W., (1993). Layer outcropping in numerical models of stratified flows. *J. Phys. Oceanogr*., 23: 1877-1884.

Sun, S., Bleck R., Rooth C., Dukowicz J., Chassignet E., and Killworth P., (1999). Inclusion of thermobaricity in isopycnic-coordinate ocean models. *J. Phys. Oceanogr.*, 29: 2719-2729.

Zalesak, S., (1979). Fully multi-dimensional flux-corrected transport algorithms for fluids. *J. Comput. Phys*., 31: 335-362.

3.0 HYCOM 2.2 SOFTWARE SUMMARY

3.1 Code Modifications

- HYCOM 2.2 maintains all of the features of HYCOM 2.1, including:
  - Orthogonal curvilinear grids.
  - Emulation of Z, Sigma or Sigma-Z models.
  - KPP, Mellor-Yamada 2.5, Price-Weller Pinkel or Kraus-Turner mixed layer models.
  - Multiple tracers.
  - Off-line one-way nesting.
- HYCOM 2.2 has alternative scalar advection techniques (Donor Cell, FCT (2^{nd} and 4^{th} order), and MPDATA, with FCT2 replacing MPDATA as the standard scheme).
- Vertical remapping uses the piecewise linear method (PLM) for fixed coordinate layers. Stability is derived from locally referenced potential density and layer target densities are spatially varying, with different isopycnal layers in semi-enclosed seas.
- The GISS mixed layer model has been incorporated.
- Atmospheric forcing includes an option to input *ustar* fields. The user may also relax to observed SST fields. COARE 3.0 bulk exchange coefficients have been improved.
- Climatological heat flux offset, Qc, has been added. A heat flux offset (W/m^2) is applied to the ocean surface to reduce mean SST bias across the globe. The offset is based on mean temperature error between observations and HYCOM 2.2 simulations. A 45 W/m^2 heat flux will change SST by approximately 1° C.
- Improved support for rivers. Rivers are represented as bogused surface precipitation. The model treats rivers as a "runoff" addition to the surface precipitation field. Monthly mean river discharge values were constructed from Perry *et al*. (1996), which had one mean value for each river. The set was converted to monthly values for use in ocean modeling studies (Barron and Smedstad, 2002).
- Precipitation acts as a freshwater surface flux that in turn decreases the salinity of the top layer.

4.0 HYCOM 2.2 SOFTWARE INVENTORY

4.1 HYCOM 2.2 Components

A complete description of each HYCOM 2.2 module and its subsequent subroutines, including a definition, purpose, relationship to other modules, etc. is found in Section 6.0. Common Blocks are described in Section 7.0. The HYCOM 2.2 software is run through a series of modules, makefiles, input files, and scripts. These are provided along with descriptions and user instructions in the HYCOM 2.2 User's Manual (Wallcraft *et al*., 2008).

4.1.1 HYCOM 2.2 Source Files

archiv.f, barotp.f, bigrid.f, blkdat.f, cnuity.f, convec.f, diapfl.f, dpthuv.f, dpudpv.f, forfun.f, geopar.f, hybgen.f, hycom.f, hycom_cice.f, icloan.f, inicon.f, inigiss.f, inikpp.f, inimy.f, isnan.f, latbdy.f, machine.f, matinv.f, mod_floats.f, mod_hycom.f, mod_incupd.f, mod_OICPL.f, mod_pipe.f, mod_tides.f, mod_xc.f, mod_za.f, momtum.f, mxkprf.f, mxkrt.f, mxkrtm.f, mxpwp.f, overtn.f, poflat.f, prtmsk.f, psmoo.f, restart.f, thermf.f, trcupd.f, tsadvc.f, wtime.f

4.1.2 HYCOM 2.2 Subroutines

advem, advem_fct2, advem_fct4, advem_mpdata, advem_pcm, Archive_ESMP, blkin8, blkini, blkinl, blkinr, convch, convcm, diapfl1aiij, diapf1aj, diapf1bj, diapf1uif, diapf1vij, diapf2, diapf2j, diapf2, diapf3j, dpudpvj, Export_ESMP, f_invmtx, f_stat, floats, floats_init, floats_restart, flush, forday, getenv, hat53, hybgenaj, hybgenbj, HYCOM_Final, HYCOM_Init, HYCOM_Run, HYCOM_SetServices, ieee_retrospective, Import_ESMP, incupd, incupd_init, incupd_rd, incupd_read, indxi, indxj, initrc, interp2d_expabs, intrph, Setup_ESMP, latbdf, latbdp(n), latbdt, machine, mlbdep, momtum_hs, mxgissaij, mxkppaij, mxkprfaj, mxkprfbij, mxkprfbj, mxkprfciju, mxkprfcijv, mxkprfcj, mxkrta, mxkrtaaj, mxkrtabj, mxkrtb, mxkrtbaj, mxkrtbbj, mxkrtmaj, mxkrtmbj, mxmyaij, mxpwpaij, mxpwpaj, mxpwpbiju, mxpwpbijv, mspwpbj, OICPL_Final, OICPL_Init, OICPL_Run_I2O, OICPL_Run_O2I, OICPL_SetServices, ousal2, pcmtrc, pipe_comparall, pipe_compare, pipe_compare_notneg, pipe_compare_same, pipe_compare_sym1, pipe_compare_sym2, pipe_init, plmtrc, plmtrcs, plmtrcx, psmooth_dif, psmooth_ice, psmooth_new, rantab, rantab_ini, restart_in, restart_in3d, restart_out, rtwi, smshsc_a3, thermf, thermf_oi, thermfj, tidal_arguments, tides_astrol, tides_driver, tides_force, tides_mkw, tides_nodal, tides_set, trcupd, trcupd_903, trcupd_904, tridcof, tridmat, tridrhs, tsdff_1x, tsdff_2x, wscale, xlflush, zaio_endian

4.1.3 HYCOM 2.2 Common Blocks

Named common blocks are defined in the common_blocks.h file and are called from several HYCOM 2.2 subroutines. The common blocks are described in Section 7.0.

common/bb, common/consts, common/czgetc, common/czioxr, common/czioxw, common/ czioxx, common/forcing, common/gissi1, common/gissi2, common/gissi3, common/gissr1, common/gissr2, common/gissr3, common/ halobp, common/hycom1c, common/hycom1i, common/hycom1r, common/hycom2r, common/hycom3r, common/hycom4i, common/hycom4r, common/hycom5i, common/hycom5r common/ iovars, common/kppi, common/ kppltr, common/kppr, common/linepr, common/momtumr4, common/mxgissij_b, common/myr, common/parms1i, common/parms1r, common/pwpr, common/ rdforfi, common/swtchs, common/ testpt, common/tidef, common/varblsd, common/varblsi, common/varblsr, common/wall1i, common/wallr, common/xcagetr, common/xclgetr, common/xcmaxr4, common/xcmpii, common/xcsum8, common/xctilr4, common/xctilra

4.2 HYCOM 2.2 Software Organization and Implementation

4.2.1 Directory Structure

HYCOM 2.2 is set up to be domain independent, and all of the pre- and post-processing programs except the model code are compiled only once. The model code must be recompiled for each new domain, but only one file, **dimensions.h**, is region-specific. The model script is configured to allow data files (input and output) to be resident on a different machine (e.g., an archive system). The actual run initiates from a scratch directory, and files are copied from scratch to permanent storage (possibly on another machine) using commands associated with the `pget` and `pput` environment variables. If all model components reside on a single machine (or if the archive directory is Network File System (NFS) mounted on the run machine), `pget` and `pput` can both be `cp` (e.g., `setenv pget /usr/bin/cp`). Otherwise, the `../bin` directory contains several examples of appropriate `pput` and `pget` commands.

The directory ~${user}/hycom/ALL contains all of the domain-independent pre- and post-processing programs for running HYCOM 2.2 globally. These programs are found in the following subdirectories:

archive/src_*/	Source code for modifying HYCOM 2.2 archive files and converting them to other file formats.
bin/	Utilities – this directory should be in the user's path.
config/	Machine and parallelization-specific part of makefiles.
force/src_*/	Source code for interpolation of NRL format wind/flux files on “native” grid to the model grid.
libsrc/	Common source files.

plot/src_*/	Source code for plotting archive files and 2D fields from any HYCOM ".a" file using National Center for Atmospheric Research (NCAR) graphics.
relax/src_*/	Source code for interpolation of a climatology to a HYCOM model grid for use in boundary relaxation zones or for model initialization.
subregion/src_*/	Source code for extraction of a subregion from an archive file.
topo/src_*/	Source code for bathymetry processing.

A second directory contains the data files needed to run a simulation for a specific domain. For example, everything needed to process and run HYCOM globally (GLBa) in conjunction with the ALL directory is found in the ~${user}/hycom/GLBa0.08 directory. The following subdirectories are located within the hycom_2.2.00_ GLBa directory:

archive/	Script for modifying HYCOM 2.1 to 2.2 archive files.
config/	Machine and parallelization specific part of makefiles.
doc/	Documentation.
expt_01.0/	Example simulation.
data/	
force/	Atmospheric forcing data files.
offset/	
plot/	
include/	Region definition for setup programs.
meanstd/src_*/	Mean and standard deviation of HYCOM 2.2 archive fields.
plot/	Example of plotting and scripts.
relax/	Relaxation data files and scripts.
010/	"Isopycnal" climatology for a HYCOM 2.2 simulation.
gdem3/	GDEM3 climatology on the horizontal grid.
plot/	Plot climatology.
src_2.2.00_32_one/	HYCOM 2.2 source code for 32 layers and Message Passing Interface (MPI).
subregion/	
topo/	Bathymetry data files and scripts.
partit/	Partition text file for domain decomposition into tiles.

5.0 HYCOM 2.2 DETAILED DESIGN

The following sections give a detailed description of the purpose, variables, logic and constraints for the software elements in the model.

5.1 Constraints and Limitations

1. Sea surface temperature is the average temperature across the top layer, which is typically 3 m thick.

5.2 HYCOM Logic and Basic Equations

5.2.1 HYCOM Boundary Conditions

HYCOM 2.2 is equipped with two types of boundary conditions: Newtonian relaxation in sponge layers and full open-ocean boundary conditions.

Relaxation Boundary Conditions

HYCOM 2.2 includes a simple Newtonian relaxation scheme that can be used for sponge boundary zones or for relaxation to climatology within any model subdomain specified by the user. Within relaxation boundary zones, temperature, salinity, and vertical coordinate pressure levels are updated for each time step as follows:

$$\begin{aligned} T_{t+1}^k &= T_t^k + \Delta t\mu\left(\hat{T}_t^k - T_t^k\right) \\ S_{t+1}^k &= S_t^k + \Delta t\mu\left(\hat{S}_t^k - S_t^k\right) \\ p_{t+1}^k &= p_t^k + \Delta t\mu\left(\hat{p}_t^k - p_t^k\right), \end{aligned} \tag{1}$$

where the hat indicates GDEM3 (Carnes, 2002) climatology, k is the layer or interface number, and μ^{-1} represents the relaxation time scale. The user specifies the values of μ^{-1} at each grid point, setting it to nonzero values where relaxation is to be performed. This results in a two-dimensional mask that defines relaxation zones. Offline, HYCOM will first horizontally interpolate GDEM3 climatology to model grid points at the original z-levels, and then transform these vertical profiles to isopycnic coordinates at each model grid point. Thus, the profiles of $\hat{T}$, $\hat{S}$, and $\hat{p}$ used in (1) are isopycnic beneath the surface mixed layer.

When HYCOM 2.2 is run with isopycnic vertical coordinates, T and S are both relaxed in the non-isopycnic mixed layer (layer 1), while salinity only is relaxed in deeper layers with temperature diagnosed from the equation of state to preserve the isopycnic reference density.

All interfaces except number 2 are relaxed to avoid adjusting the mixed layer base (MLB). Of course, all interfaces greater than two are prevented from becoming shallower than interface 2.

When HYCOM 2.2 is run with hybrid vertical coordinates, both T and S are relaxed in the upper n_{hyb} layers, where n_{hyb} is the user-specified number of hybrid layers. Salinity alone is relaxed in deeper layers, with temperature being diagnosed from the equation of state. In the hybrid coordinate mode all pressure interfaces are relaxed to climatology.

Open Boundary Conditions

The main features of the HYCOM 2.2 open boundary scheme are as follows:

1. There is no distinction between inflow and outflow boundaries. This approach is taken in recognition of the fact that, regardless of the direction of the physical flow, information generally passes through the boundary in both directions. Making a distinction between inflow and outflow boundaries is, therefore, justified only with regard to advection of material properties, such as temperature, salinity, and potential vorticity.

2. Boundary conditions for barotropic and baroclinic modes are formulated separately.

3. The boundary conditions developed by Browning and Kreiss (1982, 1986), which work well in single-layer, shallow-water models, are applied to the HYCOM 2.2 barotropic mode, specifically the pressure field and normal velocity component. Browning and Kreiss propose that well-posed boundary conditions for modeling fluid flow in open domains may be derived from the theory of characteristics. In the case of two independent variables x and t, characteristics are curves $x(t)$, which, if used as coordinate axes, reduce a set of coupled partial differential equations to a set of uncoupled ordinary differential equations. They occur during efforts to construct, through Taylor series expansion, the solution of a system of partial differential equations in the surrounding area of a boundary curve in x,t space along which the dependent variables and their normal derivatives are prescribed. Specifically, characteristics are curves that are unsuitable as boundary curves because the Taylor series coefficients cannot be uniquely determined from conditions prescribed along these curves.

Consider a simple hyperbolic system describing gravity wave propagation in a shallow fluid layer moving at speed U :

$$\begin{aligned} u_t + U_0 u_x + g h_x &= 0 \\ h_t + U_0 h_x + H u_x &= 0. \end{aligned} \tag{2}$$

There are two sets of characteristics in this problem; their respective slopes in the x,t plane are

$$\left(\frac{\partial x}{\partial t} \right)_{char} = U_0 \pm c, \tag{3}$$

where $c = \sqrt{gH}$ is the gravity wave phase speed. Tracking the characteristics in x,t space is equal to following gravity waves that propagate upstream and downstream through the moving fluid.

The ordinary differential equations obtained by transforming x,t derivatives in the set of partial differential equations in equation (2) into derivatives taken along characteristics are

$$\begin{aligned} u_s + c_1 h_s &= 0 \\ u_s - c_1 h_s &= 0, \end{aligned} \tag{4}$$

where $c_1 = \sqrt{g/H}$ and subscript s denote differentiation along a characteristic. After integration over s, these equations demonstrate that $u + c_1 h$ and $u - c_1 h$, respectively, are constant along the two sets of characteristics. The solution at a given point x,t can therefore be constructed by superimposing u,h combinations carried along the two characteristics intersecting at x,t. This applies to interior as well as boundary, points.

Let $cU_{\gg 0} > 0$. On the upstream or "western" boundary, two characteristics, $U_0 + c$ going west to east and $U_0 - c$ going east to west, gather information from the exterior (specified by observations or a coarse-mesh model) and from the interior, respectively. The combination of these two characteristics yields the final boundary values of u and h. If superscript o denotes values obtained from the outer coarse-mesh model or data, i signifies values from the inner, fine-mesh model, and * represents the actual boundary values, then

$$\begin{aligned} u* + c_1 h* &= u^o + c_1 h^o \\ u* - c_1 h* &= u^i - c_1 h^i. \end{aligned} \tag{5}$$

This is a system of two equations for the two desirable boundary values $u*, h*$. The solutions are

$$\begin{aligned} u* &= \frac{1}{2}\left[u^o + u^i + c_1 \left(h^o - h^i \right) \right] \\ h* &= \frac{1}{2}\left[h^o + h^i + c_1^{-1} \left(u^o - u^i \right) \right]. \end{aligned} \tag{6}$$

Boundary conditions for the case $U_0 < 0$ and the eastern boundary are analogous. If the model contains thermodynamical variables satisfying conservation laws dominated by advection processes, the method of characteristics suggests that these variables be updated by coarse mesh fields or through data at inflow points, and from within the model at outflow points.

4. Barotropic tangential velocity components are prescribed.

5. Baroclinic velocities normal to the boundary, as well as total (barotropic plus baroclinic) mass fluxes, are set. Concerning equation (5) above, prescribing the mass flux across boundaries is the most direct way to stabilize the time-mean circulation in sub-basins forced by strong inflow/outflow. Concerning form 6 above, nudging is executed by replacing a grid point value ϕ by a linear combination of ϕ and a prescribed value ϕ_b:

$$\phi_{new} = (1 - w)\phi + w\phi_b. \tag{7}$$

If nudging is performed in a finite-width sponge zone, the weight w should gradually increase from zero in the interior of the domain to a finite value ≤ 1 at the boundary. The width of the sponge zone where $w > 0$ and the rate at which w increases toward the boundary is best determined by experimentation.

The same applies to the viscosity enhancement factor and the damping factor applied to the layer thickness tendency. The damping factor should increase from near-zero at the domain

boundary to a value of 1 at the inner edge of the sponge zone. Viscosity may be increased stepwise to between 5 and 10 times its value outside the sponge zone.

6. Baroclinic tangential velocity components are nudged toward given values.

7. Other boundary conditions for the baroclinic mode are applied not only at points directly on the boundary, but in a finite-width "sponge" zone. They include interface pressure nudging, damping of the tendency term in the continuity equation, and enhanced viscosity in the momentum equations.

5.2.2 HYCOM 2.2 Interior Diapycnal Mixing

There are three diapycnal mixing algorithms included in HYCOM 2.2. When HYCOM 2.2 is run with hybrid vertical coordinates and the Kraus-Turner mixed layer model is activated, one of two interior diapycnal mixing algorithms must be chosen. The first choice is basically the implicit KPP mixing scheme with the surface boundary layer algorithm removed, and is presented as model 1 below. The second option is the explicit diapycnal mixing algorithm modified to run with hybrid vertical coordinates, presented as model 2 below. When HYCOM 2.2 is run with isopycnic vertical coordinates, the interior diapycnal mixing model is used, presented as model 3 below.

Both versions of the explicit model mix only the thermodynamical variables and scalars carried at pressure grid points. The explicit KPP-like model also mixes momentum at u and v grid points.

Model 1: Hybrid Coordinate Implicit Algorithm (Interior Mixing From KPP)

This model holds the interior ocean part of the KPP vertical mixing algorithm. Model variables are decomposed into mean (denoted by an overbar) and turbulent (denoted by a prime) components. Diapycnal diffusivities and viscosity parameterized in the ocean interior as follows:

$$\overline{w'\theta'} = -K_\theta \frac{\partial \overline{\theta}}{\partial z}, \quad \overline{w'S'} = -K_S \frac{\partial \overline{S}}{\partial z}, \quad \overline{w'\mathbf{v}'} = -K_m \frac{\partial \overline{\mathbf{v}}}{\partial z}, \tag{8}$$

where (K_θ, K_S, K_m) are the interior diffusivities of potential temperature, salinity (plus other scalars), and momentum (viscosity), respectively. Interior diffusivity/viscosity is assumed to consist of three components, which are illustrated here for potential temperature:

$$K_\theta = K_\theta^s + K_\theta^w + K_\theta^d, \tag{9}$$

where K_θ^s is the contribution of resolved shear instability, K_θ^w is the contribution of unresolved shear instability due to the background internal wave field, and K_θ^d is the contribution of double diffusion. Only the first two processes contribute to viscosity.

The contribution of shear instability is parameterized in terms of the gradient Richardson number calculated at model interfaces:

$$Ri_g = \frac{N^2}{\left(\frac{\partial \bar{u}}{\partial z}\right)^2 + \left(\frac{\partial \bar{v}}{\partial z}\right)^2},$$

where mixing is triggered when $Ri_g = Ri_0 < 0.7$. The contribution of shear instability is the same for θ diffusivity, S diffusivity, and viscosity $\left(K^s = K^s_\theta = K^s_S = K^s_m\right)$, and is given by

$$\begin{aligned} \frac{K^s}{K^0} &= 1 && Ri_g < 0 \\ \frac{K^s}{K^0} &= \left[1 - \left(\frac{Ri_g}{Ri_0}\right)^2\right]^P && 0 < Ri_g < Ri_0, \\ \frac{K^s}{K^0} &= 0 && Ri_g > Ri_0 \end{aligned} \tag{10}$$

where $K^0 = 50 \times 10^{-4}\,\mathrm{m^2 s^{-1}}$, $Ri_0 = 0.7$, and $P = 3$.

The diffusivity that results from unresolved background internal wave shear is given by

$$K^w_\theta = K^w_S = 0.1 \times 10^{-4}\,\mathrm{m^2 s^{-1}}. \tag{11}$$

Based on the Peters *et al.* (1988) analysis, Large *et al.* (1994) concluded that viscosity due to unresolved background internal waves should be an order of magnitude larger, and is thus assumed to be

$$K^w_m = 1.0 \times 10^{-4}\,\mathrm{m^2 s^{-1}}. \tag{12}$$

Regions where double diffusive processes are essential are identified using the double diffusion density ratio:

$$R_\rho = \frac{\alpha \frac{\partial \bar{\theta}}{\partial z}}{\beta \frac{\partial \bar{S}}{\partial z}}, \tag{13}$$

where α and β are thermal expansion coefficients for T and S, respectively. For salt fingering (warm, salty water overlying cold, fresh water), salinity/scalar diffusivity is shown by

$$\begin{aligned} \frac{K^d_S}{K_f} &= \left[1 - \left(\frac{R_\rho - 1}{R^0_\rho - 1}\right)^2\right]^P && 1.0 < R_\rho < R^0_\rho \\ \frac{K^d_S}{K_f} &= 0 && R_\rho \geq R^0_\rho \end{aligned} \tag{14}$$

and temperature diffusivity is given by

$$K^d_\theta = 0.7 K^d_S, \tag{15}$$

where $K_f = 10 \times 10^{-4}\,\mathrm{m^2 s^{-1}}$, $R^0_\rho = 1.9$, and $P = 3$. For diffusive convection, temperature diffusivity is given by

$$\frac{K_\theta^d}{\nu_\theta} = 0.909 \exp\left\{4.6 \exp\left[-0.54\left(R_\rho^{-1} - 1\right)\right]\right\}, \tag{16}$$

where ν_T is the molecular diffusivity of temperature. Salinity/scalar diffusivity is given by

$$\begin{aligned} K_S^d &= K_\theta^d \left(1.85 - 0.85 R_\rho^{-1}\right) R_\rho \quad 0.5 \le R_\rho \le 1 \\ K_S^d &= K_\theta^d \left(0.15 R_\rho\right) \qquad\qquad R_\rho < 0.5 \end{aligned} \tag{17}$$

Given the resulting K profiles, the vertical diffusion equation is solved using the same method as the full KPP mixing scheme. See Section 5.2.12 for more information. Although the full KPP algorithm is a semi-implicit method, only one iteration is needed for the interior diapycnal-mixing algorithm because the mixing is so weak. After mixing is completed at the pressure grid points, the viscosity coefficients are interpolated to u and v grid points, then the momentum components are mixed.

Model 2: Hybrid Coordinate Explicit Algorithm

The explicit diapycnal-mixing algorithm is based on the model of McDougall and Dewar (1998). The central dilemma in applying diapycnal mixing in an isopycnic coordinate model is to exchange potential temperature (θ), salinity (S), and mass (layer thickness, expressed as $\partial p / \partial \rho$) between model layers while preserving the isopycnic reference density in each layer, simultaneously satisfying the following conservation laws:

$$\left(\frac{\partial \theta}{\partial t}\right)_\rho + \left(\frac{\partial \rho}{\partial t}\frac{\partial p}{\partial \rho}\right)\frac{\partial \theta}{\partial p} = -\frac{\partial F_\theta}{\partial p} \tag{18}$$

$$\left(\frac{\partial S}{\partial t}\right)_\rho + \left(\frac{\partial \rho}{\partial t}\frac{\partial p}{\partial \rho}\right)\frac{\partial S}{\partial p} = -\frac{\partial F_S}{\partial p} \tag{19}$$

$$\frac{\partial}{\partial t}\left(\frac{\partial p}{\partial \rho}\right)_\rho + \frac{\partial}{\partial \rho}\left(\frac{\partial \rho}{\partial t}\frac{\partial p}{\partial \rho}\right) = 0. \tag{20}$$

The expression $(\partial \rho / \partial t)(\partial p / \partial \rho)$ is the generalized vertical velocity in isopycnic coordinates while F_θ, F_S are the diapycnal heat and salt fluxes. Although this model was derived for isopycnic layer models, its use is applicable for hybrid layer models with the actual layer densities, not their isopycnic reference densities, preserved during the mixing process.

Mutually consistent forms of the turbulent heat, salt, and mass fluxes are derived by integrating (18) through (20) across individual layers and layer interfaces. In a discrete layer model, regardless of whether it is isopycnic, layer variables such as θ, S, and momentum are assumed to be constant within layers and to have discontinuities at interfaces. Model layers will be denoted by index k increasing downward. Interfaces will also be denoted by index k, with interfaces k and $k+1$ being located at the top and bottom of layer k, respectively.
Integration of (18) and (19) across the interior of layer k gives

$$\delta p^k \left(\frac{\partial \theta^k}{\partial t} \right) = F_\theta^{k,u} - F_\theta^{k,l} \tag{21}$$

$$\delta p^k \left(\frac{\partial S^k}{\partial t} \right) = F_S^{k,u} - F_S^{k,l}, \tag{22}$$

where $\delta p^k = p^{k+1} - p^k$ is the thickness of layer k and the fluxes symbolize values infinitely close to the interfaces. The fluxes are known to have discontinuities at interfaces, but in contrast to the layer variables, they differ linearly with p in each coordinate layer consistent with the fact that $\partial\theta/\partial t$ and $\partial S/\partial t$ are p-independent. Equations (21) and (22) must satisfy the constraint that ρ remains constant while θ and S change. The required condition is

$$\frac{1}{\rho}\frac{d\rho}{dt} = \beta \frac{\partial S}{\partial t} - \alpha \frac{\partial \theta}{\partial t} = 0, \tag{23}$$

where

$$\alpha = -\frac{1}{\rho}\left(\frac{\partial \rho}{\partial \theta} \right)_{S,p} \qquad \beta = \frac{1}{\rho}\left(\frac{\partial \rho}{\partial S} \right)_{\theta,p}$$

are the thermodynamical expansion coefficients for potential T and S, respectively. Combining (21) through (23), the constraint becomes

$$\beta^k \left(F_S^{k,l} - F_S^{k,u} \right) - \alpha^k \left(F_\theta^{k,l} - F_\theta^{u} \right) = 0. \tag{24}$$

Physical perception suggests that the turbulent fluxes directly above and below an interface, while usually different, must depend linearly on the magnitude of the discontinuity of the fluxed variable at the interface. It is also logical to postulate that the proportionality factor is independent of the fluxed variable. These assumptions are expressed analytically as

$$\frac{F_\theta^{k+1,u}}{\theta^{k+1} - \theta^k} = \frac{F_S^{k+1,u}}{S^{k+1} - S^k} = m^{k+1,u} \qquad \frac{F_\theta^{k,l}}{\theta^{k+1} - \theta^k} = \frac{F_S^{k,l}}{S^{k+1} - S^k} = m^{k,l}. \tag{25}$$

Combining (24) and (25) yields

$$m^{k,u}\left[\beta^k \left(S^k - S^{k-1} \right) - \alpha^k \left(\theta^k - \theta^{k-1} \right) \right] = m^{k,l}\left[\beta^k \left(S^{k+1} - S^k \right) - \alpha^k \left(\theta^{k+1} - \theta^k \right) \right].$$

To satisfy this relation in general, $m^{k,u}$ and $m^{k,l}$ must be of the form

$$m^{k,l} = \frac{c^k}{\beta^k \left(S^k - S^{k-1} \right) - \alpha^k \left(\theta^k - \theta^{k-1} \right)} \qquad m^{k,u} = \frac{c^k}{\beta^k \left(S^{k+1} - S^k \right) - \alpha^k \left(\theta^{k+1} - \theta^k \right)}.$$

Substitution of these expressions into (25) yields

$$F_\theta^{k,u} = c^k \frac{\theta^k - \theta^{k-1}}{\beta^k \left(S^k - S^{k-1}\right) - \alpha^k \left(\theta^k - \theta^{k-1}\right)} \qquad F_\theta^{k,l} = c^k \frac{\theta^{k+1} - \theta^k}{\beta^k \left(S^{k+1} - S^k\right) - \alpha^k \left(\theta^{k+1} - \theta^k\right)} \quad (26)$$

in addition to analogous expressions for $F_S^{k,u}$ and $F_S^{k,l}$ involving the same constant c^k.

To resolve the proportionality factor c^k in (26), first notice that the denominators in (26) represent a relative jump $\partial\rho/\rho$, and the fluxes in (26) are finite-difference analogs of expressions of the general form $F_Q = c\rho\partial Q/\partial\rho$ for variable Q. If the turbulent fluxes of Q are also represented in the context of K theory, $F_Q = -K\partial Q/\partial p$. Equating these two expressions for F_Q results in

$$c^k = -\frac{K}{\rho}\frac{\partial\rho}{\partial p} = -K\frac{N^2}{g}.$$

The fluxes then become

$$\begin{aligned} F_\theta^{k,u} &= -G^{k,u}\left(\theta^k - \theta^{k-1}\right) \quad F_\theta^{k,l} = -G^{k,l}\left(\theta^{k+1} - \theta^k\right) \\ F_S^{k,u} &= -G^{k,u}\left(S^k - S^{k-1}\right) \quad F_S^{k,l} = -G^{k,l}\left(S^{k+1} - S^k\right) \end{aligned} \quad (27)$$

where

$$G^{k,u} \equiv \frac{\left(K\frac{N^2}{g}\right)^k}{\beta^k \left(S^k - S^{k-1}\right) - \alpha^k \left(\theta^k - \theta^{k-1}\right)} \quad G^{k,l} \equiv \frac{\left(K\frac{N^2}{g}\right)^k}{\beta^k \left(S^{k+1} - S^k\right) - \alpha^k \left(\theta^{k+1} - \theta^k\right)}. \quad (28)$$

Due to the physical uncertainties surrounding the magnitude of the exchange coefficient K, significant freedom exists in assessing the term $\partial\rho/\partial z$ in N^2. One understandable choice was selected for use:

$$\frac{\Delta\rho}{\Delta p} = \frac{\rho^{k+1} - \rho^{k-1}}{2\left(p^{k+1} - p^k\right)}.$$

Having derived expressions for the vertical fluxes, the expressions used to advance layer thickness (mass flux), θ, and S in time must now be derived. Considering the mass flux first, the expression $\left(d\rho/dt\right)\left(\partial p/\partial\rho\right)$ is estimated by converting (18) and (19) to flux form and integrating them across interfaces. This step removes the ambiguity that $\partial\theta/\partial t$ and $\partial S/\partial t$ are indeterminate at interfaces. The flux equations for heat and salt, obtained in the typical method by combining the advective forms of the equations with (20), are

$$\frac{\partial}{\partial t}\left(\theta\frac{\partial p}{\partial\rho}\right) + \frac{\partial}{\partial\rho}\left(\frac{d\rho}{dt}\frac{\partial p}{\partial\rho}\theta\right) = -\frac{\partial F_\theta}{\partial\rho} \quad (29)$$

and

$$\frac{\partial}{\partial t}\left(S\frac{\partial p}{\partial\rho}\right) + \frac{\partial}{\partial\rho}\left(\frac{d\rho}{dt}\frac{\partial p}{\partial\rho}S\right) = -\frac{\partial F_S}{\partial\rho}. \quad (30)$$

Integrating these equations over an infinitely thin slab straddling layer interface $k+1$ at the base of layer k (which will be represented by $k+1/2$ to avoid confusion with the layer

index), the time tendency terms drop out due to the compactness of $\partial p / \partial \rho$, producing

$$\left(\frac{d\rho}{dt} \frac{\partial p}{\partial \rho} \right)^{k+1/2} = -\frac{F_\theta^{k+1,u} - F_\theta^{k,l}}{\theta^{k+1} - \theta^k} \tag{31}$$

and

$$\left(\frac{d\rho}{dt} \frac{\partial p}{\partial \rho} \right)^{k+1/2} = -\frac{F_S^{k+1,u} - F_S^{k,l}}{S^{k+1} - S^k}. \tag{32}$$

By virtue of (27) and (28), the two previous expressions reduce to

$$\left(\frac{d\rho}{dt} \frac{\partial p}{\partial \rho} \right)^{k+1/2} = G^{k,l} - G^{k+1,u}. \tag{33}$$

This expression gives the mass flux at interface $k+1/2$ which, in conjunction with the heat and salt fluxes given by (27), forms a constant set that can be used to solve the conservation equations for layer thickness, potential temperature, and salinity.

Inserting (33) into (20) and integration over layer k yields

$$\frac{\partial}{\partial t}\left(\delta p\right)^k + \left(G^{k,u} + G^{k,l}\right) - G^{k-1,l} - G^{k+1,u} = 0. \tag{34}$$

The G terms have been grouped in a manner that imitates the second derivative of G with respect to k, illustrating that diapycnal mixing tends to transport mass from thick layers (N^2 small) to thin layers (N^2 large). The prognostic equations for heat and salt are acquired from the flux form of the conservation equations (29) and (30), which are integrated over the interior portion of layer k to give

$$\frac{\partial}{\partial t}\left[\theta^k \left(\partial p\right)^k \right] + \left(G^{k+1,u} + G^{k-1,l}\right)\theta^k - G^{k,u}\theta^{k-1} - G^{k,l}\theta^{k+1} = 0 \tag{35}$$

and

$$\frac{\partial}{\partial t}\left[S^k \left(\partial p\right)^k \right] + \left(G^{k+1,u} + G^{k-1,l}\right)S^k - G^{k,u}S^{k-1} - G^{k,l}S^{k+1} = 0. \tag{36}$$

The G terms have once more been arranged to accentuate the diffusive nature of the equations. Equations (34) through (36) are integrated subject to the boundary conditions

$$\left(\frac{d\rho}{dt} \frac{\partial p}{\partial \rho} \right)^{ktop-1/2} = F_\theta^{ktop,u} = F_S^{ktop,u} = 0 \qquad \left(\frac{d\rho}{dt} \frac{\partial p}{\partial \rho} \right)^{kbot+1/2} = F_\theta^{kbot,l} = F_S^{kbot,l} = 0.$$

Model 3: Isopycnic Coordinate Explicit Algorithm

When HYCOM 2.2 is run with isopycnic coordinates, a diapycnal mixing model is used. The interior diapycnal mixing algorithm is basically model 2 above, but with only salinity and layer thickness advancing in time and temperature diagnosed from the equation of state.

5.2.3 HYCOM 2.2 Equation of State

The equation of state rooted in HYCOM 2.2 is the approximation to the UNESCO equation of state described by Brydon *et al*. (2001). At a given reference pressure level p, the density in sigma units is specified by a seven term polynomial function cubic in potential temperature θ and linear in salinity S:

$$\sigma(\theta, S, p) = C_1(p) + C_2(p)\theta + C_3(p)S + C_4(p)\theta^2 + C_5(p)S\theta + C_6(p)\theta^3 + C_7(p)S\theta^2.$$

One benefit of this straightforward polynomial representation is that it can be inverted to calculate $\theta(\sigma, S, p)$ and $S(\sigma, \theta, p)$.

Two sets of coefficients are supplied in HYCOM 2.2 for reference pressures of 0, 20 Mpa. The sigma values calculated with these coefficients are referred to as σ_0 and σ_2, respectively. If the user chooses σ_0 to represent model vertical coordinates, the density structure will be characterized reasonably well in the upper ocean. In the deep ocean, however, there will be regions where σ_0 does not increase monotonically with depth, triggering model vertical coordinate interfaces to fold over. The user must consider this issue in selecting the proper set of density coordinates for model simulations.

Cabbeling is of no consequence when HYCOM 2.2 is run with isopycnic coordinates because there is no penetrating shortwave radiation and because salinity alone is advected and diffused within isopycnic layers. When HYCOM 2.2 is run with hybrid vertical coordinates, however, cabbeling becomes an issue since T and S are always mixed in the vertical and shortwave radiation does penetrate into the isopycnic coordinate domain. Horizontal advection and diffusion of T and S, and T and S fluxes across vertical coordinates relocated by the hybrid coordinate algorithm, could also play a role in cabbeling. When HYCOM 2.2 is run with hybrid vertical coordinates, reliance is placed on the hybrid vertical coordinate adjustment algorithm to restore and uphold isopycnic conditions.

Cabbeling effects can be minimized in two ways. HYCOM 2.2 has the option of horizontally advecting and diffusing either S and density or T and density, rather than T and S. HYCOM 2.2 also offers the option to flux salinity and density or temperature and density across vertical coordinates repositioned by the hybrid coordinate algorithm. For further information on the troubles that can be caused by cabbeling, see Sections 5.2.1 and 5.2.6.

HYCOM 2.2 is equipped with the algorithm of Sun *et al.* (1999), which accounts for the dynamical impact of thermobaric compressibility, or thermobaricity. And even though isopycnic target densities are potential densities referenced to a prescribed pressure level, the default option in HYCOM 2.2 for determining water column stability across a model interface is to use potential density that is locally referenced to the pressure of that interface.

5.2.4 Synthetic Floats, Drifters, and Moorings

HYCOM 2.2 has features for deploying and tracking synthetic floats and drifters, as well as seeding the model with synthetic moorings, during model run time. There are currently four types of floats supported:

1) **3D Lagrangian**, in which the floats are vertically advected by the diagnosed vertical velocity;
2) **isopycnic floats**, which remain at the depth of a specified density surface;
3) **isobaric**, which remain at a constant pressure depth; and
4) **stationary (synthetic mooring)**.

Isobaric floats can double as surface drifters by releasing them in the top model layer. Stationary floats can be stacked vertically to create a synthetic mooring that samples model fields at very high frequency. Dynamical and thermodynamical water properties are optionally interpolated to the position of each float. Time series of float location and depth, as well as the interpolated water properties, are archived for further analysis. For stationary floats, velocity components $u, v, and\ w$ are output rather than float position and depth.

Non-stationary floats and drifters are preferentially tracked during a model run. This is because the short sampling time interval required to correctly advect the float (in the presence of energetic diurnal, inertial, and synoptic variability) requires large model fields to be archived far too often for off-line execution to be possible. Even when monthly forcing drives the model, velocity fields should be sampled every one or two hours to accurately advect the float.

The horizontal and temporal interpolation schemes used to advect the floats are adapted from algorithms developed by Garraffo *et al*., (2001a; 2001b). Horizontal interpolation is achieved using a sixteen-point grid box surrounding the float. Two-dimensional polynomial interpolation is performed if enough acceptable grid points are available; otherwise, bilinear interpolation from the four grid points surrounding the float is done. Temporal interpolation is achieved using a fourth-order Runga-Kutta algorithm (described below).

Vertical Interpolation Algorithm

Horizontal interpolation is performed for both layer and interface variables and is fairly straightforward since values from the model layer including the float at the surrounding grid points are handled as input. For interface variables, however, it is first essential to vertically interpolate at the surrounding grid points as follows: Given a float at pressure p_d within model layer k, which is bounded by upper interface k and lower interface $k+1$, the quotient q_d is first determined using

$$q_d = \frac{p^{k+1} - p_d}{p^{k+1} - p^k}. \tag{37}$$

The virtual pressure surface $\hat{p}$ is identified as the surface within model layer k where q_d is constant; i.e., where

$$\hat{p} = q_d p^k + (1 - q_d) p^{k+1} \quad . \tag{38}$$

At each input grid point i, j, interface variables are vertically interpolated to the pressure depth $\hat{p}^{i,j} = q_d p^{i,j,k} + (1 - q_d) p^{i,j,k+1}$. The vertically interpolated quantity of interface variable A at grid point i, j then becomes

$$A^{i,j} = q_d A^{i,j,k} + (1 - q_d) A^{i,j,k+1}. \quad (39)$$

The resulting input array $A^{i,j}$ is then horizontally interpolated to the float location.

Horizontal Interpolation Algorithm

The first step of the horizontal interpolation algorithm is to select the sixteen-point input box. The model grid point i_0, j_0 located immediately to the southwest of the float location is identified. The sixteen-point grid box then consists of points $i_0 - 1$ to $i_0 + 2$ and $j_0 - 1$ to $j_0 + 2$. If a sufficient number of water grid points are available, a 2D polynomial surface is fit to the data for interpolation. If too few water grid points are accessible, a bilinear scheme using the four grid points i_0 to $i_0 + 1$, j_0 to $j_0 + 1$ is called upon to do the interpolation. If none of these four points are water points, the float is assumed to have run aground. All model variables are interpolated from their native grid (p, u, or v). For example, for zonal velocity u, sixteen u grid points are chosen to perform the interpolation. Besides land points, others in the 16-point box are masked from the interpolation if the layer containing the float is a zero thickness layer at the bottom.

HYCOM 2.2 horizontal interpolation uses the 2D polynomial fit of Mariano and Brown (1992), who used it for the large-scale trend surface fit in their parameter matrix objective analysis algorithm. The minimum number of surrounding water grid points required for polynomial interpolation is presently set at nine. This was chosen so that HYCOM 2.2 could have a more accurate polynomial interpolation whenever possible while still having enough grid points to resolve the 2D structure of the field.

Vertical Adjustment of Floats

Three-dimensional Lagrangian floats are vertically advected by the diagnosed vertical velocity interpolated to the float using the same Runga-Kutta method (described below) used for horizontal velocity components. Isobaric floats are simply kept at their initial pressure depth. The pressure depth of isopycnic floats is set to the depth of the initial reference isopycnic surface as follows: Calculate the pressure depth of the reference isopycnic surface at each of the selected pressure grid points surrounding the float, which involves identifying the two model layers whose densities bracket the isopycnic reference density and then linearly and vertically interpolating between the middle pressure depths of each layer. The scalar estimates of the isopycnic surface depth at the surrounding grid points are then horizontally interpolated to the float location.

Runga-Kutta Time Interpolation

The fourth-order Runga-Kutta time interpolation algorithm is a widely available procedure requiring estimates of model velocity at the present time and at two earlier times. If the user selects the 3D Lagrangian float option, time interpolation is performed on all three velocity components. Otherwise, it is done only on the horizontal velocity components. Preferably, the time interval separating each of the velocity component fields in the interpolation should be between one and two hours. In HYCOM 2.2, the user selects the time interval as an integer number of baroclinic time steps and is warned if the time interval does not fall between one and two hours.

When the float is advected at each model time, the velocity component fields at the earliest of the three times represent the previous float advection time. When the float update subroutine is called at the intermediate time, the only action taken is to store the intermediate fields required to advect the float at the next advection time. Thus, the floats are updated every *second* time the subroutine is called. In addition to horizontal velocity components, previous fields of w_l, $\partial p / \partial x$, and $\partial p / \partial y$ are stored so that the time interpolation of vertical velocity can be performed for 3D Lagrangian floats.

Float position is saved as longitude (θ) and latitude (φ). Because the HYCOM 2.2 horizontal mesh can be any orthogonal curvilinear grid, the following formula is used to advect floats:

$$\begin{aligned} \frac{d\theta}{dt} &= u\frac{d\theta}{dx} + v\frac{d\theta}{dy} \\ \frac{d\varphi}{dt} &= u\frac{d\varphi}{dx} + v\frac{d\varphi}{dy} \end{aligned} \quad . \tag{40}$$

The first terms on the right side (40) are computed on *u* grid points while the second terms are computed on *v* grid points. These terms are then interpolated to the float locations from the surrounding *u* and *v* grid points, respectively.

Turbulent horizontal velocity can optionally be added to the deterministic horizontal velocity used to advect the floats. The parameter settings for this option in **blkdat.input** (`tbvar` and `tdecri`) are described in Griffa (1996).

Implementation and Initialization of the Float Algorithm

The user must first choose the velocity sampling time interval for the Runga-Kutta time interpolation. The model baroclinic time interval Δt_{bc} is set prior to beginning a model run. The velocity sampling time interval Δt_{vel} should be between one and two hours. When HYCOM 2.2 is run at low horizontal resolution, Δt_{bc} is usually set to order one hour. In this case, the velocity sampling time interval is set to $\Delta t_{vel} = 2\Delta t_{bc}$. At higher resolutions, the velocity-sampling interval is set at $\Delta t_{vel} = n_{vel}\Delta t_{bc}$, where n_{vel} is made a value that insures that Δt_{vel} is between one and two hours.

With this selection made, the float position is updated every $2n_{vel}$ baroclinic time steps using velocity fields at the present time t and at the two earlier times $t - n_{vel}\Delta t_{bc}$ and

$t - 2n_{vel}\Delta t_{bc}$. The user also selects the time interval between which float information is output for further analysis. The options are to output each time the drifter is updated, $2n_{vel}$ baroclinic time steps, or to output it once every n_{out} days. For the float code to work correctly, Δt_{bc} must be selected so that there are an integer number (n_{bc}) of baroclinic time steps per day, while n_{vel} must be selected so that $n_{bc} / 2n_{vel}$ is an integer.

The user must choose the float type (Lagrangian, isopycnic, isobaric, or mooring) before starting the model. Float deployment information is given in the file **float.inp,** which includes four columns of data. The first column details the float/drifter/mooring type, with 0 for three-dimensional Lagrangian, 1 for isobaric, 2 for isopycnic, and 3 for mooring. The second column represents deployment time in days from the start of the model run. If the value is zero, the float is released instantly. The third column is termination time in days from the beginning of the simulation. If the value is zero, the float lasts until it runs aground. The fourth and fifth columns represent the initial longitude and latitude. Information in column six varies by float type. For Lagrangian floats, isobaric floats/drifters, and moorings, it contains the initial depth in meters. For isopycnic floats, it holds the reference isopycnic surface value.

Preliminary Calculations

The float subroutine is called every n_{vel} time steps, and float advection is performed every $2n_{vel}$ time steps. At intermediate time steps where the floats are not advected, the subroutine stores only old velocity fields and other fields required for the Runga-Kutta time interpolation. Interpolation of water properties to the float location is executed only every n_{out} time steps when float information is stored for further analysis.

Before the update loop for each individual float, many 3D fields are computed, if necessary. If the floats are Lagrangian, fields vital to estimating vertical velocity are calculated. Fields of w_l are estimated at model interfaces on pressure grid points. For the later interpolation of w_a to the float location, $\partial p / \partial x$ fields are determined on u grid points and $\partial p / \partial y$ fields are estimated on v grid points. For all floats, the relative vorticity field is determined on p grid points at the times when float information is output.

Identification of the Model Layer Containing the Float

Following the preliminary calculations, the main float loop is executed once for each float. The first step is to determine the model layer that contains the float. This is initially assumed to be the model layer k that contained the float during the previous time step. This is tested by interpolating the pressure depths of interfaces k and $k+1$ to the location of the float and determining if the previous pressure depth of the float falls between those limits. If the float is no longer within this layer (or if the float has just been released), a special algorithm is executed to determine the model layer containing the float. Moving down from the surface, interface pressure depths are interpolated to the float location until the new layer containing the float is identified. Once the model layer is identified, the ratio q_d is calculated using (37).

Float Advection

Once the model layer is identified, the Runga-Kutta time interpolation algorithm is executed to move the float horizontally. For 3D Lagrangian floats, the Runga-Kutta algorithm is also performed to move the float vertically. The depth of isopycnic floats is set at the depth of the reference isopycnic surface. For isobaric floats, the float is not moved vertically. For moorings, of course, neither horizontal nor vertical advection is performed.

Interpolation of Water Properties to the Float Location

Many water properties are interpolated to the floats and output for further analysis. The variables are longitude, latitude, float depth, water depth, temperature, salinity, density, relative vorticity, potential vorticity, vertical viscosity, and vertical temperature diffusivity. For moorings, water depth is replaced with vertical velocity and viscosity and temperature diffusivity are replaced with horizontal velocity components.

5.2.5 HYCOM 2.2 Potential Temperature Balance

In HYCOM 2.2, mixed layer thickness and temperature are fully analyzed and mixed layer temperature balance terms are not calculated prognostically. One can only use the prognostic layer potential temperature equations to diagnose the temperature balance within individual model layers. It is possible to estimate mixed layer temperature balance terms by vertically integrating terms of the layer potential temperature balance from the surface to the diagnosed depth of the MLB. However, using hybrid vertical coordinates leads to obstacles in the interpretation of vertical advection and diffusion's influence on mixed layer temperature.

Potential Temperature Balance within Individual Model Layers

HYCOM 2.2 equations are created in x, y, s, t space, where s is a generalized vertical coordinate. In this structure, the vertically integrated HYCOM 2.2 equation for the conservation of potential temperature within model layer k (Bleck, 2002) is

$$\begin{aligned}\frac{\partial}{\partial t_s}\left(\theta\Delta p\right)_k = &-\nabla_s\cdot\left(\mathbf{v}\theta\Delta p\right)_k - \left(\dot{s}\theta\frac{\partial p}{\partial s}\right)_{k+1} + \left(\dot{s}\theta\frac{\partial p}{\partial s}\right)_k + \theta_k\nabla_s\cdot\left[K_{H\Delta p}\nabla\left(\Delta p_k\right)\right] \\ &+\nabla_s\cdot\left(K_{H\vartheta}\Delta p_k\nabla_s\theta_k\right) - \left(K_{V\theta}\frac{\partial\theta_k}{\partial s}\right)_{k+1} + \left(K_{V\theta}\frac{\partial\theta_k}{\partial s}\right)_k - \left(Q_k - Q_{k+1}\right),\end{aligned} \tag{41}$$

where $\Delta p_k = p_{k+1} - p_k$ is layer thickness, $\dot{s}\theta\partial p/\partial s$ is the temperature flux across an s interface related to mass flux, $\dot{s}\partial p/\partial s$ and $K_{H\vartheta}, K_{V\theta}$ are horizontal and vertical diffusion coefficients due to subgrid-scale velocity fluctuations, $K_{H\Delta p}$ is the horizontal thickness diffusion coefficient, and Q represents diabatic source terms. Multiplication of (41) by c_p/g gives the heat balance in layer k. To derive the potential temperature balance, (41) is rearranged as

$$
\begin{aligned}
(\Delta p)_k \frac{\partial \theta_k}{\partial t_s} = & -\theta_k \frac{\partial}{\partial t}(\Delta p)_k - (\mathbf{v}\Delta p)_k \cdot \nabla_s \theta_k - \theta_k \nabla_s \cdot (\mathbf{v}\Delta p)_k - \left(\dot{s}\theta \frac{\partial p}{\partial s}\right)_{k+1} + \left(\dot{s}\theta \frac{\partial p}{\partial s}\right)_k \\
& + \nabla_s \cdot \left(K_{H\vartheta} \Delta p_k \nabla_s \theta_k\right) + \theta_k \nabla_s \cdot \left[K_{H\Delta p} \nabla (\Delta p_k)\right] \\
& - \left(K_{V\theta} \frac{\partial \theta_k}{\partial s}\right)_{k+1} + \left(K_{V\theta} \frac{\partial \theta_k}{\partial s}\right)_k - (Q_k - Q_{k+1}),
\end{aligned}
\tag{42}
$$

then the first term on the right side of (42) is calculated using the vertically integrated HYCOM 2.2 continuity (thickness tendency) equation for layer k (Bleck, 2002):

$$
\frac{\partial}{\partial t_s} \Delta p_k = -\nabla_s \cdot (\mathbf{v}\Delta p)_k - \left(\dot{s} \frac{\partial p}{\partial s}\right)_{k+1} + \left(\dot{s} \frac{\partial p}{\partial s}\right)_k + \nabla_s \cdot \left(K_{H\Delta p} \Delta p_k\right). \tag{43}
$$

Multiplying (43) by θ_k and substituting into (42) produces the potential temperature balance within layer *k*:

$$
\begin{aligned}
(\Delta p)_k \frac{\partial \theta_k}{\partial t_s} = & -(\mathbf{v}\Delta p)_k \cdot \nabla_s \theta_k \\
& - \left(\dot{s}\theta \frac{\partial p}{\partial s}\right)_{k+1} + \left(\dot{s}\theta \frac{\partial p}{\partial s}\right)_k + \theta_k \left(\dot{s} \frac{\partial p}{\partial s}\right)_{k+1} - \theta_k \left(\dot{s} \frac{\partial p}{\partial s}\right)_k \\
& + \nabla_s \cdot \left(K_{H\vartheta} \Delta p_k \nabla_s \theta_k\right) \\
& - \left(K_{V\theta} \frac{\partial \theta}{\partial s}\right)_{k+1} + \left(K_{V\theta} \frac{\partial \theta}{\partial s}\right)_k \\
& + (Q_k - Q_{k+1}).
\end{aligned}
\tag{44}
$$

Potential temperature is adapted by horizontal advection (top line, right side); temperature flux associated with mass flux across model interfaces (second line); horizontal diffusion (third line); vertical diffusion (fourth line); and vertical heat flux divergence (bottom line). It is simple to estimate the role of these terms during a model run. However, interpretation of the contributions of vertical advection diffusion to temperature change in each layer [lines 2 and 4 of (44)] is not clear-cut in the hybrid coordinate system. Subroutine *hybgen* computes the layer temperature change due to mass flux across layer interfaces [line 2 in (44)] while vertical mixing routines **mx*.f** [line 4 in (44)] compute temperature change as a result of vertical diffusion across layer interfaces. Together these terms demonstrate the total influence of vertical temperature flux on layer potential temperature.

The interpretation of how these terms [lines 2 and 4 of (44)] contribute to the total influence of vertical temperature flux varies between a model layer in the surface *p* coordinate domain and a model layer in the interior isopycnic domain. In the surface *p* domain, the vertical flux terms in line 2 of (44) justify the contribution of vertical advection of temperature across layer interfaces. Advection can affect temperature within each layer since this advection creates vertical fluxes across interfaces. Vertical advection in the *p* domain is accounted for as follows:

First, the continuity equation solver modifies layer thickness in response to horizontal divergence patterns and thickness diffusion (e.g., in response to local vertical velocity), but does not change temperature within each model layer. The hybrid coordinate generator then pushes model interfaces back to their initial depths. The resulting mass fluxes across interfaces ($\dot{s} \neq 0$) generate the layer temperature changes caused by vertical advection. For example, consider the surface *p* layers at the equator where equatorial upwelling is present. At each time step, the continuity equation shifts the interfaces upward in the water column without altering the potential temperature in the layers. The hybrid coordinate generator then transfers the interfaces back down to their previous depths and cools the temperature within each layer due to the resulting upward fluxes ($\dot{s} > 0$).

In the isopycnic interior, the hybrid coordinate generator does not relocate model interfaces to their original depths, but only shifts layers vertically if necessary to restore isopycnic conditions. Interior vertical mixing and penetrating shortwave radiation are the only processes in the isopycnic interior that propel layer density away from the target isopycnic values and cause ***hybgen*** to shift model layers. Because shortwave radiation is typically insignificant in the isopycnic interior, layer temperature changes produced by ***hybgen*** take place only in response to interior diapycnal mixing. Therefore, the combined contribution of lines 2 and 4 of (44) accounts for the layer temperature change due to vertical diffusion alone. Vertical advection does not affect layer temperature in the isopycnic interior.

Mixed Layer Temperature Balance

Mixed layer temperature balance calculations are not clear-cut in HYCOM 2.2 because the model (when run with one of the Reynolds stress vertical mixing models) does not recognize the existence of the mixed layer. Temperature and layer thickness are prognostic variables, but temperature and mixed layer thickness are diagnosed after each time step. HYCOM 2.2 does not calculate the actual mixed layer temperature balance. Mixed layer temperature balance terms may be estimated by integrating the terms of (44) from the surface down to the determined depth of the MLB. When this is finished, interpretation of the contributions of the individual terms of (44) is simple with the exception of the terms in lines 2 and 4. This is because the terms in line 2 (***hybgen***) explain different physical processes in the surface *p* domain versus the interior isopycnic domain described above. The best approach is to calculate the combined temperature change produced by the terms in lines 2 and 4 (***hybgen*** plus the vertical mixing model) and refer to this as one term in the mixed layer temperature balance that characterizes the total influence of vertical temperature flux (the combined contributions of advection and diffusion). In this case, the contribution of the terms in lines 2 and 4 may be estimated as a residual from the total contribution of all other terms.

5.2.6 HYCOM 2.2 Generalized Vertical Coordinates

Hybrid vertical coordinates allow model layers below the mixed layer (layer 1) with reference isopycnic densities smaller than the mixed layer density to supply vertical resolution within the mixed layer. In HYCOM 2.2, a minimum thickness δ_k is chosen separately for each model layer *k*. Layers with zero thickness at the MLB must keep at least the specified minimum

thickness in HYCOM 2.2. Layers that are significantly less dense than the mixed layer all year remain fixed and level near the ocean surface, giving a permanent *z*-coordinate domain. Vertical coordinates are non-isopycnic from the surface down through the first model layer, which has an isopycnic reference density exceeding the mixed layer density. In regions where the mixed layer thickness and density have significant seasonal cycles, one or more model layers that are isopycnic during summer and non-isopycnic in winter usually exist below the *z*-coordinate domain. The winter vertical coordinate profile in these regions generally consists of level *z*-coordinates close to the surface that shift first to non-isopycnic, non-level coordinates and then to isopycnic coordinates.

The vertical coordinate adjustment algorithm is from Bleck and Boudra (1981), who developed a quasi-isopycnic model where layers are isopycnic only if interface separation was large enough. Wherever interface separation became too small during a given time step, the vertical coordinate was repositioned to maintain minimum thickness. The fluxes of properties across the relocated interfaces then allow these layers to become non-isopycnic.

This algorithm was later revised to include an isentropic atmospheric model to stop model layers from becoming zero thickness layers at the surface. A cushion function (see below) was employed to provide a smooth transition between the isentropic coordinate interior atmosphere and a sigma coordinate domain adjacent to the surface (Bleck and Benjamin, 1993). This version of the algorithm was turned upside-down and put in the HYCOM 2.2 for smooth transition between the isopycnic ocean interior and a *z* coordinate domain close to the ocean surface. The algorithm has evolved to provide an efficient transition to sigma coordinates in shallow water areas and level coordinates in very shallow areas.

Specification of Minimum Layer Thickness

In order for HYCOM 2.2 to be used as a coastal ocean model, the procedure for determining the minimum layer thickness was altered so that the grid generator yields a fluid transition to σ coordinates in shallow water, then back to *p* coordinates in very shallow water. The vertical grid structure is now controlled by specifying the minimum thickness permitted for each layer *k* as follows:

$$\delta_k = \begin{Bmatrix} \max\left[\delta_{2k}, \min\left(\delta_{1k}, D/N_s\right)\right] & \left(1 \le k \le N_s\right) \\ \delta_{1k} & \left(k > N_s\right) \end{Bmatrix}, \tag{45}$$

where

$$\delta_{1k} = \begin{Bmatrix} \min\left(\delta_1^{\max}, \delta_{11}\alpha_1^{(k-1)}\right) & \left(1 \le k \le N_h\right) \\ \delta_{\text{int}} & \left(k > N_h\right) \end{Bmatrix}, \tag{46}$$

$$\delta_{2k} = \min\left(\delta_2^{\max}, \delta_{21}\alpha_2^{(k-1)}\right), \tag{47}$$

where D is water depth, N_s is the number of layers beneath the surface permitted to transition to sigma coordinates, and N_h is the number of layers beneath the surface that are hybrid layers, since minimum thickness requirements have been enforced. The value of N_s is

pre-selected while the value of N_h is computed at each grid point during every time step. The minimum thickness δ_{1k} given by (46) governs the *p*-isopycnic coordinate transition in the open ocean. The minimum thickness δ_{2k} given by (47) prevents layers from becoming too thin in very shallow water. In order to estimate δ_{1k} and δ_{2k}, the minimum thicknesses of layer 1 (δ_{11} and δ_{21}), the largest allowable minimum thicknesses ($\delta_1^{\max}$ and $\delta_2^{\max}$), and the expansion/contraction factors (α_1 and α_2) must be identified, the latter usually selected to be greater than one to provide the highest resolution close to the surface. Estimation of δ_{1k} requires pre-selection of an interior minimum thickness δ_{int}, which is generally much smaller than the minimum thicknesses of the layers above so that sharp pycnoclines can evolve in the isopycnic interior. In these equations, δ_{int} is set to 1 m. The two minimum thicknesses estimated in (46) and (47) are then used in (45) to compute the global minimum thickness δ_k.

If N_s is zero, then $\delta_k = \delta_{1k}$ everywhere and the coastal transition is not applied. If N_s is nonzero, the coastal transition is applied in model layers $1 \le k \le N_s$. In these layers, the transition to sigma coordinates starts where the water depth becomes shallow enough to make $\delta_s < \delta_{1k}$. The transition back to level coordinates, preventing layers from becoming too thin, occurs where the water depth is shallow enough to make $\delta_s \le \delta_{2k}$.

During each time step, the model equations are solved under the assumption that HYCOM 2.2 is a pure Lagrangian coordinate layer model. The grid generator then moves vertical interfaces to restore isopycnic conditions to the greatest extent possible while enforcing the minimum thickness requirements given by (45). A "cushion" function from Bleck (2002) is used to insure a gradual numerical transition between the isopycnic interior and the domain of fixed coordinate layers at their specified minimum thicknesses. At all grid points, the top layer is always restored to the constant thickness δ_1 from (45) by the grid generator.

Basin-scale and global HYCOM 2.2 runs generally follow the open-ocean convention of conserving isopycnic conditions throughout most of the water column. This requires that the near surface *p* coordinate layer be thin in the stratified tropical and subtropical ocean where the winter mixed layer is not very deep. This configuration does not give good surface-to-bottom resolution over the middle and outer continental shelf, and the bottom boundary layer usually cannot be resolved. Included in HYCOM 2.2 is the capability for adding more layers at the surface to nested models for the purpose of expanding the thickness of the offshore *p* coordinate layer and giving additional interfaces to deform into σ coordinates over the shelf.

HYCOM 2.2 is a purely Lagrangian layer model and includes a hybrid coordinate generator (Bleck, 2002; Halliwell, 2005) for relocating layer interfaces at the end of each baroclinic time step. HYCOM 2.2 is classified as a Lagrangian Vertical Direction (LVD) model where the continuity (thickness tendency) equation is solved prognostically throughout the domain, using the Arbitrary Lagrangian-Eulerian (ALE) technique for re-mapping the vertical coordinate and maintaining different coordinate types in the domain (Adcroft and Hallberg, 2005). This is different from Eulerian Vertical Direction (EVD) models with fixed *z* and σ coordinates, which use the continuity equation to diagnose vertical velocity. Level coordinate models can fail in areas of steep topography due to the required discretization of the bottom depth unless partial or shaved cells are used (Adcroft *et al*., 1997; Pacanowski and Gnanadesikan, 1998; Ezer and Mellor, 2004). When level coordinates are used in HYCOM 2.2, intersection of model layers

with the bottom is handled numerically so that bottom depth discretization is not an issue.

Grid Generator Implementation

The HYCOM 2.2 vertical coordinate adjustment in the open ocean works as follows:

$$\alpha_k > \alpha_{k+1},$$

where α is specific volume. Interfaces are labeled so that interface k is the upper interface of layer k. Consider three successive isopycnic layers labeled 0, 1, and 2 in a stratified water column. Suppose that α_1 varies from its isopycnic reference value $\hat{\alpha}_1$. To restore isopycnic conditions, the water column must be re-discretized to conserve the total height of the column, given by the integral $\int \alpha dp$, while changing α_1 to $\hat{\alpha}_1$. Conservation of the integral requires that one or more layer interfaces be transferred to other pressure levels. In modifying the density in layer 1, one interface is repositioned in the spirit of the "donor cell" transport scheme. If layer 1 is too dense ($\alpha_1 < \hat{\alpha}_1$), the upper interface is moved upward to shift less-dense water from layer 0 to layer 1. If layer 1 is too light ($\alpha_1 > \hat{\alpha}_1$), the lower interface is relocated downward to move denser water from layer 2 to layer 1. This technique does not work for the model layer in contact with the bottom if it is too light. A special algorithm is included to handle this case by extruding water into the layer above.

Case 1: $\alpha_1 > \hat{\alpha}_1$

The upper interface is moved and mass is exchanged between layers 0 and 1. Conservation of $\int \alpha dp$ requires

$$\alpha_0(p_1 - p_0) + \alpha_1(p_2 - p_1) = \alpha_0(\hat{p}_1 - p_0) + \hat{\alpha}_1(p_2 - \hat{p}_1), \tag{48}$$

where $\hat{p}_1$ is the pressure of the upper interface after re-discretization. Solving (48) for $\hat{p}_1$ gives the expression

$$\hat{p}_1 = \frac{p_1(\alpha_0 - \alpha_1) + p_2(\alpha_1 - \hat{\alpha}_1)}{\alpha_0 - \hat{\alpha}_1}. \tag{49}$$

Given that the weight assigned to p_2 is negative, (49) will not necessarily produce a solution $\hat{p}_1 > p_0$ for large differences between α_1 and $\hat{\alpha}_1$. To keep the minimum thickness of layer 0, p_1 is replaced with

$$\tilde{p}_1 = \max(\hat{p}_1, p_0 + \Delta_0), \tag{50}$$

where Δ_0 is a specified minimum layer thickness. Certainly, changing the interface to $\tilde{p}$ instead of $\hat{p}$ no longer allows isopycnic conditions to be restored.

Following Bleck and Benjamin (1993), the user chooses the absolute minimum thickness δ_0. The actual minimum thickness Δ_0 is then computed to assure a smooth transition between the isopycnic and non-isopycnic domains. The function Δ_0 is controlled by a continuously differentiable "cushion" function, which for large positive arguments $\Delta p \equiv \hat{p}_1 - p_0$ returns the argument Δp (meaning that $\tilde{p}_1 = \hat{p}_1$) and for large negative arguments returns δ_0:

$$\Delta_0 = \left\{ \begin{array}{lll} \delta_o & \text{if} & \dfrac{\Delta p}{\delta_0} \le q_{\min} \\ \left(\dfrac{\delta_0}{q_{\max}}\right)\left(\dfrac{\Delta p}{\delta_0}\right)\left\{1 + C_a\left[1 - C_b\left(\dfrac{\Delta p}{\delta_0}\right)\right]^2\right\} & \text{if} & q_{\min} < \dfrac{\Delta p}{\delta_0} < q_{\max} \\ \Delta \mathrm{p} & \text{if} & \dfrac{\Delta p}{\delta_0} \ge q_{\max} \end{array} \right\}, \tag{51}$$

where the constants are given by

$$C_a = \frac{q_{\min}^2 (q_{\max} - 1)}{(q_{\max} - q_{\min})^2}, \quad C_b = q_{\min}^{-1}. \tag{52}$$

The limits $q_{\min}$ and $q_{\max}$ control the width of the transition zone between isopycnic and z coordinates. They are set to default values of $q_{\min} = -2$ and $q_{\max} = 4$.

If temperature and salinity are fluxed across the relocated interfaces, then perfect restoration of isopycnic conditions is impossible$\left(\widetilde{\alpha_1} \neq \widehat{\alpha_1}\right)$. In rare cases, the change in α_1 may not even have the expected sign. For instance, in model layers just below the Mediterranean salt tongue, studies were observed where raising interface 1 unexpectedly decreased α_1. Repeated application of the vertical coordinate adjustment algorithm then serves to drive α_1 further from its reference value and generates unacceptably large vertical coordinate migration. This difficulty can be exacerbated when T and S are advected. HYCOM 2.2 includes a feature to suppress the vertical coordinate adjustment when the change in α_1 does not carry the expected sign. Although this prevents α_1 from diverging quickly from its reference value, restoration to the reference value is impossible. The only way to completely circumvent this issue is to adjust temperature and density, or salinity and density.

Even without this issue, substantial deviations from layer reference density could still occur when T and S are altered. An iterative procedure has been included to insure that $\left|\widetilde{\alpha}_1 - \widehat{\alpha}_1\right|$ is smaller than a prescribed tolerance after employing the vertical coordinate adjustment algorithm. This iterative procedure works in the following way: After $\widehat{p}_1$ is estimated from (49), changes in T and S as a result of fluxes across the relocated vertical coordinate are approximated, then $\widetilde{\alpha}_1$ is estimated from the model equation of state. If $\widetilde{\alpha}_1$ is outside the required tolerance, then $\widehat{p}_1$ and $\widetilde{\alpha}_1$ are updated again and the tolerance test applied. Up to five iterations are performed. Details are discussed below.

Case 1 has two sub-cases that are discussed separately:

Case 1a: $\widetilde{p}_1 < p_1$

Here, the upper interface is raised, permitting lighter water to increase α_1 to a value $\widetilde{\alpha}_1$ that is closer, but not necessarily equal to, the reference value $\widehat{\alpha}_1$. The estimate of $\widetilde{\alpha}_1$ is acquired by substituting tildes for carats in (48):

$$\widetilde{\alpha}_1 = \frac{\alpha_1(p_2 - p_1) + \alpha_0(p_1 - \widetilde{p}_1)}{p_2 - \widetilde{p}_1}. \tag{53}$$

If the user chooses to flux α and salinity or α and temperature, further modification of α is not necessary. However, if the user chooses to mix T and S, the next move is to see if the actual change in α has the expected sign, and if $\left|\tilde{\alpha}_1 - \hat{\alpha}_1\right|$ is within the desired tolerance. The new temperature value that results from raising the interface is

$$\tilde{T}_1 = \frac{p_1 - \hat{p}_1}{p_2 - \hat{p}_1}(T_0 - T_1). \tag{54}$$

The new salinity $\tilde{S}_1$ is approximated in the same way, and a new estimate of $\tilde{\alpha}_1$ is performed using the model equation of state. If the change in α_1 has the wrong sign, then p_1 is not adjusted. Otherwise, if $\left|\tilde{\alpha}_1 - \hat{\alpha}_1\right|$ surpasses the required tolerance, then $\hat{p}_1$ is recomputed using

$$\hat{p}_1 = \hat{p}_1 + \frac{\tilde{\alpha}_1 - \hat{\alpha}_1}{\alpha_0 - \hat{\alpha}_1}(p_2 - \hat{p}_1) \tag{55}$$

.

At this stage, $\hat{p}_1$ is not permitted to surpass p_1. The resulting specific volume $\tilde{\alpha}_1$ is then recomputed and if the prescribed tolerance is not met, $\hat{p}_1$ is re-calculated through the same procedure. The procedure is repeated up to five times as necessary. If the upward relocation of interface 1 is adequately limited by an interior blocking layer above, then a special algorithm to dissolve the blockage is used.

Case 1b: $\tilde{p}_1 > p_1$

In this example, the upper interface must be lowered to preserve the minimum thickness of layer 0. Conservation of $\int \alpha dp$ then leads to

$$\tilde{\alpha}_0 = \frac{\alpha_1(\tilde{p}_1 - p_0) + \alpha_0(p_1 - p_0)}{\tilde{p}_1 - p_0}. \tag{56}$$

The protection of minimum layer thickness by increasing the thickness of layer 0 supersedes all efforts to restore isopycnic conditions. Consequently, none of the algorithms described for case 1a are invoked.

Case 2: $\alpha_1 > \hat{\alpha}_1$

Case 2 has the lower interface shifting downward. Mass from layer 2 is entrained into layer 1. Conservation of $\int \alpha dp$ requires

$$\alpha_1(p_2 - p_1) + \alpha_2(p_3 - p_2) = \hat{\alpha}_1(\hat{p}_2 - p_1) + \alpha_2(p_3 - \hat{p}_2). \tag{57}$$

The lower interface must be repositioned to

$$\hat{p}_2 = \frac{p_1(\hat{\alpha}_1 - \alpha_1) + p_2(\alpha_1 - \alpha_2)}{\hat{\alpha}_1 - \alpha_2}. \tag{58}$$

To maintain the minimum thickness of layer 2, $\hat{p}_2$ is exchanged with

$$\tilde{p}_2 = \min(\hat{p}_2, p_3 - \Delta_2), \tag{59}$$

where Δ_2 is determined from the cushion function

$$\Delta_2 = \left\{ \begin{array}{lll} \delta_2 & \text{if} & \dfrac{\Delta p}{\delta_2} \le q_{\min} \\ \left(\dfrac{\delta_2}{q_{\max}}\right)\left(\dfrac{\Delta p}{\delta_2}\right)\left\{1 + C_a\left[1 - C_b\left(\dfrac{\Delta p}{\delta_2}\right)\right]^2\right\} & \text{if} & q_{\min} < \dfrac{\Delta p}{\delta_2} < q_{\max} \\ \Delta \text{p} & \text{if} & \dfrac{\Delta p}{\delta_2} \ge q_{\max} \end{array} \right\}. \qquad (60)$$

with $\Delta p \equiv p_3 - \hat{p}_2$. The iterative algorithm to improve restoration of isopycnic conditions is also performed when salinity and temperature are adjusted. This algorithm is used throughout the water column except for the deepest layer with nonzero thickness, which intersects the bottom. Case 2a below is used for this layer.

Case 2a: $\alpha_1 > \hat{\alpha}_1$: Layer 1 is the Bottom Layer

Since interface 2 cannot move downward here, it is necessary to use constraint (48) and move interface 1 down to revive isopycnic conditions in layer 1. To achieve this goal, the water in layer 1 must be restratified into two sublayers so that the density of the upper sublayer precisely equals the density of layer 0, the density of the lower sublayer nears the desired reference density, and the density averaged over the two sublayers equals the original layer density.

Given

$$q = \frac{\alpha_1 - \hat{\alpha}_1}{\alpha_0 - \hat{\alpha}_1},$$

interface 1 is repositioned using

$$\hat{p}_1 = p_1 + q(p_2 - p_1). \qquad (61)$$

Thermodynamical variables in the lower sublayer are then computed using

$$\hat{T}_1 = T_1 - \left(\frac{q}{1-q}\right)(T_1 - T_0) \qquad (62)$$

for temperature, and using the equivalent equation for salinity and density. Once again, two of the thermodynamical variables are identified using (62) with the third coming from the equation of state. The closeness of lower sublayer density to the reference density is sacrificed if necessary to achieve two goals: 1) to prevent the thickness of layer 1 from decreasing by more than 50% using

$$\tilde{p}_1 = \min(\hat{p}_1, p_2 - \Delta_2), \qquad (63)$$

where $\Delta_2 = (p_2 - p_1)/2$, and 2) to avoid runaway changes in T and S using

$$|T_1 - T_0| \le |T_0 - T_{-1}| \qquad (64)$$

for temperature and the equivalent equation for salinity.

Vertical Remapping of Dynamical and Thermodynamical Layer Variables

The remapping of thermodynamical variables and momentum after vertical coordinates are rearranged must conserve their vertically averaged values and, for the thermodynamical variables, restore density to the isopycnic reference value as much as possible. The algorithm given in HYCOM 2.2 remaps each variable from the old to the new vertical grid. It satisfies these two conditions and does not depend on the direction (top to bottom or bottom to top) in which it is performed, unlike a pure donor cell scheme.

The user can adjust any two of the thermodynamical variables and diagnose the third by using the equation of state. When the hybrid coordinate generator is called, it is implemented separately at each grid point. Thermodynamical variables are adjusted first at pressure grid points. Before tuning the vertical coordinates, the initial 1D profiles of temperature, salinity, and density, including the 1D array of interface pressures, are saved. If the full Kraus-Turner mixed layer model is chosen, the layer containing the MLB is separated into two sublayers, and the MLB is temporally considered to be an extra vertical coordinate. The sublayers above and below the MLB are therefore temporarily considered to be two model layers. Thermodynamical variables in the two sublayers are approximated using the "unmixing" algorithm described in Section 5.2.7.4.

Once the profiles are stored, the vertical adjustment of vertical coordinates is carried out at the pressure grid points. Each variable is then remapped from the old to the new vertical coordinates as shown here for temperature. The interface pressures on the old grid are $p_m, m = 1, M$ and the pressures on the new adjusted grid are $\tilde{p}_n, n = 1, N$, where N is the number of model layers. Notice that $M = N$ unless the full Kraus-Turner mixed layer model is used, in which case $M = N + 1$ to account for the two sublayers within the layer holding the MLB. The old temperature profile is mapped onto the new adjusted vertical coordinates using

$$\tilde{T}_n = \frac{1}{\tilde{p}_{n+1} - \tilde{p}_n} \sum_{m=1}^{M} T_m \left\{ \max\left[\tilde{p}_n, \min\left(p_{m+1}, \tilde{p}_{n+1} \right) \right] - \min\left[\tilde{p}_{n+1}, \max\left(p_m, \tilde{p}_n \right) \right] \right\}. \qquad (65)$$

The summation is performed between $m = m_1$ to $m = m_2$ in order to eliminate layers on the old vertical grid that do not overlap layer n on the new grid.

After tuning the thermodynamical variables, the momentum components are adjusted to insure that vertically-averaged momentum is conserved at every grid point. The old and new vertical coordinates gathered at pressure grid points are first interpolated to the u grid points. The adjustment of u is executed using equation (65). The same technique is used to update v at the v grid points.

Reducing diffusion requires that the grid generator use the piecewise linear method with a monotonized central-difference (MC) limiter (van Leer, 1977) for layers in fixed coordinates $\left(1 \le k \le N_h\right)$ while continuing to use the remapping technique for layers that are non-fixed (and therefore tending to isopycnal coordinates). The piecewise linear method takes the place of the "constant within each layer" profile of donor-cell with a linear profile that equals the layer average in the middle of the layer. The slope is limited to maintain monotonicity. There

are several potential limiters but the MC limiter is quite popular (Leveque, 2002). To further minimize numerical diffusion, vertical coordinates in the isopycnic interior are only relocated 1/8 of the distance necessary to completely restore isopycnic conditions. With the grid generator applied at each time step, layer densities are still kept close to the isopycnic target densities.

5.2.7 HYCOM Mixed Layer Submodel Formulations

5.2.7.1 KPP Vertical Mixing Algorithm

The K-Profile Parameterization (KPP; Large *et al*., 1994; 1997) is the first non-slab mixed layer model incorporated into HYCOM 2.2. The KPP model offers mixing from surface to bottom, efficiently matching the large surface boundary layer diffusivity/viscosity profiles to the weak diapycnal diffusivity/viscosity profiles of the interior ocean. This model functions on a relatively coarse and unevenly spaced vertical grid. It parameterizes the influence of a larger group of physical processes than other frequently used mixing schemes. In the ocean interior, the role of background internal wave breaking, shear instability mixing, and double diffusion (both salt fingering and diffusive instability) are parameterized. In the surface boundary layer, there is parameterization of wind-driven mixing influences, surface buoyancy fluxes, and convective instability. The KPP algorithm also parameterizes the influence of nonlocal mixing of S and T, which allows for the development of countergradient fluxes.

The KPP model is semi-implicit and requires multiple iterations. The first iteration has vertical profiles of diffusivity/viscosity coefficients computed at model interfaces from the initial model variable profiles. The model variables are then mixed by solving the 1D vertical diffusion equation at every grid point. The solution procedure, involving the formulation of a matrix problem and the inversion of a tri-diagonal matrix, is described in Section 5.2.12. In the second iteration, the vertically mixed profiles of model variables estimate new diffusivity/viscosity profiles, which then mix the original profiles of model variables. This procedure is repeated until the mixed profiles of model variables differ irrelevantly from the previous iteration's mixed profiles. HYCOM 2.2 tests revealed that two iterations are reasonably sufficient, given the computational overhead necessary for each one.

The full KPP formulation is first applied at pressure grid points, where thermodynamical variables and tracers are mixed. For this reason, momentum components are horizontally interpolated to the pressure grid points. After completing the iterative procedure at pressure points, mixing is done at momentum (u and v) points by interpolating the final viscosity profiles at pressure points to the momentum points, then solving the vertical diffusion equation. The full iterative procedure is not required at the momentum points.

The KPP algorithm does not need a convection algorithm that mixes adjacent layers when the upper layer is denser than the lower layer. HYCOM 2.2 performs convection when Kraus-Turner mixing is used (see Section 5.2.7.4).

The next three sections describe how surface-to-bottom diffusivity/viscosity profiles are

computed during each iteration of the KPP algorithm. The final section explains how the semi-implicit scheme performs the vertical mixing at both pressure and momentum grid points.

Surface Fluxes

Before executing the KPP algorithm, surface fluxes of thermodynamical variables and momentum are evenly distributed over the uppermost model layer, excluding penetrating shortwave radiation. This radiation heats deeper layers down to depths that depend on water clarity. For further information see Section 5.2.11.

Diapycnal Diffusivity in the Ocean Interior

Model variables are divided into mean (denoted by an overbar) and turbulent (denoted by a prime) components. Diapycnal diffusivities and viscosity are parameterized in the ocean interior as:

$$\overline{w'\theta'} = -\nu_\theta \frac{\partial \overline{\theta}}{\partial z}, \quad \overline{w'S'} = -\nu_S \frac{\partial \overline{S}}{\partial z}, \quad \overline{w'\mathbf{v}'} = -\nu_m \frac{\partial \overline{\mathbf{v}}}{\partial z}, \tag{66}$$

where $(\nu_\theta, \nu_S, and\, \nu_m)$ are the interior diffusivities of potential temperature, salinity (includes other scalars), and momentum (viscosity), respectively. Interior diffusivity/viscosity is assumed to have three components, which are illustrated here for potential temperature:

$$\nu_\theta = \nu_\theta^s + \nu_\theta^w + \nu_\theta^d, \tag{67}$$

where ν_θ^s is resolved shear instability, ν_θ^w is the contribution of unresolved shear instability due to the background internal wave field, and ν_θ^d is double diffusion. Only the first two processes contribute to viscosity.

The shear instability contribution is parameterized in terms of the gradient Richardson number computed at model interfaces:

$$Ri_g = \frac{N^2}{\left(\frac{\partial \overline{u}}{\partial z}\right)^2 + \left(\frac{\partial \overline{v}}{\partial z}\right)^2}, \tag{68}$$

where mixing is prompted when $Ri_g = Ri_0 < 0.7$. Vertical derivatives are approximated at model interfaces as follows:

Given model layer k bounded by interfaces k and $k+1$, the vertical derivative of $\overline{u}$ at interface k is approximated as

$$\frac{\partial \overline{u}}{\partial z} = \frac{\overline{u}^{k-1} - \overline{u}^k}{0.5 * \left(\delta h^k + \delta h^{k-1}\right)}, \tag{69}$$

where the denominator holds the thickness of layers k and $k-1$. The contribution of shear instability is identical for θ diffusivity, S diffusivity, and viscosity $\left(\nu^s = \nu_\theta^s = \nu_S^s = \nu_m^s\right)$, and is given by

$$\frac{\nu^s}{\nu^0} = 1 \qquad Ri_g < 0$$
$$\frac{\nu^s}{\nu^0} = \left[1 - \left(\frac{Ri_g}{Ri_0}\right)^2\right]^P \qquad 0 < Ri_g < Ri_0, \text{ and} \tag{70}$$
$$\frac{\nu^s}{\nu^0} = 0 \qquad Ri_g > Ri_0,$$

where $\nu^0 = 50 \times 10^{-4}\,\mathrm{m^2 s^{-1}}$, $Ri_0 = 0.7$, and $P = 3$.

The diffusivity resulting from unresolved background internal wave shear is given by

$$\nu_\theta^w = \nu_S^w = 0.1 \times 10^{-4}\,\mathrm{m^2 s^{-1}}. \tag{71}$$

Based on Peters *et al.* (1988), the Large *et al.* authors (1994) concluded that viscosity should be an order of magnitude larger:

$$\nu_m^w = 1.0 \times 10^{-4}\,\mathrm{m^2 s^{-1}}. \tag{72}$$

Regions where double diffusive processes are important are found using the double diffusion density ratio computed at model interfaces:

$$R_\rho = \frac{\alpha \dfrac{\partial \overline{\theta}}{\partial z}}{\beta \dfrac{\partial \overline{S}}{\partial z}}, \tag{73}$$

where α and β represent thermodynamical expansion coefficients for T and S, respectively. For salt fingering (warm, salty water overlying cold, fresh water), salinity/scalar diffusivity is shown by

$$\frac{\nu_S^d}{\nu_f} = \left[1 - \left(\frac{R_\rho - 1}{R_\rho^0 - 1}\right)^2\right]^P \qquad 1.0 < R_\rho < R_\rho^0$$
$$\frac{\nu_S^d}{\nu_f} = 0 \qquad R_\rho \geq R_\rho^0 \tag{74}$$

and temperature diffusivity is given by

$$\nu_\theta^d = 0.7 \nu_S^d, \tag{75}$$

where $\nu_f = 10 \times 10^{-4}\,\mathrm{m^2 s^{-1}}$, $R_\rho^0 = 1.9$, and $P = 3$. For diffusive convection, temperature diffusivity is provided by

$$\frac{\nu_\theta^d}{\nu} = 0.909 \exp\left\{4.6 \exp\left[-0.54\left(R_\rho^{-1} - 1\right)\right]\right\}, \tag{76}$$

where ν represents the molecular viscosity of temperature, while salinity/scalar diffusivity is given by

$$\nu_S^d = \nu_\theta^d \left(1.85 - 0.85 R_\rho^{-1}\right) R_\rho \qquad 0.5 \leq R_\rho \leq 1$$
$$\nu_S^d = \nu_\theta^d \left(0.15 R_\rho\right) \qquad R_\rho < 0.5 \tag{77}$$

Surface Boundary Layer Thickness

Diagnosis of boundary layer thickness h_b is derived from the bulk Richardson number

$$Ri_b = \frac{\left(B_r - B\right)d}{\left(\overline{\mathbf{v}}_r - \overline{\mathbf{v}}\right)^2 + V_t^2}, \tag{78}$$

where B denotes buoyancy, d is depth, the subscript r represents reference values, and where the two terms in the denominator denote the influence of resolved vertical shear and unresolved turbulent velocity shear, respectively. Reference values are averaged over the depth range εd, where $\varepsilon = 0.1$. At depth $d = h_b$, the reference depth εh_b symbolizes the thickness of the surface layer where Monin-Obukhov similarity theory applies. If model layer 1 is more than 7.5 m thick, reference values in (78) are set to layer 1 values. Otherwise, averaging is done over depth range εd.

The surface boundary layer thickness (which is different from mixed layer thickness) is the depth range over which turbulent boundary layer eddies penetrate before becoming stable compared with the local buoyancy and velocity. It is calculated as the minimum depth at which Ri_b exceeds the critical Richardson number $Ri_c = 0.3$. The Richardson number Ri_b is given in (78) as a layer variable, and therefore assumed to be the Richardson number at the middle depth of each layer. Moving downward from the surface, Ri_b is computed for each layer. When the first layer is reached where $Ri_b > 0.3$, h_b is found by linear interpolation between the central depths of that layer and the layer above.

The unresolved turbulent velocity shear in the denominator of (78) is found from

$$V_t^2 = \frac{C_s\left(-\beta_T\right)^{1/2}}{Ri_c\kappa^2}\left(c_s\varepsilon\right)^{-1/2} dNw_s, \tag{79}$$

where C_s is constant between 1 and 2, β_T is the ratio of entrainment buoyancy flux to surface buoyancy flux, $\kappa = 0.4$ is von Karman's constant, and w_s is the salinity/scalar turbulent velocity scale. The salinity/scalar turbulent velocity scale is derived using

$$\begin{aligned} w_S &= \kappa\left(a_S u^{*3} + c_S\kappa\sigma w^{*3}\right)^{1/3} \rightarrow \kappa\left(c_S\kappa\sigma\right)^{1/3} w^* \qquad \sigma < \varepsilon \\ w_S &= \kappa\left(a_S u^{*3} + c_S\kappa\varepsilon w^{*3}\right)^{1/3} \rightarrow \kappa\left(c_S\kappa\varepsilon\right)^{1/3} w^* \qquad \varepsilon \le \sigma < 1 \end{aligned} \tag{80}$$

where a_x and c_x are constants, $w^* = \left(-B_f / h\right)^{1/3}$ represents convective velocity scale with B_f as the surface buoyancy flux, and $\sigma = d / h_b$. Expressions to the right of the arrows denote the convective limit. In HYCOM 2.2, w_s values are kept in a 2D lookup table as functions of u^{*3} and σw^{*3} to reduce computations. If the surface forcing is stabilizing, the diagnosed value of h_b must be smaller than both the Ekman length scale $h_E = 0.7u^* / f$ and the Monin-Obukhov length L.

Surface Boundary Layer Diffusivity

After computing interior diapycnal diffusivity at model interfaces and estimating h_b, surface boundary layer diffusivity/viscosity profiles are computed at model interfaces and smoothly paired to the interior diffusivities and viscosity. Boundary layer diffusivities and viscosity are

parameterized as:

$$\overline{w'\theta'} = -K_\theta\left(\frac{\partial\overline{\theta}}{\partial z}+\gamma_\theta\right),\quad \overline{w'S'} = -K_S\left(\frac{\partial\overline{S}}{\partial z}+\gamma_S\right),\ and\quad \overline{w'\mathbf{v}'} = -K_m\left(\frac{\partial\overline{\mathbf{v}}}{\partial z}\right), \tag{81}$$

where γ_θ,γ_S are nonlocal transport terms. The diffusivity/viscosity profiles are parameterized as

$$K_\theta(\sigma)=h_b w_\theta(\sigma)G_\theta(\sigma),\quad K_S(\sigma)=h_b w_S(\sigma)G_S(\sigma),\quad K_m(\sigma)=h_b w_m(\sigma)G_m(\sigma), \tag{82}$$

where G is a smooth shape function represented by a third-order polynomial function

$$G(\sigma)=a_0+a_1\sigma+a_2\sigma^2+a_3\sigma^3 \tag{83}$$

that is determined separately for each model variable. The salinity/scalar velocity scale w_S is found using (80). The potential temperature and momentum velocity scales w_θ, w_m are also produced from equations analogous to (80), but with the two constants replaced by a_θ, c_θ and a_m, c_m, respectively. Because turbulent eddies do not cross the ocean surface, all K coefficients are zero at the surface, requiring that $a_0 = 0$. The remaining coefficients of the shape function for a given variable are selected to fulfill requirements of Monin-Obukhov similarity theory and to insure that the resulting value and first vertical derivative of the boundary layer K-profile fits the value and first derivative of the interior ν profile for the same variable computed using (67) through (72) and (73) through (77).

Application of this method is shown here for potential temperature only. The matching yields:

$$\begin{aligned} G_\theta(1) &= \frac{\nu_\theta(h_b)}{h_b w_\theta(1)} \\ \frac{\partial}{\partial\sigma}G_\theta(1) &= -\frac{\frac{\partial}{\partial z}\nu_\theta(h_b)}{w_\theta(1)}-\frac{\nu_\theta(h_b)\frac{\partial}{\partial\sigma}w_\theta(1)}{h w_\theta^2(1)} \end{aligned}. \tag{84}$$

After determining the coefficients in (83), the K profile is computed using

$$K_\theta = h_b w_\theta \sigma\left[1+\sigma G_\theta(\sigma)\right], \tag{85}$$

where

$$G_\theta(\sigma)=(\sigma-2)+(3-2\sigma)G_\theta(1)+(\sigma-1)\frac{\partial}{\partial\sigma}G_\theta(1). \tag{86}$$

At model interfaces within the surface boundary layer, the K profile for potential temperature is provided by (85). At model interfaces below the boundary layer, the K profile equals the interior diffusivity ($K_\theta = \nu_\theta$).

The nonlocal flux terms in (81) activate when the surface forcing is destabilizing. The KPP model parameterizes nonlocal flux only for scalar variables. Although nonlocal fluxes could also be important for momentum, the form these fluxes take is currently unknown (Large *et al*., 1994). The nonlocal fluxes for scalar variables are parameterized as

$$\begin{aligned} &\gamma_\theta = 0 \quad \gamma_s = 0 \quad \zeta \ge 0 \\ &\gamma_\theta = C_s\frac{\overline{w'\theta'}_0+\overline{w'\theta'}_R}{w_\theta(\sigma)h_b} \quad \gamma_S = \frac{\overline{w'S'}_0}{w_s(\sigma)h} \quad \zeta<0, \end{aligned} \tag{87}$$

where ζ is a stability parameter equal to d/L and L denotes Monin-Obukhov length. The terms $\overline{w'\theta'}_0$ and $\overline{w'S'}_0$ are surface fluxes and the term $\overline{w'\theta'}_R$ is the penetrating shortwave radiation contribution.

Vertical Mixing

Given the K profiles at pressure grid points, 1D vertical diffusion is solved at each grid point by preparing a matrix equation and inverting a tri-diagonal matrix. Details of this procedure are found in Section 5.2.12. After solving for all variables at the pressure grid points (including velocity components interpolated from the momentum grid points), the KPP procedure is repeated starting with equation (66) using the new profiles of all variables. The user selects the number of iterations to be performed. Two iterations are typically found to be sufficient. Mixed layer thickness is gauged at the pressure grid points through vertical interpolation to the depth where density exceeds the surface layer density by a set amount.

After mixing at pressure grid points is complete, the momentum grid points are mixed. Instead of repeating the entire KPP procedure, the K_m profiles estimated at the pressure grid points during the final iteration are horizontally interpolated to the u and v grid points and the vertical diffusion equation is then solved.

5.2.7.2 The Mellor-Yamada Level 2.5 Turbulence Closure Model

HYCOM 2.2 has been equipped with the Mellor-Yamada Level 2.5 turbulence closure algorithm, providing mixing from surface to bottom. It concurrently provides relatively weak diapycnal mixing of the ocean interior, strong mixing of the surface boundary layer, and enhanced mixing near the ocean bottom. For additional information on the model and the physical assumptions upon which it is based, refer to the Princeton Ocean Model Users Guide (Mellor, 1998) or to Mellor and Yamada (1982).

Summarizing the algorithm, we begin with parameterizing viscosity and scalar diffusivity as follows:

$$\begin{aligned} K_M &= qlS_M \\ K_H &= qlS_H, \end{aligned} \tag{88}$$

where S_M and S_H are given as

$$S_H\,[1-(3A_2B_2+18A_1A_2)\,G_H\,] = A_2\left(1-\frac{6A_1}{B_1}\right) \tag{89}$$

and

$$S_M\,(1-9A_1A_2G_H\,)-S_H\,[(18A_1+9A_1A_2)\,G_H\,] = A_1\left(1-3C_1-\frac{6A_1}{B_1}\right), \tag{90}$$

with

$$G_H = -\frac{l^2}{q^2}\frac{g}{\rho_0}\frac{\partial\rho}{\partial z} \tag{91}$$

being a Richardson number. In these equations, q is the turbulence velocity scale, l represents the turbulence length scale, and ρ denotes potential density. The variables q^2 (turbulent kinetic energy) and $q^2 l$ are prognostic variables. The equations for these variables, given here as a function of the generalized vertical coordinate s for the purpose of HYCOM 2.2 implementation, are

$$\frac{\partial}{\partial t_s}\left(\frac{\partial p}{\partial s}q^2\right)+\nabla_s\cdot\left(\mathbf{V}\frac{\partial p}{\partial s}q^2\right)-\nabla_s\cdot\left[A_H\frac{\partial p}{\partial s}\nabla_s\left(q^2\right)\right]+\frac{\partial}{\partial s}\left(\frac{ds}{dt}\frac{\partial p}{\partial s}q^2\right)=\mathscr{S}_q \quad (92)$$

and

$$\begin{aligned}&\frac{\partial}{\partial t_s}\left(\frac{\partial p}{\partial s}\left(q^2 l\right)\right)+\nabla_s\cdot\left(\mathbf{V}\frac{\partial p}{\partial s}\left(q^2 l\right)\right)\\&-\nabla_s\cdot\left(A_H\frac{\partial p}{\partial s}\nabla_s\left(q^2 l\right)\right)+\frac{\partial}{\partial s}\left(\frac{ds}{dt}\frac{\partial p}{\partial s}\left(q^2 l\right)\right)=\mathscr{S}_l,\end{aligned} \quad (93)$$

where $\mathscr{S}_q$ and $\mathscr{S}_l$ are the sum of all local processes. Equations (92) and (93) are of the same form as the T and S equations in generalized vertical coordinates. Terms two through four in both expressions represent horizontal advection, horizontal diffusion, and fluxes across generalized vertical coordinates when these coordinates are relocated ($ds/dt \neq 0$) by the hybrid coordinate adjustment algorithm. For S and T, the local processes are vertical diffusivity and surface forcing. For q^2 and $q^2 l$, the local processes are boundary forcing and vertical diffusivity as well as three more forcing and damping mechanisms described below.

The strategy for applying this turbulence closure model in HYCOM 2.2 is to solve the same local equations for q^2 and $q^2 l$ solved in POM. The local equations are

$$\frac{\partial}{\partial t}\left(q^2\right)=F_q+\frac{\partial}{\partial z}\left[K_q\frac{\partial}{\partial z}\left(q^2\right)\right]+2K_M\left[\left(\frac{\partial u}{\partial z}\right)^2+\left(\frac{\partial v}{\partial z}\right)^2\right]+\frac{2g}{\rho_0}K_H\frac{\partial \rho}{\partial z}-\frac{2q^3}{B_1 l} \quad (94)$$

and

$$\begin{aligned}&\frac{\partial}{\partial t}\left(q^2 l\right)=F_l+\frac{\partial}{\partial z}\left(K_q\frac{\partial}{\partial z}\left(q^2 l\right)\right)\\&+E_1 l\left[K_M\left(\left(\frac{\partial u}{\partial z}\right)^2+\left(\frac{\partial v}{\partial z}\right)^2\right)+E_3\frac{g}{\rho_0}K_H\frac{\partial \rho}{\partial z}\widetilde{W}-\frac{q^3}{B_1}\right],\end{aligned} \quad (95)$$

where F_q and F_l are local boundary forcing and the last four terms of each equation are vertical diffusion, generation by vertical shear, conversion to and from potential energy, and dissipation. The parameter $\widetilde{W}$ is the wall proximity function given as $\widetilde{W}=1+E_2(l+kL)$, with $L^{-1}=(\eta-z)^{-1}+(H-z)^{-1}$, and where η is sea surface elevation and H is water depth. The local solution procedure in HYCOM 2.2 is essentially the same as in POM, as are several constant values, which are described in Mellor (1998).

Existing HYCOM 2.2 algorithms are used to compute changes in q^2 and $q^2 l$ due to the two nonlocal terms and the ds/dt term in (92) and (93). Horizontal advection and diffusion of q^2

and q^2l are executed in subroutine *tsadvc.f*, while the ds/dt flux terms are approximated in the hybrid vertical coordinate subroutine *hybgen.f*. Both q^2 and q^2l are carried at the two leap frog time steps, and time smoothing is applied using the same method as other model scalars.

In the POM Mellor-Yamada algorithm, the variables q^2 and q^2l are carried at vertical interfaces while HYCOM 2.2 scalars are carried as layer variables. This involves defining a separate vertical coordinate system on which the 1D solutions for q^2 and q^2l are performed. M-Y interfaces sit at the central depths of HYCOM 2.2 layers. There is an additional layer and interface in the M-Y coordinate system. With q^2 and q^2l defined on M-Y interfaces, implementation of the POM local solution procedure in HYCOM 2.2 is simple. The M-Y vertical coordinate system is shown alongside the HYCOM 2.2 vertical coordinate system in Figure 2. Because M-Y interfaces are positioned at the central depths of HYCOM 2.2 layers, values on M-Y interfaces are assumed to represent values in the associated HYCOM 2.2 layers. The resulting layer arrays are then passed to the HYCOM 2.2 routines that compute the horizontal advection, horizontal diffusion, and ds/dt flux terms of (92) and (93).

Instead of using POM procedures to solve vertical diffusion equations for other model variables, HYCOM 2.2 uses portions of the KPP mixing procedure. Specifically, the vertical viscosity and scalar diffusivity profiles from equation (89) obtained from the local solutions of (94) and (95) are employed in the tridiagonal matrix algorithm described in Section 5.2.12, which solves the vertical diffusion equations of all other model scalars. Refer to Section 5.2.7.1 for details on how the solution is carried out for HYCOM 2.2 variables.

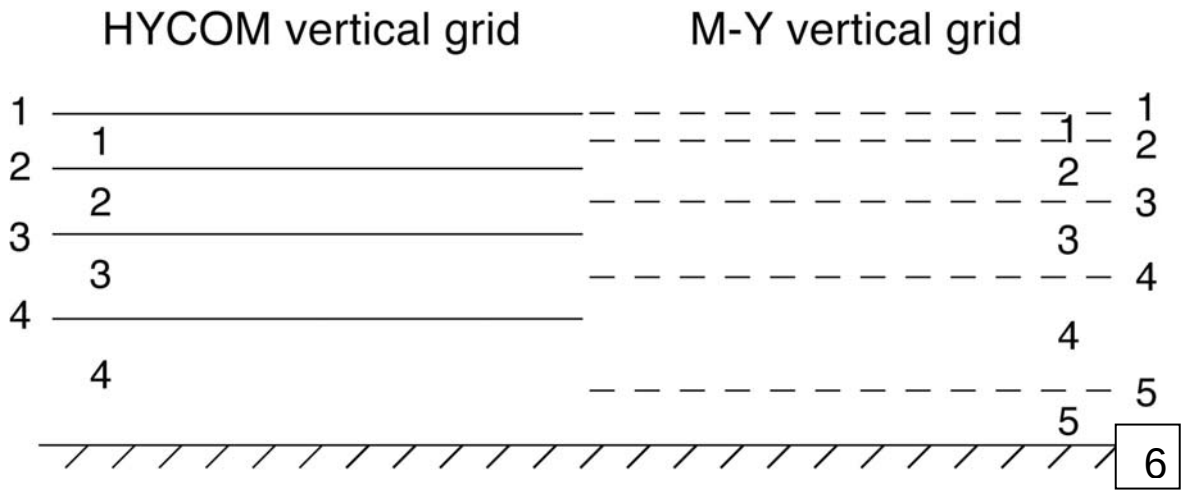


Figure 2: Side-by-side comparison of the HYCOM 2.2 vertical grid and the M-Y vertical grid used to perform the local solutions of (94) and (95).

5.2.7.3 Price-Weller-Pinkel Dynamical Instability Vertical Mixing Algorithm

HYCOM 2.2 contains the Price, Weller, and Pinkel (PWP) dynamical instability model for vertical mixing (*Price et al.,* 1986). Mixing is accomplished in three steps: (1) static instability relief in the upper-ocean mixed layer, (2) bulk mixed layer entrainment based on a bulk Richardson number, and (3) vertical shear instability mixing between adjacent layers based on a gradient Richardson number.

Step 1: Relief of Static Instability

The PWP algorithm functions at pressure grid points. The mixing algorithm is executed by horizontally interpolating velocity components on u and v grid points to p grid points. After mixing at p points, the new mixed layer thickness is interpolated to u and v grid points so that momentum can be mixed.

First, surface boundary conditions are applied. Surface momentum fluxes are added in **momtum.f**, and act by accelerating the water in layer 1 only, before the PWP mixing algorithm is executed. Once the PWP subroutine is called, the surface fluxes of T and S are applied as described in Section 5.2.11. Shortwave radiation penetrates to heat the surface and subsurface model layers, with the exponential decay of red and blue wavelengths as a function of Jerlov water type. The residual surface heat flux, along with all surface mass flux, modifies the fluid in model layer 1 only.

If static instability remains near the ocean surface after applying surface fluxes, it is relieved as follows: If model layer 2 is less dense that layer 1, the water in both layers is entirely mixed. If the water in layer 3 is less dense than this mixed water, it is completely mixed with the water from layers 1 and 2. The process is repeated until a model layer is encountered that is denser than the mixed water above.

The initial diagnosis of mixed layer depth is then performed. Moving down from the surface, the mixed layer depth is assigned to the depth of the first interface across which the density jump exceeds a set value. The MLB always resides on a model interface in the PWP model. Model variables, including velocity components that were interpolated to p points, are homogenized within the mixed layer.

Step 2: Bulk Richardson Number Instability

The bulk Richardson number is calculated using

$$R_b = \frac{g\Delta\rho h}{\rho_0\left[(\Delta u)^2 + (\Delta v)^2\right]}, \tag{96}$$

where h denotes mixed layer thickness. Density and velocity differences are taken between the mixed layer and the model layer immediately below. If $R_b < 0.65$, the mixed layer entrains the layer directly below, homogenizing all variables within the new mixed layer. The process is repeated, with the mixed layer entraining additional layers, until $R_b \geq 0.65$.

Step 3: Gradient Richardson Number Instability

The gradient Richardson number is estimated using

$$R_g = \frac{g\Delta z\Delta\rho}{\rho_0\left[(\Delta u)^2 + (\Delta v)^2\right]}, \tag{97}$$

and if $R_g < 0.25$, mixing is performed. At every interface, $\Delta\rho$, Δu, and Δv are evaluated as the difference between the layer above and the layer below, while Δz is evaluated as the average thickness of the layers above and below.

The procedure below performs shear instability mixing. The number R_g is evaluated at all vertical interfaces between the MLB and the bottom. The interface with the smallest value of R_g is noted. If $R_g < 0.25$, the layers above and below the interface are somewhat mixed so that R_g is increased to 0.30. New values of R_g are computed and a new interface with the smallest value of R_g is identified. If this value is less than 0.25, the two adjacent layers are partially mixed in the same way. This process is repeated until the minimum value of R_g over all layers exceeds 0.25.

After this mixing is completed, the MLB depth is set as the depth of the first interface below the surface where the density jump exceeds a prescribed value. The final vertical homogenization is performed for thermodynamical variables and other scalars that are saved at p grid points.

Momentum Mixing

After mixing at pressure grid points, the depth of the MLB is horizontally interpolated to u and v grid points, and the velocity components are homogenized within the mixed layer. On both sets of grids, the model interface closest to the interpolated mixed layer depth is identified as the MLB, and vertical homogenization is performed between the surface and the recognized interface.

Background Interior Diapycnal Mixing

While the PWP algorithm provides for shear instability mixing below the surface mixed layer, it does not provide for background mixing due to other processes like internal wave breaking. When PWP is selected, the hybrid explicit diapycnal-mixing algorithm is activated to provide

the additional mixing. Price *et al.* (1986) gives more information on the PWP model and its physical assumptions.

5.2.7.4 Krauss-Turner Mixed Layer Models

The Kraus-Turner mixed layer is a vertically homogenized water layer with a depth taken from the turbulence kinetic energy (TKE) equation. It is converted into a diagnostic equation by setting the time-dependent term to zero, assuming a balance between sources and sinks of TKE in the water column. The focus here is on the three different K-T models included in HYCOM 2.2 along with matters related to the implementation of K-T mixing in a hybrid coordinate model. Although these models use the same TKE balance to direct entrainment/detrainment, the performance of these models differs drastically due to discrepancies in the numerical algorithms.

Since the K-T model only governs surface mixed layer mixing, the user must also utilize one of the interior diapycnal mixing algorithms provided in HYCOM 2.2. These models are described in Section 5.2.2.

Model 1: Full K-T Model (hybrid coordinates with unmixing)

The greatest difficulty of merging a Kraus-Turner mixed layer model to a hybrid coordinate ocean model is in properly handling the mixed layer base (MLB), the depth of which is a model prognostic variable. In isopycnic coordinate models, where the slab mixed layer is identical to layer 1, the MLB must coincide with a model vertical coordinate. This is not true in a hybrid coordinate model, so special accounting is required to track MLB depth along with discontinuities in thermodynamical and dynamical variables occurring at the MLB. The buoyancy change across the MLB must be known to approximate the contribution of entrainment to the TKE balance, and the magnitude of the discontinuity in a given property must be known to compute changes in the value of that property within the mixed layer due to entrainment.

In hybrid coordinates, the model layer containing the MLB, here as layer k, can be separated into one sublayer above the MLB that is part of the mixed layer and one sublayer below that is part of the stratified ocean interior. Water properties must be identified in both sublayers to quantify discontinuities across the MLB. Because the basic HYCOM 2.2 tracks only vertically averaged properties in model layers, an algorithm has been added to artificially generate estimates of water properties inside the two sublayers by "unmixing" the water column. The algorithm reduces numerically induced property exchanges between the mixed layer and the deeper ocean as much as possible. This feature is described here using mixed layer temperature. The temperature equals the average temperature between the surface and the MLB, and is therefore the average over model layers 1 through $k-1$ plus the upper sublayer of layer k. If the estimate of upper sublayer temperature created by the unmixing algorithm is inaccurate, neither the mean mixed layer temperature nor the lower sublayer temperature beneath the MLB will be accurate. An erroneous unmixing algorithm will lead to an incorrect

flux of temperature across the MLB. In test simulations of the Atlantic Ocean, model performance was very sensitive to unmixing errors, specifically at high latitudes. The accuracy has recently improved, particularly in the high latitude North Atlantic.

Between individual implementations of K-T mixing, water properties in layer k vary as a result of processes such as horizontal advection, diffusion, and penetrating shortwave radiation. With previous sublayer information lost, it is impossible to generate perfect estimates of upper and lower sublayer variables without getting erroneous property fluxes across the MLB. To solve this problem, the layer number k including the MLB, upper sublayer variables, and the averaged variables within k are stored at the end of the Kraus-Turner mixed layer algorithm. The next time K-T is called, the previously stored variable values are used to make a best-guess of upper sublayer variables during the unmixing process as shown below.

The Kraus-Turner model is implemented as follows:

- Thermodynamical variables are first mixed at the pressure grid points.
- A search determines the layer k containing the MLB.
- Temperature and salinity are averaged over layers 1 through $k-1$.
- Convection is performed if required.
- The density associated with the averaged T and S is computed. If greater than the layer k density, the MLB is moved to the layer k base (interface k+1).
- Temperature and salinity are averaged from the surface through layer k. If the resultant density is greater than the layer $k+1$ density, the MLB is moved down to interface $k+2$.
- The process is repeated until the program detects a layer with greater density than the mixed layer. If convection is a factor, the MLB will be located on a model interface, so unmixing is not required. In reality, whenever the MLB is located within one centimeter of a model interface, it is moved there and unmixing is unnecessary.

If the MLB is still within a model layer, the unmixing algorithm is performed. If $k \neq \hat{k}$, where $\hat{k}$ is the model layer that contained the MLB during the previous time step, then the following assumptions are made for upper sublayer T and S:

$$T_{up} = T_{k-1}, \quad S_{up} = S_{k-1}. \tag{98}$$

If $k = \hat{k}$, the first guesses below are made:

$$T_{up} = \hat{T}_{up} + T_k - \hat{T}_k, \quad S_{up} = \hat{S}_{up} + S_k - \hat{S}_k. \tag{99}$$

Upper sublayer variables are then assumed to have differed by a value equal to the change occurring in the full layer k variables. Lower sublayer variables are estimated using

$$T_{dn} = \frac{p_m - p_k}{p_{k+1} - p_m}\left(T_k - T_{up}\right) \tag{100}$$

and

$$S_{dn} = \frac{p_m - p_k}{p_{k+1} - p_m}\left(S_k - S_{up}\right), \tag{101}$$

where p_m denotes pressure level of the MLB. Spurious extrema of T and S are prevented by demanding that T_{dn} fall within the value packet defined by $\left(T_{up}, T_k, T_{k+1}\right)$, and S be within the values packet defined by $\left(S_{up}, S_k, S_{k+1}\right)$. If T_{dn} must be adjusted, then T_{up} is recalculated using

$$T_{up} - T_{up} + \frac{p_{k+1} - p_m}{p_m - p_k}\left(T_{dn} - T_k\right). \tag{102}$$

If S_{dn} must be adjusted, then S_{up} is recomputed using

$$S_{up} - S_{up} + \frac{p_{k+1} - p_m}{p_m - p_k}\left(S_{dn} - S_k\right). \tag{103}$$

After unmixing is complete, the density profile is sent to the K-T TKE algorithm to compute a new mixed layer depth. Two possibilities exist: First, density may be averaged over the mixed layer, providing a homogeneous slab mixed layer profile with a discontinuity at the MLB. Second, the unmixed density profile above the MLB (including the upper sublayer) can be provided to the K-T TKE algorithm. This is significant because in the previous time step, differential advection and diffusion in the mixed layer resulted in a heterogeneous profile in the mixed layer. Homogenizing the mixed layer before calling the K-T TKE algorithm changes the energetics of the mixed layer, generally leading to the K-T algorithm calculating a different MLB depth. Simulations in the Atlantic Ocean revealed that providing the heterogeneous profile to the K-T algorithm improved the realism of the model fields, particularly at high latitudes.

Temperature and salinity are homogenized over the maximum old and new mixed layer depths, and surface fluxes are distributed over the new depth. The final step is storing layer number k that holds MLB, the upper sublayer T and S, and the layer k average T and S to be used as the old values the next time the K-T algorithm is executed.

Next, momentum components on the *u* and *v* grid points are mixed. They are mixed from the surface to old and new mixed layer thickness maxima, given by $\tilde{p}_m$, that are interpolated from pressure grid points. If $\tilde{p}_m$ falls within one cm of $k+1$ at a *u* grid point, then *u* is homogenized from the surface through layer k. If the MLB is in layer k, unmixing is performed. Model

simulations are insensitive to the accuracy of u and v estimates in the two sublayers. For u and v, the first guess for the upper sublayer value is the layer $k-1$ value.

HYCOM 2.2 includes optional penetrating shortwave radiation. If this option is not selected, all shortwave radiation is assumed to be absorbed in the mixed layer.

Another point to consider is that the MLB is a material surface that can be advected by the flow field. This advection is partially accounted for by an algorithm added to the continuity equation. If MLB is in layer k, and model interfaces k and $k+1$ are adjusted vertically during solution of the continuity equation by δp_k and δp_{k+1}, respectively. The Kraus-Turner prognostic MLB at p_m is adjusted vertically by

$$\delta p_m = \frac{p_{k+1} - p_m}{p_{k+1} - p_k} \delta p_k + \frac{p_m - p_k}{p_{k+1} - p_k} \delta p_{k+1}. \tag{104}$$

The vertical motion at the MLB should be the linearly interpolated value between model interfaces k and $k+1$. Adding the vertical motion resulting from time smoothing of pressure interfaces is necessary so that interface adjustments δp_k and δp_{k+1} in (104) represent the sum of the continuity and time smoothing adjustments. The result is a MLB located at a model interface that always remains at the interface depth if no vertical mixing or diabatic heating/cooling occurs.

Additional vertical motion of the MLB is possible, however, because it can slope relative to the model vertical coordinates. The normal MLB flow component can produce additional vertical motion. This portion of the advection is not included due to ambiguities in determining the flow field at the MLB in hybrid coordinates.

Model 2: Simplified K-T Model (hybrid coordinates without unmixing)

Due to the intricacies of creating an unmixing scheme that diminishes numerically induced property exchange between the mixed layer and ocean interior, a simplified alternative K-T model was developed by relaxing the condition that the MLB be a prognostic variable. The tradeoff is between improved computational efficiency and increased numerical errors. Since the full K-T model 1 is far from perfect, this simplified model may be sufficient for most purposes. In Atlantic basin tests, model 2 had problems at high latitudes, where unrealistic patterns of deep water formation occurred and improbable pathways were observed for the Gulf Stream and North Atlantic Current. In tropical and subtropical latitudes with relatively strong stratification, model 2 clearly performed as well as the full model 1.

The K-T algorithm calculates the mixed layer thickness change over a small Δt, and thus requires old mixed layer input as the initial condition. This depth, which is the cumulative result of past applications of the K-T algorithm at a given point, can either be used as a prognostic variable (as in K-T model 1 above) or diagnosed *ad-hoc*. The present scheme uses option 2 to avoid unmixing and the ambiguities of constructing a representative mixed layer velocity field to advect the MLB (which as a material variable must follow the flow).

Starting at the top of the water column (layer 1), the algorithm looks for the first layer k whose density surpasses the average density $\overline{\rho}$ of the overlying layers combined (layers 1 through $k-1$). This layer is assumed to contain the MLB. Layer k is separated into sublayers with different densities. The upper sublayer density ρ_{up} is set equal to $\overline{\rho}$ while the lower sublayer density ρ_{lo} is given a value in the range $(\rho_k + \rho_{k+1})$, for example $\rho_{lo} = (\rho_k + \rho_{k+1})/2$. Density conservation during sublayer creation yields the depth of the interface separating the sublayers, which is taken to be the mixed layer depth. This pressure interface is given by

$$p_m = p_k(1-q) + p_{k+1}q, \tag{105}$$

where p_k, p_{k+1} are the upper and lower interface pressures of layer k, and

$$q = \frac{\rho_{lo} - \rho_k}{\rho_{lo} - \overline{\rho}}. \tag{106}$$

Assignment of T/S values for the depth interval $(0, p_{k+1})$ follows the above idea of assuming homogeneity between the surface and p_m. With the mixed layer depth previously determined by (105), lower sublayer values of T and S follow from the requirement that their column integrals be invariant during the redistribution process. Specifically, $T_{up} = \overline{T}$ and $S_{up} = \overline{S}$, where the overbar is the average over layers k through $k-1$. The lower sublayer values become

$$T_{lo} = \frac{T_k - \overline{T}q}{1-q}, \quad S_{lo} = \frac{S_k - \overline{S}q}{1-q}, \tag{107}$$

with q defined by (106).

Given the nonlinear nature of the equation of state, the upper and lower sublayer densities must be recomputed after T and S have been homogenized over layers 1 through $k-1$ and unmixed in layer k. Then the density profile, distinguished by homogeneous conditions between the surface and pressure p_m and a density discontinuity at that pressure, is provided to the K-T TKE algorithm to compute the new mixed layer depth. Temperature and salinity are homogenized over the new mixed layer depth, and the surface fluxes are spread over the same depth range. The resulting T and S profiles are then projected back onto the original hybrid coordinate profile. The sublayer information is not stored.

Note that the sublayer formation and deletion cycle alone does not cause a drift in the T/S profile given that layers 1 through $k-1$ are fully homogenized to start with. In other words, the original T/S profile is recovered if $\Delta t \to 0$. Initial experiments with this scheme show that assigning ρ_{lo} a value closer to ρ_k than to ρ_{k+1} gives the best results. To skip forming sublayers, set $\rho_{lo} = \rho_k$ (i.e., $q = 0$). The T/S profile transmitted to the K-T TKE algorithm is

obtained from the original profile by homogenizing T and S down to depth p_k and setting $p_m = p_k$.

Model 3: K-T Model (isopycnic vertical coordinates)

When HYCOM 2.2 is run with isopycnic vertical coordinates, the model automatically uses K-T model 3. The calculation of entrainment and detrainment rates from the TKE balance is well documented elsewhere but the detrainment algorithm is discussed here in comparison to the K-T models used with hybrid vertical coordinates. Detrainment is straightforward for hybrid vertical coordinates, but considerable difficulties arise because the MLB is not a vertical coordinate surface. Conversely, the isopycnic MLB is a vertical coordinate, but detrainment is difficult because the density of detrained water must equal the reference density of the layer into which it is detrained. Although the same TKE balance directs entrainment and detrainment, the performance of the K-T model differs considerably between the hybrid and isopycnic versions.

The mixed layer detrainment algorithm now relieves the numerical process of springtime mixed layer retreat at high latitudes where stabilizing buoyancy flux (TKE suppression) results mainly from freshwater input, ice melt, and increased rainfall rather than sensible heat input. In the original detrainment algorithm, heat entering the ocean during one model time step is spread over a layer of depth L, the Monin-Obukhov length. The old mixed layer depth h_1 is thereby divided into a new mixed layer of depth L and a somewhat cooler fossil mixed layer (FML) of depth $h_1 - L$. The FML is separated once more into two sublayers: the lower one cooled until its density meets that of the nearest (in density space) isopycnic layer, and the upper one heated to raise its temperature to that of the new mixed layer. The cooled sublayer is then detrained into the proper isopycnic layer while the heated sublayer is added to the new mixed layer, creating a final mixed layer. At this point the FML ceases to exist as an identifiable layer. Troubles arise with this scheme in near-freezing water where heat redistribution within the FML has little effect on density, particularly the lower sublayer density. If there is no suitable isopycnic coordinate layer to receive the wintertime mixed layer water, the model mixed layer may not recede. This issue prompted a refinement for the detrainment algorithm.

An interesting hypothesis for temperature treatment in the detrainment scheme is to examine how incoming fresh water is spread over the depth range L, followed by a redistribution of salinity in the FML. The goal is to lower S in the upper sublayer to the new mixed-layer value while using the extra salt to make the lower sublayer denser and therefore more easily detrainable. Trials show that polar mixed layer detrainment is not an issue if salinity is treated this way; however, an artificial salinity maximum in the seasonal thermocline may occasionally need to be generated. The strategy here is to allow downward transfer of S in the FML, given that salinity in the upper and lower sublayers remains in the range set by the old mixed layer and the model layer into which the particular sublayer eventually will merge.

This strategy, involving three layers- the receiving isopycnic layer plus the old and new mixed layers denoted by indices k, 1, and L, respectively- is expressed in the form of two constraints on the resulting salinities S_{lo}, S_{up} in the lower and upper sublayers of the FML:

$$\mathrm{S_{lo}} \leq \max(\mathrm{S_k}, \mathrm{S_1}) \quad (\text{constraint 1})$$

$$\min(\mathrm{S_1}, \mathrm{S_L}) \leq \mathrm{S_{up}} \quad (\text{constraint 2})$$

If the receiving layer is as fresh as or fresher than the old mixed layer ($S_k \leq S_1$), constraint 1 forbids downward salinity transfer in the FML, and detrainment must happen exactly as described in Bleck *et al.* (1992; Appendix E). The same is true if the new mixed layer is saltier than the old one ($S_L \leq S_1$). The following discussion is therefore limited to the case $S_L < S_1 < S_k$.

Setting $S_{lo} = S_k$ and $S_{up} = S_L$ maximizes salinity flux from the perspective of constraints 1 and 2, respectively. The option that bears the smallest salinity flux satisfies both constraints. An important side effect of the first option $S_{lo} = S_k$ is that the T, S properties of the lower sublayer match those of the receiving layer. Since detrainment under these circumstances is computationally clean, the detrainment algorithm explores this possibility first. The condition $S_{lo} = S_k$, together with the requirement $T_{up} = T_L$, distinctively determines S_{up}. If the latter value satisfies constraint 2, detrainment may proceed.

Consider how setting $S_{lo} = S_k$ violates constraint 2. Downward transfer of S in the FML is then controlled by letting $S_{up} = S_L$. The reduced salt flux in this case cannot make the lower sublayer as saline as the receiving layer. This mismatch requires using the detrainment algorithm described in Bleck *et al.* (1992; Appendix E).

Details of the detrainment algorithm are now described for the case $S_L < S_1 < S_k$. Let T_L, S_L represent the T, S values obtained by distributing thermal energy and freshwater input through the preceding time step over depth L, the new mixed layer. The procedure begins with an initially homogenous FML of thickness $h_1 - L$ distinguished by old mixed layer values T_1, S_1. (Note: The subscript 1 is used to denote values already modified by surface fluxes.) Downward transfer of heat and salt across an imaginary interface creates an upper sublayer characterized by T_{up}, S_{up} and a lower sublayer characterized by T_{dn}, S_{dn}. These four values are related to T_1, S_1 through

$$T_1 = \frac{T_{up}(h_1 - L - h) + T_{lo} h}{h_1 - L} \tag{108}$$

$$S_1 = \frac{S_{up}(h_1 - L - h) + S_{lo} h}{h_1 - L}, \tag{109}$$

where h is the thickness of the lower sublayer, i.e., the part of the FML to be detrained into receiving layer k.

If the process is controlled by constraint 1, set $T_{lo} = T_k$ and $S_{lo} = S_k$. Also $T_{up} = T_L$ regardless of the constraint limiting salinity transfer in the FML. It then follows from (108) that

$$h = (h_1 - L)\frac{T_{up} - T_1}{T_{up} - T_k}, \tag{110}$$

which, if substituted in (109), yields

$$S_{up} = \frac{S_1(h_1 - L) - S_k h}{h_1 - L - h} = S_1 + (S_1 - S_k)\frac{T_{up} - T_1}{T_1 - T_k}. \tag{111}$$

If this value violates constraint 2, an adapted version of the detrainment scheme of Bleck *et al.* (1992) must be invoked. In this case, salinity transfer is expanded by setting $S_{up} = \min(S_1, S_L)$ which, by way of (109), leads to

$$S_{lo} = \frac{S_1(h_1 - L) - S_{up}(h_1 - L - h)}{h}. \tag{112}$$

The analogous formula for T_{lo}, based on (108) and the requirement $T_{up} = T_L$, is

$$T_{lo} = \frac{T_1(h_1 - L) - T_L(h_1 - L - h)}{h}. \tag{113}$$

Notice that the value of h in (112) and (113) differs from the value in (110), and moreover is unknown at this point. Mixing the lower portion of the FML with layer k generates water of salinity

$$S_{new} = \frac{S_{lo}h + S_k h_k}{h + h_k} = \frac{S_{up}h + S_k h_k - (S_{up} - S_1)(h_1 - L)}{h + h_k} \tag{114}$$

and temperature

$$T_{new} = \frac{T_{lo}h + T_k h_k}{h + h_k} = \frac{T_L h + T_k h_k - (T_L - T_1)(h_1 - L)}{h + h_k}. \tag{115}$$

In agreement with Bleck *et al*. (1992; Appendix E), (114) and (115) are of the form

$$T_{new} = \frac{ah + b}{ch + d} \tag{116}$$

and

$$S_{new} = \frac{eh + f}{ch + d}. \tag{117}$$

The problem is finding h so that $\rho(T_{new}, S_{new}) = \rho_k$. Except for altering the definition of e and f, the procedure for finding h outlined in Bleck *et al*. (1992) remains unchanged.

5.2.7.5 GISS Mixed Layer Model

The Goddard Institute for Space Studies (GISS; Canuto *et al.*, 2001; 2002) mixed layer model is a level 2 Reynolds stress model where viscosity and *T*, *S* diffusivities are parameterized as

functions of the turbulent kinetic energy dissipation rate, the gradient Richardson number RiT, the Brunt-Vaisala frequency, and a density ratio $R\rho = \alpha\partial T/\partial z(\beta\partial S/\partial z)-1$. The GISS model parameterization is suitable for the following four cases (Canuto *et al.*, 2002):

1. doubly stable ($\partial T/\partial z>0$, $\partial S/\partial z<0$, $R\rho < 0$, $RiT > 0$),
2. doubly unstable ($\partial T/\partial z<0$, $\partial S/\partial z>0$, $R\rho > 0$, $RiT < 0$),
3. salt fingering ($\partial T/\partial z>0$, $\partial S/\partial z>0$, $R\rho > 0$,$RiT > 0$), and
4. diffusive convection ($\partial T/\partial z<0$, $\partial S/\partial z<0$, $R\rho > 0$, $RiT > 0$).

Equations for the second-order moments are solved to obtain diffusivity and viscosity coefficients (Canuto *et al.*, 2002). Both small-scale (unresolved) and large-scale (resolved) shear contributes to the gradient Richardson number RiT . The model is solved differently in two regimes depending on whether unresolved or resolved shear has the dominant influence on stability. The former regime symbolizes the comparatively quiescent ocean interior while the latter symbolizes the intense mixing of the surface boundary layer. The boundary between the two regimes is determined by whether RiT estimated from resolved shear alone surpasses a critical value. Different parameterizations of turbulent kinetic energy dissipation rate are used in these two regimes. Nonlocal effects are not parameterized.

After computing the viscosity, T diffusivity, and S diffusivity profiles at p grid points, the same implicit procedure used to solve the vertical diffusion equation for the KPP model is used for the GISS model. After mixing at p grid points, viscosity profiles are horizontally interpolated to u and v grid points, then the vertical diffusion equation is solved for the momentum components on their native grids (Halliwell, 2003).

5.2.8 HYCOM 2.2 Horizontal Mesh

HYCOM 2.2 employs a “C” grid of standard Cartesian coordinates with the x-axis pointing eastward and the y-axis pointing northward. The HYCOM 2.2 mesh is shown in Figure 1 below for pressure (P), velocity (U and V), and vorticity (Q) grid points. The grid in Figure 1 is for 7 x 7 pressure grid points. The grid meshes for the other variables have 8 x 8 grid points. All fields in this example would be dimensioned 8 x 8, with the eighth row and column unused for pressure grid point variables.

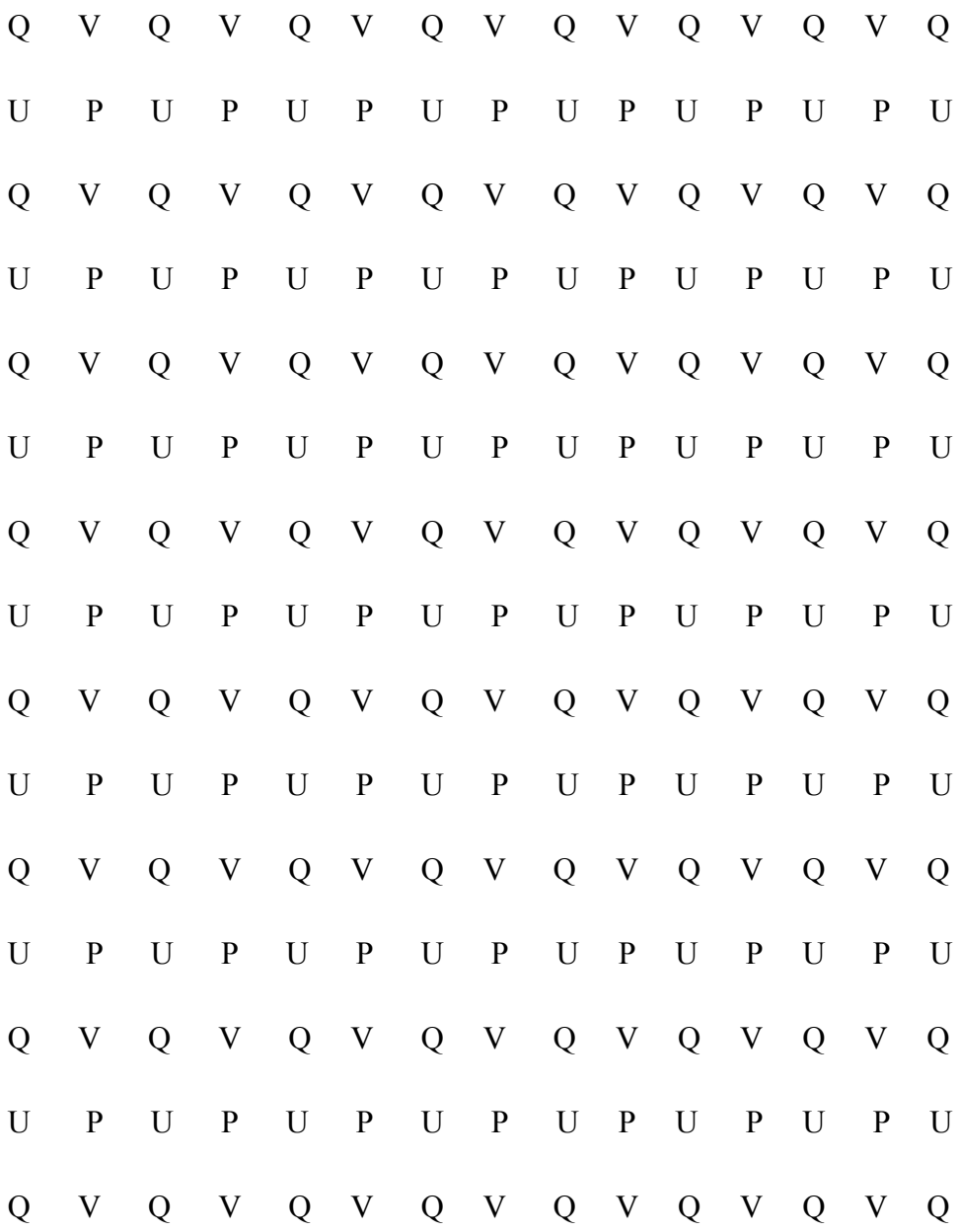


Figure 1: Example of a HYCOM 2.2 horizontal mesh grid.

5.2.9 HYCOM 2.2 Momentum Balance

The HYCOM 2.2 momentum equation has been designed to handle horizontally varying density in all model layers. Surface momentum flux accelerates only the fluid in layer 1 except when the Kraus-Turner mixed layer model is active, allowing momentum change to be distributed over the full mixed layer depth that was computed during the previous time step.

The Horizontal Pressure Gradient Force in Generalized Coordinates

The most significant feature of the HYCOM 2.2 momentum equation is the horizontal pressure gradient force (PGF). In a hydrostatic fluid ($\partial\varphi/\partial s = -\alpha\partial p/\partial s$), where ϕ is the geopotential, α denotes the specific volume, and s represents the generalized vertical coordinate, the layer mass-weighted PGF satisfies

$$\frac{\partial p}{\partial s}\left[\alpha\nabla_s p + \nabla_s\phi\right] = \nabla_s\left(\frac{\partial p}{\partial s}\alpha p\right) + \frac{\partial}{\partial s}\left(p\nabla_s\phi\right). \tag{118}$$

The 3D gradient term on the right shows that only boundary forces cause net accelerations of the fluid system. This has implications for vortex spinup and spindown. Given that the curl of the right side of (118) reduces to

$$\frac{\partial}{\partial s}\left(\nabla_s\times\nabla_s\phi\right).$$

The interface pressure torques controlling vortex spinup and spindown in individual s coordinate layers have the form $\left(\nabla_s\times p\nabla_s\phi\right)$. These properties must be preserved when numerically solving the fluid dynamics equations. Therefore we must find a finite-difference expression for the PGF term $\left[\alpha\nabla_s p + \nabla_s\phi\right]$ in the horizontal momentum equation that, after multiplying by layer pressure thickness, can be transformed by finite difference operations into an analog of the right side of (118). To demonstrate this, the x component of the rightmost term of (118) is written in the simplest, most plausible form, $\delta_s\left(\overline{p}^x\delta_x\phi\right)$. Finite difference product differentiation rules permit expansion as follows:

$$\delta_s\left(\overline{p}^x\delta_x\phi\right) = \left(\delta_s\overline{p}^x\right)\delta_x\overline{\phi}^x + \overline{p}^{xs}\delta_x\delta_s\phi$$

$$\delta_s\left(\overline{p}^x\delta_x\phi\right) = \left(\delta_s\overline{p}^x\right)\delta_x\overline{\phi}^x - \overline{p}^{xs}\delta_x\left(\alpha\delta_s p\right)$$

$$\delta_s\left(\overline{p}^x\delta_x\phi\right) = \left(\delta_s\overline{p}^x\right)\delta_x\overline{\phi}^x - \delta_x\left(\overline{p}^s\alpha\delta_s p\right) + \overline{\alpha\delta_s p}^x\delta_x\overline{p}^s.$$

A finite difference equation analogous to (118) comes by rearranging terms and adding the analogous equation for the *y* component:

$$\overline{\alpha\delta_s p}^x\delta_x\overline{p}^s + \left(\delta_s\overline{p}^x\right)\delta_x\overline{\phi}^s = \delta_x\left(\overline{p}^s\alpha\delta_s p\right) + \delta_s\left(\overline{p}^x\delta_x\phi\right)$$

$$\overline{\alpha\delta_s p}^y\delta_y\overline{p}^s + \left(\delta_s\overline{p}^y\right)\delta_y\overline{\phi}^s = \delta_y\left(\overline{p}^s\alpha\delta_s p\right) + \delta_s\left(\overline{p}^y\delta_y\phi\right).$$

In preserving the conservation properties expressed by (118), the finite-difference PGF term is evaluated in the form

$$\alpha\nabla_s p + \nabla_s\phi = \begin{pmatrix} \dfrac{\overline{\alpha\delta_s p}^x}{\overline{\delta_s p}^x}\delta_x\overline{p}^s + \delta_x\overline{\phi}^s \\ \dfrac{\overline{\alpha\delta_s p}^y}{\overline{\delta_s p}^y}\delta_y\overline{p}^s + \delta_y\overline{\phi}^s \end{pmatrix}. \tag{119}$$

The result is that writing the undifferentiated factor α in the PGF formula as simply $\overline{\alpha}^{x}$ or $\overline{\alpha}^{y}$ leads to spurious momentum and vorticity generation. Avoiding this requires that α be weighted by layer thickness in the PGF formula.

In isopycnic or quasi-isopycnic models, it is convenient to express the PGF in terms of the Montgomery potential $M = \phi + p\alpha$. The proper finite difference analog of M in a staggered vertical grid (p and ϕ carried on layer interfaces and α carried within layers) is

$$M = \overline{\phi}^{s} + \alpha \overline{p}^{s}.$$

The form $\delta_s\left(\alpha \overline{p}^{s}\right) = \overline{\alpha\delta_s p}^{s} + p\delta_s\alpha$ allows the s derivative of M to be expanded into

$$\delta_s M = p\delta_s\alpha + \overline{\delta_s\phi + \alpha\delta_s p}^{s},$$

from which the finite difference analogs of the two common forms of the hydrostatic equation can be extracted:

$$\frac{\partial\phi}{\partial p} = -\alpha \rightarrow \delta_s\phi = -\alpha\delta_s p$$

$$\frac{\partial M}{\partial \alpha} = p \rightarrow \delta_s M = p\delta_s\alpha.$$

The x component of (119) becomes

$$\frac{\overline{\alpha\delta_s p}^{x}}{\overline{\delta_s p}^{x}}\delta_x \overline{p}^{s} + \delta_x \overline{\phi}^{s} = \delta_x M + \left[\frac{\overline{\alpha\delta_s p}^{x}}{\overline{\delta_s p}^{x}}\delta_x \overline{p}^{s} - \delta_x\left(\alpha \overline{p}^{s}\right)\right]. \tag{120}$$

Making use of the relation

$$\overline{AB}^{x} - \overline{A}^{x}\overline{B}^{x} = \frac{1}{4}\left(\delta'_x A\right)\left(\delta'_x B\right),$$

where δ'_x is the difference between two neighboring grid points; i.e., $\delta'_x = \Delta x\delta_x$, the term inside square brackets in (120) can be expanded into

$$\frac{1}{\overline{\delta_s p}^{x}}\left(\overline{\alpha\delta_s p}^{x} - \overline{\alpha}^{x}\overline{\delta_s p}^{x}\right)\delta_x \overline{p}^{s} - \overline{p}^{sx}\delta_x\alpha = \frac{1}{4\overline{\delta_s p}^{x}}\left(\delta'_x \delta_s p\right)\left(\delta'_x \alpha\right)\delta_x \overline{p}^{s} - \overline{p}^{sx}\delta_x\alpha$$

$$\frac{1}{\overline{\delta_s p}^{x}}\left(\overline{\alpha\delta_s p}^{x} - \overline{\alpha}^{x}\overline{\delta_s p}^{x}\right)\delta_x \overline{p}^{s} - \overline{p}^{sx}\delta_x\alpha = \frac{1}{4\overline{\delta_s p}^{x}}\left[\left(\delta'_x \delta_s p\right)\left(\delta'_x \overline{p}^{s}\right) - 4\overline{p}^{sx}\delta_s \overline{p}^{x}\right]\delta_x\alpha.$$

The expression above includes a total of four p points located one grid distance Δx apart on two consecutive s surfaces. Simplification is possible by labeling the four grid points as

$$p_1 = p\left(x - \frac{\Delta x}{2}, s - \frac{\Delta s}{2}\right) \quad p_2 = p\left(x + \frac{\Delta x}{2}, s - \frac{\Delta s}{2}\right)$$

$$p_3 = p\left(x - \frac{\Delta x}{2}, s + \frac{\Delta s}{2}\right) \quad p_4 = p\left(x + \frac{\Delta x}{2}, s + \frac{\Delta s}{2}\right)$$

With a little algebra, the term within square brackets in (120) reduces to

$$\frac{p_1 p_2 - p_3 p_4}{(p_4 - p_2) + (p_3 - p_1)} \delta_x \alpha.$$

Substitution of this expression inside the brackets of (120) gives the sought-after expression for the x component of the PGF force. An analog can be derived for the y component.

5.2.10 Ocean Heat Content (OHC) Balance

Shay *et al*. (2000) described oceanic heat content (OHC) relative to the depth of the 26°C isotherm. It is a better indicator than SST of the potential for TC intensification. Leipper and Volgenau (1972) estimated OHC as:

$$\mathrm{C} = \frac{c_p}{g} \sum_{k=1}^{N} \left[\max(T_k, 26) - 26 \right] \delta p_k. \tag{121}$$

Potential Temperature Balance within Individual Model Layers

HYCOM 2.2 equations are written in x, y, s, t space, where s is a generalized vertical coordinate. In this structure, the vertically integrated HYCOM 2.2 equation for the conservation of potential temperature within model layer k (Bleck, 2002) is

$$\begin{aligned} \frac{\partial}{\partial t_s}(\theta \Delta p)_k = &-\nabla_s \cdot (\mathbf{v} \theta \Delta p)_k - \left(\dot{s} \theta \frac{\partial p}{\partial s} \right)_{k+1} + \left(\dot{s} \theta \frac{\partial p}{\partial s} \right)_k + \theta_k \nabla_s \cdot \left[K_{H\Delta p} \nabla (\Delta p_k) \right] \\ &+ \nabla_s \cdot (K_{H\vartheta} \Delta p_k \nabla_s \theta_k) - \left(K_{V\theta} \frac{\partial \theta_k}{\partial s} \right)_{k+1} + \left(K_{V\theta} \frac{\partial \theta_k}{\partial s} \right)_k - (Q_k - Q_{k+1}), \end{aligned} \tag{122}$$

where $\Delta p_k = p_{k+1} - p_k$ is layer thickness, $\dot{s} \theta \partial p / \partial s$ denotes temperature flux across an s interface associated with mass flux $\dot{s} \partial p / \partial s$, $K_{H\vartheta}, K_{V\theta}$ are horizontal and vertical diffusion coefficients as a result of subgrid-scale velocity fluctuations, $K_{H\Delta p}$ represents the horizontal thickness diffusion coefficient, and Q is diabatic source terms. Multiplying (122) by c_p / g gives the heat balance within layer k. To derive the potential temperature balance, (122) is rearranged as

$$\begin{aligned} (\Delta p)_k \frac{\partial \theta_k}{\partial t_s} = &-\theta_k \frac{\partial}{\partial t} (\Delta p)_k - (\mathbf{v} \Delta p)_k \cdot \nabla_s \theta_k - \theta_k \nabla_s \cdot (\mathbf{v} \Delta p)_k - \left(\dot{s} \theta \frac{\partial p}{\partial s} \right)_{k+1} + \left(\dot{s} \theta \frac{\partial p}{\partial s} \right)_k \\ &+ \nabla_s \cdot (K_{H\vartheta} \Delta p_k \nabla_s \theta_k) + \theta_k \nabla_s \cdot \left[K_{H\Delta p} \nabla (\Delta p_k) \right] \\ &- \left(K_{V\theta} \frac{\partial \theta_k}{\partial s} \right)_{k+1} + \left(K_{V\theta} \frac{\partial \theta_k}{\partial s} \right)_k - (Q_k - Q_{k+1}), \end{aligned} \tag{123}$$

and the first term on the right side of (123) is evaluated with the vertically integrated HYCOM 2.2 continuity (thickness tendency) equation for layer k (Bleck, 2002):

$$\frac{\partial}{\partial t_s} \Delta p_k = -\nabla_s \cdot (\mathbf{v} \Delta p)_k - \left(\dot{s} \frac{\partial p}{\partial s} \right)_{k+1} + \left(\dot{s} \frac{\partial p}{\partial s} \right)_k + \nabla_s \cdot (K_{H\Delta p} \Delta p_k). \tag{124}$$

Multiplying (124) by θ_k substituting into (123) gives the HYCOM 2.2 potential temperature balance within layer *k*:

$$\begin{aligned}(\Delta p)_k \frac{\partial \theta_k}{\partial t_s} = & -(\mathbf{v}\Delta p)_k \cdot \nabla_s \theta_k \\ & -\left(\dot{s}\theta \frac{\partial p}{\partial s}\right)_{k+1} + \left(\dot{s}\theta \frac{\partial p}{\partial s}\right)_k + \theta_k \left(\dot{s} \frac{\partial p}{\partial s}\right)_{k+1} - \theta_k \left(\dot{s} \frac{\partial p}{\partial s}\right)_k \\ & +\nabla_s \cdot \left(K_{H\vartheta} \Delta p_k \nabla_s \theta_k\right) \\ & -\left(K_{V\theta} \frac{\partial \theta}{\partial s}\right)_{k+1} + \left(K_{V\theta} \frac{\partial \theta}{\partial s}\right)_k \\ & +\left(Q_k - Q_{k+1}\right).\end{aligned} \tag{125}$$

The potential temperature is modified by horizontal advection (top line, right side); temperature flux related to mass flux across model interfaces (second line); horizontal diffusion (third line); vertical diffusion (fourth line); and vertical heat flux divergence (bottom line). It is simple to estimate the contribution of all of these terms during a model run. However, interpretation of the contributions of vertical advection and vertical diffusion to temperature change in each layer [lines 2 and 4 of (125)] is not as direct in the hybrid coordinate system. Subroutine *hybgen* computes layer temperature change due to mass flux across layer interfaces [line 2 in (125)] while vertical mixing routines **mx*.f** [line 4 in (125)] compute temperature change due to vertical diffusion across layer interfaces. Together these terms explain the total influence of vertical temperature flux on layer potential temperature.

Interpreting the way these terms [lines 2 and 4 of (125)] contribute to the total influence of vertical temperature flux differs drastically between a model layer in the surface *p* coordinate domain and one in the interior isopycnic domain. In the surface *p* domain, the vertical flux terms of line 2 in (125) account for the donation of vertical advection of temperature across layer interfaces. Advection affects temperature in each layer by creating vertical fluxes across interfaces. Vertical advection in the *p* domain is accounted for by first making the continuity equation solver adjust layer thickness with respect to horizontal divergence patterns and thickness diffusion (e.g., in response to local vertical velocity). This does not alter temperature in each model layer. The hybrid coordinate generator then returns model interfaces back to their original depths. Resulting mass fluxes across interfaces ($\dot{s} \neq 0$) produce the layer temperature changes caused by vertical advection. For instance, consider the surface *p* layers at the equator where equatorial upwelling is present. At every time step, the continuity equation moves interfaces upward in the water column without changing the potential temperature in the layers. The hybrid coordinate generator then shifts the interfaces back down to their original depths and cools the temperature within each layer due to the resulting upward fluxes ($\dot{s} > 0$).

In the isopycnic interior, the hybrid coordinate generator does not try to relocate model interfaces to their original depths, but only attempts to shift layers vertically if necessary to restore isopycnic conditions. Interior vertical mixing and penetrating shortwave radiation are the sole processes driving layer density away from the target isopycnic values, causing *hybgen* to move model layers. Since shortwave radiation is usually irrelevant in the isopycnic interior,

layer temperature changes generated by *hybgen* occur only in response to interior diapycnal mixing. Therefore, the combined contribution of lines 2 and 4 of (125) accounts for the layer temperature change due solely to vertical diffusion. Vertical advection does not alter layer temperature in the isopycnic interior.

5.2.11 HYCOM 2.2 Surface Fluxes

HYCOM 2.2 surface fluxes of heat and mass, except for shortwave thermal radiation, are absorbed in model layer 1, which is the full surface mixed layer if HYCOM 2.2 is run in isopycnic coordinate mode. Shortwave radiation penetrates to deeper layers, with penetration depth depending on water clarity. The two-component (red and blue light) exponential decay model of Jerlov (1976) computes the penetrating shortwave radiation in HYCOM 2.2. If HYCOM 2.2 is in isopycnic coordinate mode, or if the simple Kraus-Turner (KT) mixed layer model 2 (see Section 5.2.7.4) is used, all shortwave radiation is absorbed in the mixed layer. If full KT mixed layer model 1 is used, penetrating shortwave radiation may be summoned as an option; otherwise, all shortwave radiation is absorbed in the mixed layer. Penetrating shortwave radiation is always included in KPP mixing and will be for all non-slab mixed layer models added in the future.

Penetration depth is a function of water clarity, represented by the Jerlov water type. The water type is allocated integer values from 1 through 5, with 1 being the clearest water. Given the incoming shortwave radiation flux S_0 at the surface, the flux passing through model interface k located at pressure p_k is

$$S_k = S_0\left[r\exp\left(\frac{-p_{k+1}}{\beta_R}\right) + (1-r)\exp\left(\frac{-p_{k+1}}{\beta_B}\right)\right],$$

where r is the fraction of red light, β_R is the penetration depth scale of red light, and β_B is the penetration depth scale of blue light. The parameters for all five Jerlov water types are summarized in Table 1:

Jerlov Water Type	r	β_R	β_B
1	0.58	0.35	23.0
2	0.62	0.60	20.0
3	0.67	1.00	17.0
4	0.77	1.50	14.0
5	0.78	1.40	7.9

Table 1: Jerlov water type parameters.

In HYCOM 2.2, the user selects the Jerlov water type for each grid point at the start of the model run. Future versions may see water type determined by biological or suspended sediment models.

HYCOM 2.2 features two choices for bulk parameterization of surface fluxes. The first is the standard constant bulk coefficients. The second is the sophisticated parameterization scheme of Kara *et al*. (2000), extensively tested and embedded into HYCOM 2.2 by researchers at NRL. The user also has the option of relaxing near surface temperature or salinity to climatology.

5.2.12 HYCOM 2.2 Implicit Vertical Diffusion Equation

Vertical diffusion is solved at each model grid point after model variables are updated by the continuity equation, horizontal advection and diffusion, momentum equation, and surface fluxes. It is therefore treated as a purely 1D issue with zero-flux boundary conditions at the surface and bottom. In HYCOM 2.2, this equation is solved for the KPP, GISS and Mellor Yamada mixing algorithms. It is solved for the KPP-like interior diapycnal-mixing algorithm when using the Kraus-Turner mixed layer model. This solution algorithm will be used with all future mixing models incorporated into HYCOM 2.2 that calculate vertical diffusivity/viscosity profiles at model interfaces.

Decomposing model variables into mean (denoted by an overbar) and turbulent (denoted by a prime) components, the vertical diffusion equations for potential temperature, salinity, and vector momentum are

$$\frac{\partial\overline{\theta}}{\partial t}=-\frac{\partial}{\partial z}\overline{w'\theta'}\quad\frac{\partial\overline{S}}{\partial t}=-\frac{\partial}{\partial z}\overline{w'S'}\quad\frac{\partial\overline{\mathbf{v}}}{\partial t}=-\frac{\partial}{\partial z}\overline{w'\mathbf{v}'}. \tag{126}$$

Boundary layer diffusivities and viscosity are parameterized as follows:

$$\overline{w'\theta'}=-K_\theta\left(\frac{\partial\overline{\theta}}{\partial z}+\gamma_\theta\right),\quad\overline{w'S'}=-K_S\left(\frac{\partial\overline{S}}{\partial z}+\gamma_S\right),\quad\overline{w'\mathbf{v}'}=-K_m\left(\frac{\partial\overline{\mathbf{v}}}{\partial z}+\gamma_m\right), \tag{127}$$

where the γ terms are nonlocal fluxes. For instance, the KPP model has nonlocal terms for θ and S, but not for momentum. The following solution procedure functions for any mixing model in HYCOM 2.2 computing the diffusivity/viscosity profiles at model interfaces, regardless of whether nonlocal terms are parameterized.

The following matrix equations are formulated and solved:

$$\mathbf{A_T}\Theta^{t+1}=\Theta^t+\mathbf{H}_\Theta\quad\mathbf{A_S}\mathbf{S}^{t+1}=\mathbf{S}^t+\mathbf{H_S}\quad\mathbf{A_M}\mathbf{M}^{t+1}=\mathbf{M}^t+\mathbf{H_M}, \tag{128}$$

where superscripts $t, t+1$ are model times, and $\mathbf{M}$ denotes the vector of a momentum component, either u or v. The matrices $\mathbf{A}$ are tri-diagonal coefficient matrices, and the vectors $\mathbf{H_T}$ and $\mathbf{H_S}$ are the nonlocal flux terms. Given K model layers with nonzero thickness, where an individual layer k of thickness $(\delta p)_k$ (in meters) is bounded above and below by interfaces at pressure depths p_k and p_{k+1} (also in meters), the matrix $\mathbf{A_S}$ is found by:

$$\begin{aligned}
\mathbf{A}_{\mathbf{S}}^{1,1} &= \left(1+\Omega_{S1}^{+}\right) \\
\mathbf{A}_{\mathbf{S}}^{k,k-1} &= -\Omega_{Sk}^{-} \qquad 2 \le k \le \mathrm{K} \\
\mathbf{A}_{\mathbf{S}}^{k,k} &= \left(1+\Omega_{Sk}^{-}+\Omega_{Sk}^{+}\right) \qquad 2 \le k \le \mathrm{K} \\
\mathbf{A}_{\mathbf{s}}^{k,k+1} &= -\Omega_{Sk}^{+} \qquad 1 \le k \le \mathrm{K}-1
\end{aligned} \quad , \tag{129}$$

with

$$\begin{aligned}
\Omega_{Sk}^{-} &= \frac{\Delta t}{\left(\delta p\right)_k} \frac{K_S\left(p_k\right)}{\left(p_{k+0.5}-p_{k-0.5}\right)} \\
\Omega_{Sk}^{+} &= \frac{\Delta t}{\left(\delta p\right)_k} \frac{K_S\left(p_{k+1}\right)}{\left(p_{k+1.5}-p_{k+0.5}\right)}
\end{aligned} \quad , \tag{130}$$

where $p_{k+0.5}$ is the central pressure depth of model layer k in meters. The nonlocal flux arrays are computed using

$$\begin{aligned}
H_{S1} &= \frac{\Delta t}{\left(\delta p\right)_1} K_S\left(p_{k+1}\right)\gamma_S\left(p_{k+1}\right) \\
H_{Sk} &= \frac{\Delta t}{\left(\delta p\right)_k}\left[K_S\left(p_{k+1}\right)\gamma_S\left(p_{k+1}\right)-K_S\left(p_k\right)\gamma_S\left(p_k\right)\right] \quad 2 \le k \le \mathrm{K}.
\end{aligned} \tag{131}$$

The solution comes from inverting the tri-diagonal matrix $\mathbf{A}$. The matrix equations are formulated and solved for potential temperature and momentum components in the same manner.

5.2.13 Kinematic Vertical Velocity

The vertical velocity w in Cartesian coordinates is found by vertically integrating the continuity equation

$$\left(\frac{dw}{dz}\right)_z = -\nabla_z \cdot \mathbf{v} \tag{132}$$

downward from the surface, where subscripts represent the variable held constant during partial differentiation. HYCOM 2.2 variables are stored on a non-Cartesian $\left(x, y, s\right)$ coordinate system, where the generalized vertical coordinates are surfaces of constant s, usually density in the ocean interior and fixed pressure levels near the sea surface and in shallow coastal regions. Using equation (132) to derive w profiles at HYCOM 2.2 grid points, horizontal velocity components must be re-gridded to constant z levels before integrating (132) downward from the surface. Re-gridding must be executed at high vertical resolution to present accurate vertical profiles of w.

Since high-resolution re-gridding is time consuming, a formula is given here to estimate w profiles directly from fields stored on the HYCOM 2.2 generalized coordinate system. This formula is incorporated into the HYCOM 2.2 post-processing program, **hycomproc**, to approximate w profiles from fields stored on model archives. Another formula is then derived to compute w for advection of 3D Lagrangian floats through integration of the HYCOM 2.2 continuity (thickness tendency) equation downward from the surface and pairing it with the

previously-derived w profile equation.

HYCOM 2.2 Post-Processing Equation for Vertical Velocity Profiles

Because HYCOM 2.2 equations use pressure units for the vertical coordinate, vertical velocity is defined as

$$w = \frac{dp}{dt}. \tag{133}$$

By changing the vertical coordinate in (132) from z to p, we obtain

$$\left(\frac{dw}{dp}\right)_s = -\nabla_s \cdot \mathbf{v}. \tag{134}$$

Because the HYCOM 2.2 generalized coordinate system is not Cartesian, integration of (134) downward from the surface initiates additional terms related to the sloping s interfaces. The vertical discretization has layers $k = 1,2,...,N$, with every layer k bounded by vertical coordinate surfaces at pressure depths $p_k(x,y,s)$ above and $p_{k+1}(x,y,s)$ below. From this point on, s is held constant in partial differentiation and the subscript s is dropped.

Assuming $w = 0$ at the surface, vertical velocity at the base of model layer $k = 1$ is

$$w\left(p_2^-\right) = -\int_{p_1}^{p_2^-}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)dp = -\left(p_2^- - p_1\right)\left(\frac{\partial u_1}{\partial x}+\frac{\partial v_1}{\partial y}\right), \tag{135}$$

where the integration is executed from the surface down to a minuscule distance above interface 2. To find vertical velocity at the top of layer 2, the continuity equation is integrated across interface 2 from p_2^- to p_2^+:

$$w\left(p_2^+\right) = -\int_{p_1}^{p_2^-}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)dp - \int_{p_2^-}^{p_2^+}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)dp. \tag{136}$$

If interface 2 is not level, there is a jump condition in the evaluation of the rightmost integral in (136). This jump condition is attained by evaluating the rightmost integral as illustrated for the x direction in Figure 3. The x derivative of u is given by

$$\frac{\delta u}{\delta x} = \frac{u_2 + \frac{\partial u_2}{\partial x}\left(\frac{\delta x}{2}\right) - u_1 - \frac{\partial u_1}{\partial x}\left(-\frac{\delta x}{2}\right)}{\delta x}. \tag{137}$$

The vertical velocity jump across the interface is numerically evaluated by

$$w\left(p_2^+\right) - w\left(p_2^-\right) = \frac{\delta p}{\delta x}\left[u_2 + \frac{\partial u_2}{\partial x}\left(\frac{\delta x}{2}\right) - u_1 + \frac{\partial u_1}{\partial x}\left(\frac{\delta x}{2}\right)\right], \tag{138}$$

where $\delta p = p_2^+ - p_2^-$. In the limit as the box defined by δx and δp in Figure 3 contracts to zero area, $\delta p/\delta x \to \partial p/\partial x$ and $\delta x/2 \to 0$. Thus, $w\left(p_2^+\right)$ from (136) becomes, after adding the jump condition in the y direction

$$w\left(p_2^+\right) = -(p_2 - p_1)\left(\frac{\partial u_1}{\partial x}+\frac{\partial v_1}{\partial y}\right) + (u_2 - u_1)\frac{\partial p_2}{\partial x} + \left(v_2 - v_1\right)\frac{\partial p_2}{\partial y}. \tag{139}$$

Continuing the integration downward, the vertical velocity at a pressure level P within model layer $n \geq 2$, where

$$P = p_n + q\left(p_{n+1} - p_n\right) \quad , \quad 0 < q < 1, \tag{140}$$

is

$$w(P) = -\sum_{k=1}^{n-1}\left(p_{k+1} - p_k\right)\left(\frac{\partial u_k}{\partial x} + \frac{\partial v_k}{\partial y}\right) - q\left(p_{n+1} - p_n\right)\left(\frac{\partial u_n}{\partial x} + \frac{\partial v_n}{\partial y}\right) + \sum_{k=2}^{n}\left[\left(u_k - u_{k-1}\right)\frac{\partial p_k}{\partial x} + \left(v_k - v_{k-1}\right)\frac{\partial p_k}{\partial y}\right]. \tag{141}$$

It is simple to prove that

$$w(P) = w\left(p_n^+\right) + q\left[w\left(p_{n+1}^-\right) - w\left(p_n^+\right)\right]. \tag{142}$$

Therefore, w varies linearly in the vertical in each layer while discontinuities may exist at interfaces. Equation (141) is used to evaluate w in the HYCOM 2.2 post-processing program (**hycomproc**).

Equation (141) is validated by giving the correct bottom vertical velocity when integrated from the surface to the bottom. If layer N is intersecting the bottom (the deepest layer with nonzero thickness), then Equation (141) gives the following expression for vertical velocity at the bottom:

$$w\left(p_b\right) = -\sum_{k=1}^{N}\left(p_{k+1} - p_k\right)\left(\frac{\partial u_k}{\partial x} + \frac{\partial v_k}{\partial y}\right) + \sum_{k=2}^{N}\left[\left(u_k - u_{k-1}\right)\frac{\partial p_k}{\partial x} + \left(v_k - v_{k-1}\right)\frac{\partial p_k}{\partial y}\right], \tag{143}$$

where $p_b = p_{N+1}$ is the bottom pressure depth. For this bottom velocity to be true, it must equal the bottom velocity derived from the continuity equation for barotropic velocity. Defining barotropic velocity components $\bar{u}, \bar{v}$ as vertical averages from the surface to the bottom and assuming surface vertical velocity is zero, we get the following bottom vertical velocity from the barotropic continuity equation:

$$w\left(p_b\right) = -p_b\left(\frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y}\right). \tag{144}$$

The barotropic vertical velocity components are given by

$$\begin{aligned} \bar{u} &= \frac{1}{p_b}\sum_{k=1}^{N} u_k\left(p_{k+1} - p_k\right) \\ \bar{v} &= \frac{1}{p_b}\sum_{k=1}^{N} v_k\left(p_{k+1} - p_k\right) \end{aligned}. \tag{145}$$

Equation (143) is obtained by substituting (145) into (144), which validates (141).

It is necessary to estimate w during HYCOM 2.2 runs for the purpose of vertically advecting synthetic floats. This computation becomes more efficient by taking advantage of computations already made during simulations, specifically the time evolution of layer k thickness computed by the continuity (thickness tendency) equation. The continuity equation is integrated downward from the surface and combined with equation (141) to produce the expression used to estimate w during model simulations.

If sub-grid scale processes (thickness diffusion) are neglected, the thickness tendency is given

by (Bleck, 2002):

$$\left[\frac{\partial}{\partial t}\left(\Delta p_k\right)\right]_s = -\nabla_s \cdot \left(\mathbf{v}_k \Delta p_k\right) - \left(\dot{s}\frac{\partial p}{\partial s}\right)_{k+1} + \left(\dot{s}\frac{\partial p}{\partial s}\right)_k, \tag{146}$$

where $\left(\dot{s}\partial p/\partial s\right)_k$ denotes diapycnal vertical velocity in pressure per unit time across interface k. The subscripts s demonstrate that the generalized vertical coordinate is constant during partial differentiation. Equation (146) is summed downward from the surface presuming that the surface interface is stationary. The interface vertical motion, with the subscript s again dropped, is

$$\frac{\partial p_2}{\partial t} = -\left(p_2 - p_1\right)\left(\frac{\partial u_1}{\partial x} + \frac{\partial v_1}{\partial y}\right) - u_1\left(\frac{\partial p_2}{\partial x} - \frac{\partial p_1}{\partial x}\right) - v_1\left(\frac{\partial p_2}{\partial y} - \frac{\partial p_1}{\partial y}\right) - \left(\dot{s}\frac{\partial p}{\partial s}\right)_2. \tag{147}$$

Continuing to interface 3,

$$\frac{\partial p_3}{\partial t} = \frac{\partial p_3}{\partial t} + \frac{\partial}{\partial t}\left(\Delta p_2\right), \tag{148}$$

which gives

$$\begin{aligned}\frac{\partial p_3}{\partial t} = &-\left(p_2 - p_1\right)\left(\frac{\partial u_1}{\partial x} + \frac{\partial v_1}{\partial y}\right) - u_1\left(\frac{\partial p_2}{\partial x} - \frac{\partial p_1}{\partial x}\right) - v_1\left(\frac{\partial p_2}{\partial y} - \frac{\partial p_1}{\partial y}\right)\\ &-\left(p_3 - p_2\right)\left(\frac{\partial u_2}{\partial x} + \frac{\partial v_2}{\partial y}\right) - u_2\left(\frac{\partial p_3}{\partial x} - \frac{\partial p_2}{\partial x}\right) - v_2\left(\frac{\partial p_3}{\partial y} - \frac{\partial p_2}{\partial y}\right) - \left(\dot{s}\frac{\partial p}{\partial s}\right)_3.\end{aligned} \tag{149}$$

Rearranging terms produces

$$\begin{aligned}\frac{\partial p_3}{\partial t} = &-\left(p_2 - p_1\right)\left(\frac{\partial u_1}{\partial x} + \frac{\partial v_1}{\partial y}\right) - \left(p_3 - p_2\right)\left(\frac{\partial u_2}{\partial x} + \frac{\partial v_2}{\partial y}\right)\\ &+\left(u_2 - u_1\right)\frac{\partial p_2}{\partial x} + \left(v_2 - v_1\right)\frac{\partial p_2}{\partial y} - u_2\frac{\partial p_3}{\partial x} - v_2\frac{\partial p_3}{\partial y} - \left(\dot{s}\frac{\partial p}{\partial s}\right)_3.\end{aligned} \tag{150}$$

In general, the vertical motion of interface $n+1$ at the base of layer $n \geq 2$ is

$$\begin{aligned}\frac{\partial p_{n+1}}{\partial t} = &-\sum_{k=1}^{n}\left(p_{k+1} - p_k\right)\left(\frac{\partial u_k}{\partial x} + \frac{\partial v_k}{\partial y}\right)\\ &+\sum_{k=2}^{n}\left[\left(u_k - u_{k-1}\right)\frac{\partial p_k}{\partial x} + \left(v_k - v_{k-1}\right)\frac{\partial p_k}{\partial y}\right]\\ &-u_n\frac{\partial p_{n+1}}{\partial x} - v_n\frac{\partial p_{n+1}}{\partial y} - \left(\dot{s}\frac{\partial p}{\partial s}\right)_{n+1}.\end{aligned} \tag{151}$$

The interface vertical velocity at pressure depth P within layer n, with P given by (140), is

$$\frac{\partial P}{\partial t}=\frac{\partial p_n}{\partial t}+q\left(\frac{\partial p_{n+1}}{\partial t}-\frac{\partial p_n}{\partial t}\right)=$$
$$-\left\{\left(\dot{s}\frac{\partial p}{\partial s}\right)_n+q\left[\left(\dot{s}\frac{\partial p}{\partial s}\right)_{n+1}-\left(\dot{s}\frac{\partial p}{\partial s}\right)_n\right]\right\}$$
$$-\sum_{k=1}^{n-1}\left(p_{k+1}-p_k\right)\left(\frac{\partial u_k}{\partial x}+\frac{\partial v_k}{\partial y}\right)-q\left(p_{n+1}-p_n\right)\left(\frac{\partial u_n}{\partial x}+\frac{\partial v_n}{\partial y}\right) \tag{152}$$
$$+\sum_{k=2}^{n}\left[\left(u_k-u_{k-1}\right)\frac{\partial p_k}{\partial x}+\left(v_k-v_{k-1}\right)\frac{\partial p_k}{\partial y}\right]$$
$$-u_n\left[\frac{\partial p_n}{\partial x}+q\left(\frac{\partial p_{n+1}}{\partial x}-\frac{\partial p_n}{\partial x}\right)\right]-v_n\left[\frac{\partial p_n}{\partial y}+q\left(\frac{\partial p_{n+1}}{\partial y}-\frac{\partial p_n}{\partial y}\right)\right].$$

From (141), the third and fourth lines of (152) are known as the fluid vertical velocity w at pressure depth P. As a result, (152) becomes

$$w(P)=\frac{\partial p_n}{\partial t}+q\left(\frac{\partial p_{n+1}}{\partial t}-\frac{\partial p_n}{\partial t}\right)+\left\{\left(\dot{s}\frac{\partial p}{\partial s}\right)_n+q\left[\left(\dot{s}\frac{\partial p}{\partial s}\right)_{n+1}-\left(\dot{s}\frac{\partial p}{\partial s}\right)_n\right]\right\} \tag{153}$$
$$+u_n\left[\frac{\partial p_n}{\partial x}+q\left(\frac{\partial p_{n+1}}{\partial x}-\frac{\partial p_n}{\partial x}\right)\right]+v_n\left[\frac{\partial p_n}{\partial y}+q\left(\frac{\partial p_{n+1}}{\partial y}-\frac{\partial p_n}{\partial y}\right)\right].$$

The vertical velocity of model pressure interfaces can be divided as follows:

$$\frac{\partial p_k}{\partial t}=\frac{\partial \hat{p}_k}{\partial t}-\left(\dot{s}\frac{\partial p}{\partial s}\right)_k. \tag{154}$$

If the diapycnal vertical velocity is zero, the interface vertical velocity equals $\partial \hat{p}_k/\partial t$, which may be interpreted as the local vertical velocity of a material surface. Since p_k and $\hat{p}_k$ surfaces are co-located when vertical velocity is evaluated, equation (153) can be written as

$$w(P)=\frac{\partial \hat{p}_n}{\partial t}+q\left(\frac{\partial \hat{p}_{n+1}}{\partial t}-\frac{\partial \hat{p}_n}{\partial t}\right)$$
$$+u_n\left[\frac{\partial p_n}{\partial x}+q\left(\frac{\partial p_{n+1}}{\partial x}-\frac{\partial p_n}{\partial x}\right)\right]+v_n\left[\frac{\partial p_n}{\partial y}+q\left(\frac{\partial p_{n+1}}{\partial y}-\frac{\partial p_n}{\partial y}\right)\right]. \tag{155}$$

The first term on the right side of (155) is the vertically interpolated material surface vertical velocity (the vertical velocity of s surfaces in the absence of diapycnal mass fluxes). The other two terms on the right side are the vertical component of layer k flow when the layer is not flat. It is a function of momentum components in layer n and the slope of the interfaces at the top and bottom of layer n, the latter vertically interpolated to pressure depth P. The vertical velocities at the top and bottom of layer n are obtained by making $q=0$ and $q=1$, respectively:

$$w\left(p_n^+\right)=\frac{\partial \hat{p}_n}{\partial t}+u_n\left(\frac{\partial p_n}{\partial x}\right)+v_n\left(\frac{\partial p_n}{\partial y}\right)$$
$$w\left(p_{n+1}^-\right)=\frac{\partial \hat{p}_{n+1}}{\partial t}+u_n\left(\frac{\partial p_{n+1}}{\partial x}\right)+v_n\left(\frac{\partial p_{n+1}}{\partial y}\right). \quad (156)$$

Vertical velocity at the central depth of layer n comes by setting $q=1/2$:

$$w(P)=\frac{1}{2}\left(\frac{\partial \hat{p}_n}{\partial t}+\frac{\partial \hat{p}_{n+1}}{\partial t}\right)+\frac{u_n}{2}\left(\frac{\partial p_n}{\partial x}+\frac{\partial p_{n+1}}{\partial x}\right)+\frac{v_n}{2}\left(\frac{\partial p_n}{\partial y}+\frac{\partial p_{n+1}}{\partial y}\right). \quad (157)$$

From Equation (155), the vertical velocity at the ocean bottom reduces to

$$w\left(p_b\right)=u_b\frac{\partial p_b}{\partial x}+v_b\frac{\partial p_b}{\partial y}, \quad (158)$$

where p_b is bottom pressure and u_b, v_b are momentum components in the deepest model layer with nonzero thickness.

Estimating vertical velocities from Equations (155) through (157) requires evaluating $\partial \hat{p}_k/\partial t$ at all model interfaces k. Solving the thickness diffusion equation (146) with diapycnal vertical velocity set to zero gives:

$$\left[\frac{\partial}{\partial t}\left(\Delta \hat{p}_k\right)\right]_s=-\nabla_s\cdot\left(\mathbf{v}_k\Delta \hat{p}_k\right). \quad (159)$$

An advantage of estimating w during model runs is that $\partial \hat{p}_k/\partial t$ is already computed by HYCOM 2.2 in routine **cnuity.f**. It is only necessary to compute the interface slope terms and add them to the interface vertical velocity calculated in **cnuity.f**.

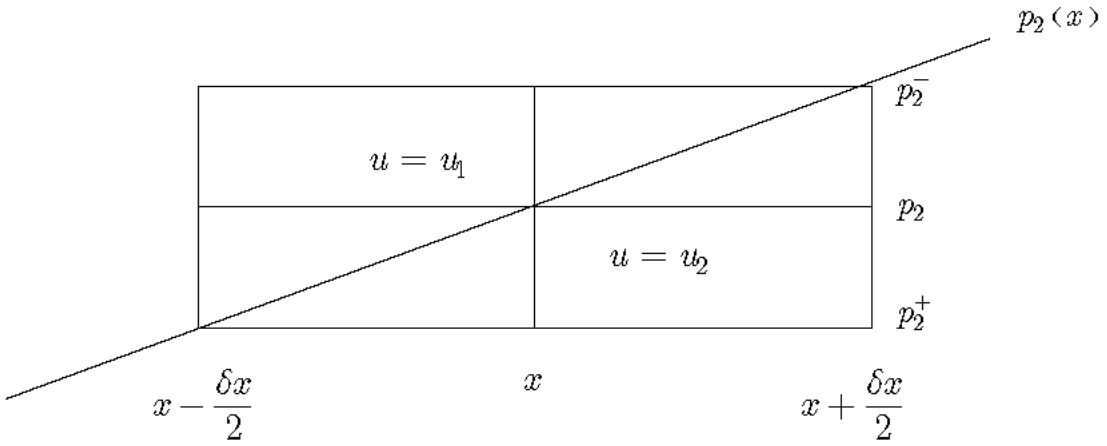


Figure 3: Vertical integration of the continuity equation (134) in a non-Cartesian grid across a model interface that slopes in the x direction.

Validation

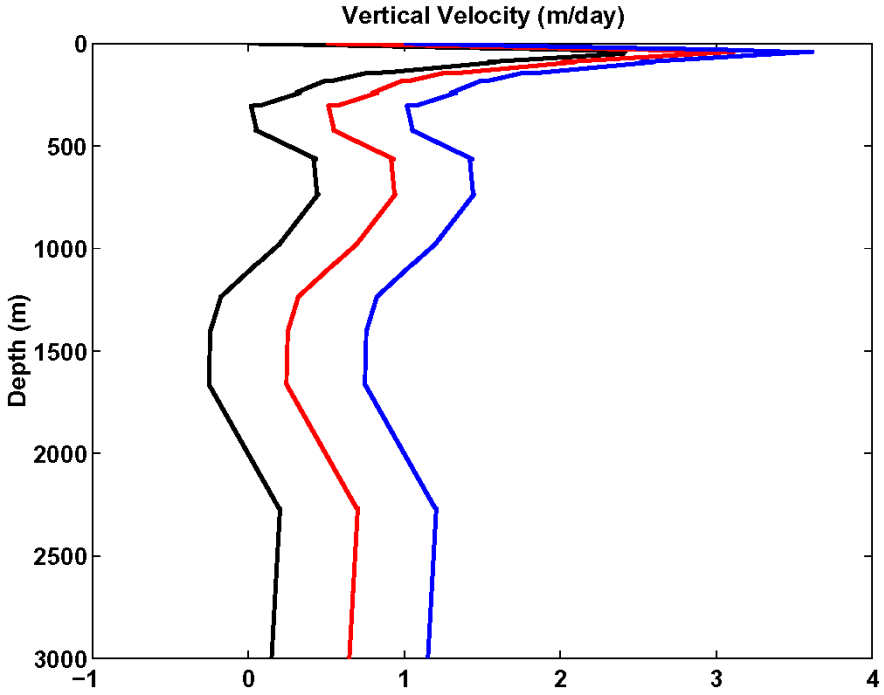
The two equations for estimating w as a function of pressure P [equations (141) and (155)] are now demonstrated to be equal to each other and to w, estimated by first re-gridding

velocity components onto level pressure coordinates and vertically integrating (134) downward from the surface. Computations were made within a low-resolution Atlantic simulation. Before integrating (134), horizontal velocity components are re-gridded using

$$\begin{aligned} \hat{u}(p) &= u(k_l) \\ \hat{v}(p) &= v(k_l) \end{aligned}, \tag{160}$$

where k_l is the layer number within pressure depth p. The re-gridded velocity components $\hat{u},\hat{v}$ are then substituted into (134). The vertical integration is done at high vertical resolution (0.1 m, or 0.001 MPa in pressure units) to minimize truncation errors and resolve the velocity jumps across model interfaces. The velocity components are re-gridded onto pressure depths $p = 0, 0.001, 0.002, \ldots$ MPa and the numerical integration is performed downward from the surface using the trapezoidal rule.

The w profile resulting from the vertical integration of (134) assuming zero vertical velocity at the surface is shown in Figure 4 at the model grid point positioned on the Equator near 28W. The profiles from equations (141) and (155), with varying P in increments of 0.1 m, are also illustrated in Figure 4. The profiles are the same with respect to numerical truncation errors, thus validating the derivation of these two equations. Vertical velocity differs linearly within each layer, and there can be jumps across model interfaces. These jumps are clearest in the upper 300 m. The main difference is that jumps in w occur over a finite depth range in the profiles computed from (134) because of the horizontal grid spacing (Figure 3). This depth range will decrease toward zero as horizontal grid spacing diminishes.



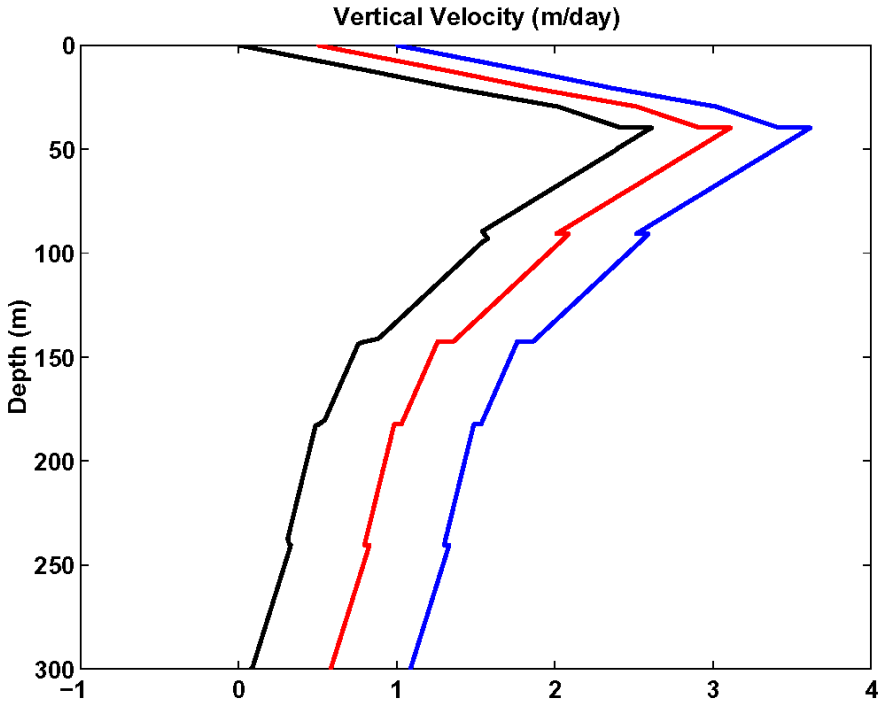


Figure 4. Vertical profile of w in m/day calculated from re-gridding horizontal velocity components onto a Cartesian coordinate system and vertically integrating (134) (black line) for the upper 3000 m (top) and the upper 300 m (bottom). Also shown are profiles calculated from (141) (red line) and from (155) (blue line), each displaced 0.5 m/day to the right.

5.2.14 Continuity Equation

Bleck and Smith (1990) make the assumption that the difference in pressure between the two interfaces of the k^{th} layer has the form

$$\Delta p_k = (1+\eta)\Delta p'_k, \tag{161}$$

where $\Delta p_k = g p_k h_k$ and $\Delta p'_k = g p_k h'_k$. Elsewhere, the decomposition is shown as

$$\mathbf{u}_k = \overline{\mathbf{u}} + \mathbf{u}'_k, \tag{162}$$

with

$$\overline{\mathbf{u}} = \frac{\sum_{k=1}^{N} p_k h_k u_k}{\sum_{k=1}^{N} p_k h_k} \tag{163}$$

and

$$\overline{\mathbf{u}}'_k = 0. \tag{164}$$

The tendency equation for the component $\Delta p'$ coming into the total expression for the change in pressure in layer k (Bleck and Smith, 1990) is written as

$$\frac{\partial}{\partial t}\Delta p'_k + \nabla\cdot(\mathbf{u}\Delta p')_k = \frac{\Delta p'_k}{p'_b}\nabla\cdot(\bar{\mathbf{u}}p'_b). \tag{165}$$

Under the hydrostatic hypothesis with a free surface, the form is written as

$$p'_b = \sum_{k=1}^{N}\Delta p'_k = g\sum_{k=1}^{N}\rho_k h'_k = g\rho_r H, \tag{166}$$

with H representing water depth

$$H(x,y) = \sum_{k=1}^{N}\langle h_k\rangle = \sum_{k=1}^{N} h_k - \xi_1, \tag{167}$$

where $\langle h_k\rangle$ is the initial thickness of the layer k and ξ_1 denotes the free surface (see Figure 5). The variable ρ_r represents column mean density.

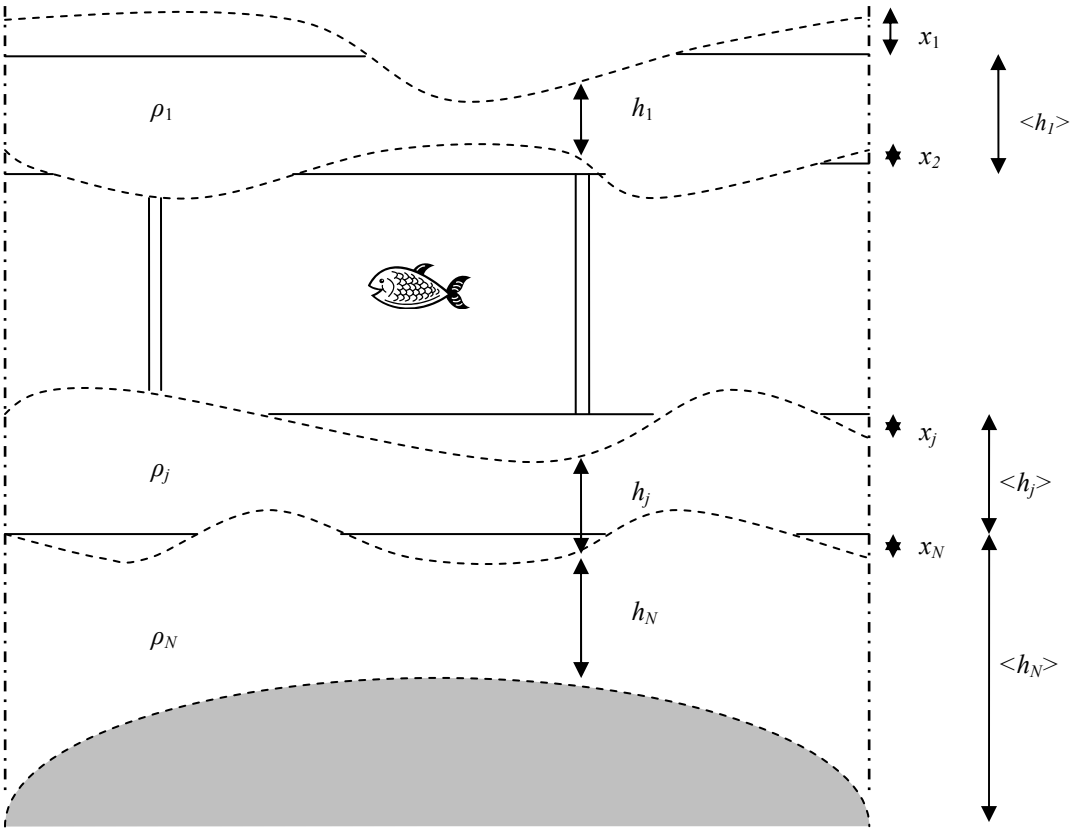


Figure 5: Vertical discretization of the multilayer ocean.

Flux-Corrected Transport Scheme

The basic flux-corrected transport (FCT) scheme is derived from the work of Zalesak (1986) and is summarized in seven steps:

1. The first inference of the variable $\Delta p'_k$ is made by introducing a classic upstream scheme. The variable is represented by $\Delta p'^{up}_{i,j,k}$ in the layer *k*. The diffusive fluxes are calculated from the total velocity field **u.** Equation (165) must be considered without its second term in a 1D form and take $\boldsymbol{P}'_k = (\mathbf{u}\Delta p')_k$. This gives

$$\Delta p'^{up}_{i+1/2,k} = (u\Delta p')^{up}_{i+1/2,k} = \begin{cases} u^{mid}_{i+1/2,k}\Delta p'^{old}_{i,k} & \text{if } u_{i+1/2,k} > 0 \\ u^{mid}_{i+1/2,k}\Delta p'^{old}_{i+1,k} & \text{if } u_{i+1/2,k} < 0 \end{cases}, \qquad (168)$$

   where *mid* and *old* are two successive instants in the leapfrog scheme. With donor cell schemes, the upstream fluxes and the velocity must be defined at the interfaces between cells.

2. When using the total velocity field, the next step is to compute the non-diffusive flux by a second-order scheme in space and centered in time:

$$P^{t^*}_{i+1/2,k} = (u\Delta p')^{*}_{i+1/2,k} = u^{mid}_{i+1/2,k}\,\frac{\Delta p'^{mid}_{i,k} + \Delta p'^{mid}_{i+1,k}}{2}. \qquad (169)$$

3. The flux of anti-diffusion **A** is introduced such that: $A_{i+1/2,k} = P'^{*}_{i+1/2,k} - P'^{up}_{i+1/2,k}$. To assure the stability of the scheme (i.e., to counter the appearance of negative values of $\Delta p'^{new}_{i,k}$), substitute the anti-diffusive flux **A** with the corrected flux $\mathbf{A}^{\boldsymbol{c}}$ such that: $A^{c}_{i+1/2,k} = C_{i+1/2,k}A_{i+1/2,k}$ and with $0 \le C \le 1$. For C = 0, we restore the flux of order 1 and C = 1 gives back the flux of order 2. The final solution is given by a combination of fluxes of orders one and two which moderates the perturbations of stability that appear. For each layer (Baraille and Filatoff, 1995),

$$C_{i+1/2,k} = \begin{cases} \min(R^{+}_{i+1,k}; R^{-}_{i,k}) & \text{if} \quad A_{i+1/2,k} \ge 0 \\ \min(R^{+}_{i,k}; R^{-}_{i+1,k}) & \text{if} \quad A_{i+1/2,k} < 0 \end{cases}, \qquad (170)$$

   where the following factors are successively introduced:

$$\mathrm{I}\begin{cases} P^{+}_{i,k} = \max(0, A_{i-1/2,k}) - \min(0, A_{i+1/2,k}) \\ P^{-}_{i,k} = \max(0, A_{i+1/2,k}) - \min(0, A_{i-1/2,k}) \end{cases} \qquad (171)$$

   then

$$\text{II}\begin{cases} Q^{+}_{i,k} = \Delta p'^{max}_{i,k} - \Delta p'_{i,k} \\ Q^{-}_{i,k} = \Delta p'_{i,k} - \Delta p'^{min}_{i,k} \end{cases} \tag{172}$$

and

$$\text{III}\begin{cases} R^{+}_{i,k} = \begin{cases} \min\left(1, \frac{\Delta x}{\Delta t}\frac{Q^{+}_{i,k}}{P^{+}_{i,k}}\right) \text{ if } P^{+}_{i,k} > 0 \\ 0 \text{ if } P^{+}_{i,k} = 0 \end{cases} \\ R^{-}_{i,k} = \begin{cases} \min\left(1, \frac{\Delta x}{\Delta t}\frac{Q^{-}_{i,k}}{P^{-}_{i,k}}\right) \text{ if } P^{-}_{i,k} > 0 \\ 0 \text{ if } P^{-}_{i,k} = 0 \end{cases} \end{cases}. \tag{173}$$

$R^{+}_{i,k}$ and $R^{-}_{i,k}$ are the largest multiplicative factors of anti-diffusive flux which assure respectively that $\Delta p'^{n+1}_{i,k} \le \Delta p'^{max}_{i,k}$ and $\Delta p'^{n+1}_{i,k} \ge \Delta p'^{min}_{i,k}$ with

$$\begin{cases} \Delta p'^{max}_{i,k} = \max(\Delta p'^{n}_{i-1,k}, \Delta p'^{n}_{i,k}, \Delta p'^{n}_{i+1,k}) \\ \Delta p'^{min}_{i,k} = \min(\Delta p'^{n}_{i-1,k}, \Delta p'^{n}_{i,k}, \Delta p'^{n}_{i+1,k}) \end{cases}. \tag{174}$$

The formulas corresponding to the bidimensional case are illustrated in Baraille and Filatoff (1995).

4. In summing equation (165) without the second term over the N layers of the vertical discretization, when $\partial p'_b / \partial t = 0$:

$$\mathbf{P}' = \sum_{k=1}^{N} \nabla \cdot \mathbf{P}'_k = 0. \tag{175}$$

Clearly, satisfying this condition requires a second order approximation of the flux $\mathbf{P}'$, and is preferable to any lower order approximation. The following form is then written as

$$\sum_{k=1}^{N} \left(P'^{*}_{i+1/2,k} - P'^{*}_{i-1/2,k} \right) = 0. \tag{176}$$

The sum of anti-diffusive flux corrections over the vertical introduces a bias in the conservation of p'_b which must be compensated. To accomplish this, compute the vertical sum of flux corrections (through a second order approximation):

$$A_{i+1/2} = \sum_{k=1}^{N} \left(1 - C_{i+1/2,k}\right) \mathbf{A}_{i+1/2,k}. \tag{177}$$

5. Evaluating the effect of the anti-diffusive flux "integral" by calculating by layer the thickness, we have:

$$\Delta p'^{\dagger}_{i,k} = \Delta p'^{up}_{i,k} - \Delta t \left(\nabla \cdot \mathbf{A}^{c} \right)_{i,k}. \tag{178}$$

From the new thickness, the bottom pressure comes from

$$\Delta p'^{\dagger}_{bi} = \sum_{k=1}^{N} \Delta p'^{\dagger}_{k}. \tag{179}$$

6. Unless $C_{i+1/2,k} = 1$ for each layer, $p'^{\dagger}_{bi} \neq p'_{bi}$. To fix this a second correction B is made to the upstream flux such that

$$B_{i+1/2,k} = \frac{\Delta p'^{\dagger}_{i,k}}{p'^{\dagger}_{bi}} A_{i+1/2}. \tag{180}$$

The final flux becomes

$$P'^{fin}_{i+1/2,k} = P'^{up}_{i+1/2,k} + A^{c}_{i+1/2,k} + B_{i+1/2,k}, \tag{181}$$

and the preceding equation $\Delta p'^{\dagger}_{i,k}$ is rectified by the form

$$\Delta p'^{\ddagger}_{i,k} = \Delta p'^{\dagger,k}_{i,k} - \Delta t \left(\nabla \cdot \mathbf{B} \right)_{i,k}, \tag{182}$$

where the bottom pressure is set at

$$\Delta p'^{\ddagger}_{bi} = \sum_{k=1}^{N} \Delta p'^{\ddagger}_{i,k}. \tag{183}$$

7. The final step is accounting for the right hand side of the original equation (165). At this point the claim is made that the flux was adjusted to best fit $\partial p'_b / \partial t = 0 \ \forall i$. From this statement the following equation can be written for each layer k

$$\frac{\partial}{\partial t} \Delta p'^{fin}_{k} + \nabla \cdot \mathbf{P}'^{fin}_{k} = \frac{\Delta p'^{fin}_{k}}{p'_b} \nabla \cdot \left(\overline{\mathbf{u}} p'_b \right) \tag{184}$$

in order to satisfy

$$\sum_{k=1}^{N} \Delta p'^{fin}_{k} = p'_b. \tag{185}$$

For all points, equation (184) is summed over the vertical providing that

$$\nabla \cdot \left(\sum_{k=1}^{N} \mathbf{P}'^{fin}_{k} \right) = \nabla \cdot \left(\overline{\mathbf{u}} p'_b \right). \tag{186}$$

In the divergence term, the vertical flux average and the flux sum are balanced. Conversely, the flux and layer thickness may need tuning. Accounting for the second term of equation (165) in the computation of the final thickness $\Delta p'^{fin}_{k}$ provides the simple equation

$$\Delta p'^{fin}_k \frac{\Delta p'^{\ddagger}_k}{p'^{\ddagger}_b} p'_b. \tag{187}$$

From this we have

$$p'^{fin}_{k+1} = \sum_{l=1}^{k} \Delta p'^{fin}_l \text{ for } k = 1,...., N. \tag{188}$$

with $p'^{fin}_{N+1} = p'^{fin}_b \equiv p'_b$. Then in equations (166), (179), (183) and (188), the surface pressure is assumed to be zero ($p' = 0$).

Interface Diffusion

In a shallow-water multilayer model, the conservation of mass is set in the equation (Baraille and Filatoff, 1995):

$$\frac{\partial}{\partial t}\Delta p_k + \nabla \cdot \left(\mathbf{u}\Delta p\right)_k = 0. \tag{189}$$

Once the decomposition (161) is brought into the formula, it is apparent that formally equation (165) is derived from the approximation $\left(1+\eta\right) \approx 1$. Note that the use of this approximation does not in any way disturb the property $\partial p'_b / \partial t = 0$ (Baraille and Filatoff, 1995). Therefore, the sum of the changes in $\Delta p'$ over the vertical in this estimation is such that at every point, p'_b remains constant. In practice, the accounting of η is such that

$$1 + \eta = \frac{1}{\rho_r H} \sum_{k=1}^{N} \rho_k h_k \tag{190}$$

in order to restore the initiation of a diffusion term $\nabla \cdot \left(\nu \nabla \Delta p'\right)$ in the conservation equation (165). As in finite differences, the product $\nu \partial \left(\Delta p\right) / \partial x$ is numerically equal to $u_d \delta p$, where δp is the growth of the thickness of a layer between two adjacent meshes. Bleck *et al*. (1992) present a "diffusion velocity" $u_d \equiv \nu / \Delta x$, where Δx is the size of the mesh, to reproduce the isopycnic diffusion. Generally $u_d = 0.5\ cm/s$ for the variable Δp.

From the preceding FCT method, $p'^{fin}_{i,k}$ and $p'^{fin}_{i-1,k}$, the pressures at the k^{th} density cross at two adjacent points with coordinates x_i and x_{i-1}, are obtained. At these two points the bottom pressures are p'_{bi} and p'_{bi-1}, respectively. This statement is made concrete by the equation

$$D_{i-1/2,k} = \min\left\{ p'_{bi} - p'_{i,k} \max\left[p'_{i-1,k} - p'_{bi-1}, \frac{u_d \Delta t}{\Delta x}\left(p'_{i-1,k} - p'_{i,k} \right) \right] \right\}. \tag{191}$$

During a time interval Δt, the variation in pressure at the interface k comes from the expression

$$\frac{\partial p'_{i,k}}{\partial t} + \frac{\Delta x}{\Delta t}\left(\nabla \cdot \mathbf{D}\right)_{i,k} = 0. \tag{192}$$

The last form is then written for the interfaces k=2,...,N:

$$p'^{n+1}_{i,k} = p'^{fin}_{i,k} - \left(D_{i+1/2,k} - D_{i-1/2,k} \right), \tag{193}$$

where

$$\Delta p'^{n+1}_{i,k} = p'^{n+1}_{i,k+1} - p'^{n+1}_{i,k}. \tag{194}$$

Remaining coherent with the previous step requires that the flux retained at the two interfaces of the layer are given as

$$\begin{cases} \mathbf{F}^{fin}_{i+1/2,k-1} = \mathbf{P}'^{fin}_{i+1/2,k-1} + \dfrac{1}{\Delta t}\mathbf{D}_{i+1/2,k} \\ \mathbf{F}^{fin}_{i+1/2,k} = \mathbf{P}'^{fin}_{i+1/2,k} - \dfrac{1}{\Delta t}\mathbf{D}_{i+1/2,k} \end{cases}.$$

5.2.15 HYCOM 2.2 Horizontal Advection and Diffusion

When HYCOM 2.2 is run with isopycnic vertical coordinates (in isopycnic coordinate mode), temperature and salinity (T and S) are advected and diffused in layer 1. Only salinity is advected and diffused in deeper layers, with temperature diagnosed from the equation of state to uphold constant density in these layers. When HYCOM 2.2 is run with hybrid vertical coordinates, the user may select whether T and S, or just S, are advected and diffused within the upper n_{hyb} layers that the user defines as hybrid layers. This option exists because the effects of cabbeling when both T and S are advected and diffused can lead to issues with adjustment of vertical coordinates by the hybrid coordinate algorithm. This is particularly true if the user selects to flux both T and S across the moving vertical coordinates. When only salinity is advected/diffused, these issues do not arise, but then temperature is no longer conserved. However, in low-resolution simulations of Atlantic Ocean climate, the non-conservation of temperature did not have a large influence on simulated fields.

In isopycnic coordinates, the thermal evolution equation is written as

$$\frac{\partial}{\partial t} T\Delta p + \underbrace{\nabla \cdot \left(\mathbf{u} T \Delta p \right)}_{advec} + \underbrace{\left(\dot{s} \frac{\partial p}{\partial s} T \right)_{bot} - \left(\dot{s} \frac{\partial p}{\partial s} T \right)_{top}}_{dia-diff} = \underbrace{\nabla \cdot \left(\nu \Delta p \nabla T \right)}_{iso-diff} + \mathcal{H}_T. \tag{195}$$

Δp represents the thickness of layer k of temperature T. The radiative exchanges are represented by the term $\mathcal{H}_T$. The expression $\left(\dot{s} \partial p / \partial s \right)$ denotes a vertical mass flux.

The advection of heat and salt in HYCOM 2.2 is treated by MPDATA through the work of Smolarkiewicz and Clark (1986) and Smolarkiewicz and Grabowski (1990). The diapycnic diffusion is accounted for in the diapycnic mixing algorithm discussed in Section 5.2.2. With regard to isopycnic diffusion, the numerical method used is based on a different set of procedures described below in Section 5.2.15.

Maintaining the positivity of thickness

The issue of advection of an isopycnic layer of thickness Δp, of temperature T and driven by a horizontal velocity $\mathbf{u}$ can be addressed through the form

$$\frac{\partial}{\partial t} T\Delta p + \nabla \cdot \left(\mathbf{u} T \Delta p \right) = 0. \tag{196}$$

From the solution of the continuity equation, for each layer, the mass flows $\left(u\Delta p\right)_{i,j}^{n+1}$ and $\left(v\Delta p\right)_{i,j}^{n+1}$ as well as a diagnostic value for $\Delta p_{i,j}^{n+1}$ are calculated. Let $\mathbf{P} = \mathbf{u}\Delta p$ and let h be the thickness of the layer of interest. Bearing in mind the conservation equation

$$\frac{\partial h}{\partial t} + \nabla \cdot \mathbf{P} = 0 \tag{197}$$

and determining the variation δh follows from this equation we get

$$\delta h\left(i,j\right) = -\frac{\Delta t}{\Delta x}\left[Px_{i+1,j}^{n+1} - Px_{i,j}^{n+1} + Py_{i,j+1}^{n+1} - Py_{i,j}^{n+1} \right]. \tag{198}$$

Suppose the layer shrinks over time, i.e., $\delta h < 0$. In this case, $\left|\delta h\right| > \Delta p^{n-1}$ sets forth the unsatisfactory condition $\Delta p'^{n+1} < 0$. The fluxes following from the numerical treatment of the continuity equation are then inconsistent with the variations of thickness computed in this exact step. The mass flux, in addition to the layer thickness, are necessary in the advection computation. The mass fluxes are created to be consistent with the variation of layer thickness so that when $\delta h < 0$, then $\left|\delta h\right| \le \Delta p^{n-1}$. Now we introduce the two utility thicknesses h_1 and h_2 by

$$h_1 = 1/2\left(\Delta p^{n+1} + \Delta p^{n-1} + \delta h\right) \; ; \; h_2 = 1/2\left(\Delta p^{n+1} + \Delta p^{n-1} - \delta h\right). \tag{199}$$

In situations where the fluxes are strictly consistent (i.e., $\delta h = \Delta p^{n+1} - \Delta p^{n+1}$),

$$h_1 = \Delta p^{n-1} \text{ and } h_2 = \Delta p^{n+1}.$$

When the flux is consistent, the utility thicknesses seen in equation (199) are introduced. Otherwise, when fluxes are inconsistent, the positivity of the layer thicknesses is kept by allowing

$$h_1 = -\delta h \text{ and } h_2 = 0.$$

For $\delta h > 0$, the respective values of h1 and h2 are found with identical reasoning.

Treatment of the tendency term

Assuring a gradual transition of Δp towards a null value requires that the finite difference portion of the tendency term in equation (195) that is usually written as

$$\frac{T^{new}\Delta p^{new} - T^{old}\Delta p^{old}}{\Delta t}$$

be written now as

$$\frac{T^{new}\left(\Delta p^{new} + \epsilon\right) - T^{old}\left(\Delta p^{old} + \epsilon\right)}{\Delta t}. \tag{200}$$

The small parameter ϵ will be non-zero when the loss of mass during a time step averages at least 90% of the previous value. To be more exact:

$$\epsilon = A + \left(\epsilon_1^2 + A^2\right)^{1/2}, \tag{201}$$

where $A = \left(0.1\Delta p^{old} - \Delta p^{new}\right)/2$. The intermediate parameter ϵ_1 is therefore set to the numerical value 10 cm, so as to assure the validity of (200) when Δp^{old} and Δp^{new} are tending toward zero.

Treatment of the diffusion term

As with finite differences, the product $v\partial T/\partial x$ is numerically equal to $u_d \delta T$, where δT denotes the temperature difference among the two adjacent mesh points bracketing u_d. A diffusion velocity $u_d \equiv v/\Delta x$, where Δx is the size of the mesh, has been introduced to simulate isopycnic mixing (Bleck *et al*., 1992). Generally, u_d = 1 cm/s for the variables *T* and *S*. Conversely, in the flux expression appearing in the diffusion term, the variation Δp is replaced with the harmonic average

$$\widetilde{\Delta p} = \frac{2}{\Delta p_{i-1}^{-1} + \Delta p_i^{-1}}. \tag{202}$$

The reasoning behind this choice is not clear but can be found in Bleck *et al*. (1992). It can be explained this way: Consider two neighboring mesh points which are related to the two values Δp_{i-1} and Δp_i and the two temperatures T_{i-1} and T_i. In order to negate the effect of introducing factor Δp in the flow expression, a substitution of a neutral value $\widetilde{\Delta p}$ characterizing a neutral state is necessary. In this neutral context, two neighboring mesh points have an equal amount of heat *Q* and an average temperature $\widetilde{T}$. If the turbulence and the turbulence diffusion are interpreted as a mixing process, then $\widetilde{T} = \left(T_{i-1} + T_i\right)/2$. Naturally the neutral state does not have the same thermal content *Q*. So then

$$\widetilde{T} = \frac{Q}{\widetilde{\Delta p}} = \frac{1}{2}\left(\frac{Q}{\Delta p_{i-1}} + \frac{Q}{\Delta p_i}\right), \tag{203}$$

an expression that led to the form (202). In addition, as the inference of the new value $T_{i,j}^{n}$ requires a division by Δp, treating the situations $\Delta p \to 0,$ requires introducing a residual thickness given the numerical value of 1 mm.

Filtering

In the classical method for compensating for dispersion problems caused by leapfrog schemes, Asselin filtering is used:

$$\hat{T}^{n}\,\widehat{\Delta p}^{\,n} = \left[T^{n}\left(1-2\gamma\right)\Delta p^{n} + \gamma\left(\hat{T}^{n-1}\,\widehat{\Delta p}^{\,n-1} + T^{n+1}\Delta p^{n+1}\right)\right]. \tag{204}$$

To make this form remain valid when $\Delta p \to 0,$ a residual thickness ϵ is introduced so that

$$\hat{T}^{n} = \left(\widehat{\Delta p}^{\,n} + \epsilon\right)^{-1}\left[T^{n}\left\{\left(1-2\gamma\right)\Delta p^{n} + \epsilon\right\} + \gamma\left(\hat{T}^{n-1}\,\widehat{\Delta p}^{\,n-1} + T^{n+1}\Delta p^{n+1}\right)\right]. \tag{205}$$

In practice, ϵ takes a numerical value equal to 10^{-3} m. With respect to thermodynamic variables, the value γ =0.015625 is used. Additionally, the filtering is carried out in two steps in the code. Collectively we have

$$\left[\widehat{T\delta p}\right]^{n} = T^{n}\left\{\left(1-2\gamma\right)\Delta p^{n} + \epsilon\right\} + \gamma\hat{T}^{n-1}\,\widehat{\Delta p}^{\,n-1}. \tag{206}$$

Then, after having computed T^{n+1} and filtering the layer thickness Δp by the equation

$$\widehat{\Delta p}^{\,n} = \left(1-2\gamma\right)\Delta p^{n} + \gamma\left(\widehat{\Delta p}^{\,n-1} + \Delta p^{n+1}\right), \tag{207}$$

the second step is given as

$$\hat{T}^{n} = \left(\widehat{\Delta p}^{\,n} + \epsilon\right)^{-1} + \left\{\left[\widehat{T\delta p}\right]^{n} + \gamma T^{n+1}\Delta p^{n+1}\right\}. \tag{208}$$

5.2.16 Barotropic Mode

Beginning with the decomposition (161) and summing over all layers of the general continuity equation, the following equation is given as

$$\frac{\partial \eta p_{b}'}{\partial t} + \nabla\cdot\left[\left(1+\eta\right)\overline{\mathbf{u}}p_{b}'\right] = 0. \tag{209}$$

Establishing the corresponding equations of motion requires consideration of the average of the general momentum equation

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla\frac{\mathbf{u}^{2}}{2} + \left(\varsigma + f\right)\mathbf{k}\times\mathbf{u} = -\nabla M - g\frac{\partial \tau}{\partial p}, \tag{210}$$

with ς representing relative vorticity, **k** denoting vertical unit vector, M as Montgomery potential, τ as Reynolds stress and *f* as Coriolis, in the sense of equation (163). Then, the decomposition (162) is included, followed by the equations (164) and (161). Finally the system is coalesced by

$$\frac{\partial}{\partial t}\left[\overline{u}\left(1+\eta\right)\right]-\left(1+\eta\right)f\overline{v}+\frac{1}{\rho_r}\frac{\partial}{\partial x}\left[\eta p_b'\left(1+\eta\right)\right]+\frac{R+Q\left(\eta\right)}{p_b'}=0$$

$$\frac{\partial}{\partial t}\left[\overline{v}\left(1+\eta\right)\right]+\left(1+\eta\right)f\overline{u}+\frac{1}{\rho_r}\frac{\partial}{\partial y}\left[\eta p_b'\left(1+\eta\right)\right]+\frac{R'+Q'\left(\eta\right)}{p_b'}=0, \tag{211}$$

which is more simply written as

$$\frac{\partial\overline{u}}{\partial t}-f\overline{v}+\frac{1}{\rho_r}\frac{\partial}{\partial x}\left(\eta p_b'\right)=\frac{\partial\overline{u}^*}{\partial t}$$

$$\frac{\partial\overline{v}}{\partial t}+f\overline{u}+\frac{1}{\rho_r}\frac{\partial}{\partial y}\eta p_b'=\frac{\partial\overline{v}^*}{\partial t}. \tag{212}$$

$Q\left(\eta\right)$ and $Q'\left(\eta\right)$ are expressions grouping the nonlinear terms. R and R' are the components of pressure gradient produced by the stratification. The pseudo-velocity $\overline{\mathbf{u}}^*\left(\overline{u}^*,\overline{v}^*\right)$ guarantees the property (164).

The time step of the external mode, Δt_B (barotropic), and the time step of the internal mode, Δt_b (baroclinic), are given as

$$\Delta t_b = N\Delta t_B. \tag{213}$$

The barotropic expressions are solved N times between two solutions of the baroclinic expressions.

Rescaling of variables

In the last step of the continuity equation, a tendency computation that utilizes layer thicknesses from height points by the application of formula (193) requires a rescaling of the variables, as they are distributed on a C grid. Furthermore, during initialization (through subroutine *inicon*), the depths at velocity points have been introduced by the equations

$${}_up'_{bi,j}=\min\left(p'_{b_{i-1,j}},p'_{b_{i,j}}\right) \text{ and } {}_vp'_{bi,j}=\min\left(p'_{b_{i-1,j}},p'_{b_{i,j}}\right). \tag{214}$$

Retaining consistency with the initial definitions, the average thickness is introduced at the point of the u component

$$\begin{aligned}{}_u\Delta p'_{i,j,k}=\max\{&0.,\\ &\min\left[{}_up_{bi,j},1/2\left({}_up'_{i,j,k+1}+{}_up'_{i-1,j,k+1}\right)\right]\\ -\ &\min\left[{}_up'_{b_{i,j}},1/2\left({}_up'_{i,j,k}+{}_up'_{i-1,j,k}\right)\right]\}\end{aligned} \tag{215}$$

The equivalent formula is used at the point of the v component.

Rearrangement of the velocity profile

When the thickness of a layer is so thin as to be considered numerically zero, it may still contain momentum. If layer k "disappears", in terms of momentum it still exists and acquires the momentum of the layer above it. Translating this mechanism, the variable q is introduced

$$q = \delta - \min\left({}_u\Delta p'_k, \delta\right) = \begin{cases} 0 & \text{if } {}_u\Delta p'_k \geq \delta \\ \delta - {}_u\Delta p'_k & \text{if } {}_u\Delta p'_k < \delta, \end{cases} \tag{216}$$

which allows the weighted value to be defined as (the same rule applies to the v component):

$$\underline{u}_k = \frac{1}{\delta}\left[u'_k\left({}_u\Delta p'_k\right) + u'_{k-1}\left(\delta - {}_u\Delta p'_k\right)\right]. \tag{217}$$

In HYCOM 2.2, $\delta = 10^{-1}$ m.

Filtering

As in the advection step, Asselin filtering for velocity is used to solve the problems of dispersion caused by the leapfrog scheme. The component u' of the baroclinic velocity for the layer k and the node (i, j) are written

$$\widehat{u'}^n \widehat{\Delta p'}^n = \left[u'^n(1-2\gamma)\left(\Delta p'^n + \epsilon\right) + \gamma\left(\widehat{u'}^{n-1}\widehat{\Delta p'}^{n-1} + u'^{n+1}\widehat{\Delta p'}^{n+1}\right)\right]. \tag{218}$$

A residual thickness ϵ is brought in for which this equation remains valid when ${}_u\Delta p'_{i,j,k} \to 0$. HYCOM 2.2 sets ϵ to a numerical value of 10^{-3} m and γ =0.25. The thickness of the layer ${}_u\Delta p'_{i,j,k}$ is also filtered by the formula:

$$\widehat{\Delta p'}^n = (1-2\gamma)\left(\Delta p'^n + \epsilon\right) + \gamma\left(\widehat{\Delta p'}^{n-1} + \Delta p^{n+1}\right). \tag{219}$$

Continuity equation

The continuity equation (210) is treated with the simplification $(1+\eta) \approx 1$. Using this approximation does not disturb the property $\partial p'_b / \partial t$ =0 (Baraille and Filatoff, 1995). The treated variable is then $p'' = \eta p'_b$. Combining forward time stepping we get

$$P''^{m+1}_b = P''^m_b - \Delta t_B \nabla \cdot \left(\overline{\mathbf{u}} p'_b\right)^m \tag{220}$$

and with Asselin time filtering becomes

$$P''^{m+1}_b = (1-\mathrm{w})\, P''^m_b + \mathrm{w}\, P''^{m-1}_b - \Delta t_B (1+\mathrm{w}) \nabla \cdot \left(\overline{\mathbf{u}} p'_b\right)^m, \tag{221}$$

setting w = 0.125.

Equations of motion

The equations of motion given by the form (212) call the reference density ρ_r introduced to represent the ocean of reference depth *H*. In HYCOM 2.2, the identification $\rho_r \equiv \rho_0$ is made.

The vector $\partial\overline{\mathbf{u}}^*/\partial t$ found in the right hand side of the equations of (212) can then be viewed as a forcing term in the generation of the linear barotropic mode. The solution of this system requires the extraction of the component $\overline{\mathbf{u}}^*$. In the preceding step, the baroclinic velocity profile expressed by the variable $\mathbf{u}''_k = \mathbf{u}'_k + \overline{\mathbf{u}}^*$ is computed. In carrying out the sums

$$S_u = \sum_{k=1}^{N} u''_k \Delta p'_k \text{ and } S_v = \sum_{k=1}^{N} v''_k \Delta p'_k \tag{222}$$

and in accounting for the property (167)

$$\overline{u}^* = \frac{S_u}{p'_b} \text{ and } \overline{v}^* = \frac{S_v}{p'_b}, \tag{223}$$

note that the pseudo-vector $\overline{\mathbf{u}}^*$ is not a variable of state in the system where evolution is sought. In the preceding step, the transition is efficiently inferred by $\mathbf{u}'^n_{i,j,k} \rightarrow \left(\mathbf{u}'_k + \overline{\mathbf{u}}^*\right)^{n+1}_{i,j}$. In addition, the weighting w is introduced so that

$$\overline{\mathbf{u}}^{m+1}_{i,j} = \left(1-\mathrm{w}\right)\overline{\mathbf{u}}^{m}_{i,j} + \mathrm{w}\,\overline{\mathbf{u}}^{m-1}_{i,j} - \Delta t_B \left(1+\mathrm{w}\right)\left[-\alpha_0 \left(\frac{\partial p''_b}{\partial x}\right)^{m+1}_{i,j} + \overline{f\overline{v}}^{m}_{i,j} + \overline{u}^{*,n+1}_{i,j}/\Delta t_b\right]. \tag{224}$$

The Coriolis term is described by the centered equation

$$\overline{f\overline{v}}_{i,j} = 1/8\left(f'_{i,j} + f'_{i,j+1}\right)\left[\left(\overline{v}_v p'_b\right)_{i,j} + \left(\overline{v}_v p'_b\right)_{i-1,j} + \left(\overline{v}_v p'_b\right)_{i,j+1} + \left(\overline{v}_v p'_b\right)_{i-1,j+1}\right], \tag{225}$$

with the barotropic potential vorticity f' described as

$$f'_{i,j} = \frac{f}{p'_{b_{i,j}}}. \tag{226}$$

The continuity equation is solved first. The pressure gradient of the equation of motion (224) employs the value of the state of perturbation η from this computation. The combination of the forward time-stepping in the continuity equation and backward time stepping in the momentum equations is called the forward-backward scheme.

6.0 PRIMARY HYCOM 2.2 FORTRAN ROUTINES

Routine	Description
archiv	Writes an archive file. **Calling Sequence:** subroutine archiv(n, kkout, iyear,iday,ihour, intvl) **Data Declaration:** integer n, kkout, iyear,iday,ihour real sssc, sstc character intvl **I/O:** stdout, write flnmarc, nop, cformat open unit nop **Uses:** mod_xc, mod_za
barotp	Advances barotropic equations from baroclinic time level -*m*- to level -*n*-. There is explicit time integration of barotropic flow (forward-backward scheme). In order to combine the forward-backward scheme with a leapfrog treatment of the Coriolis term, *v-eqn* must be solved before *u-eqn* every other time step. **Calling Sequence:** subroutine barotp(m,n) **Data Declaration:** integer m, n **I/O:** stdout **Uses:** mod_xc, mod_pipe, mod_tides
cnuity	Continuity equation (flux-corrected transport version). **Calling Sequence:** subroutine cnuity(m,n) **Data Declaration:** integer m,n **I/O:** stdout, write text, textu, textv **Uses:** mod_xc, mod_pipe, mod_floats
convch	Convective adjustment subroutine of **convec.f**. Performs convection of u, v, thermodynamical variables and tracers. **Calling Sequence:** subroutine convch(m,n) **Data Declaration:** integer m,n I/O: stdout **Uses:** mod_xc
convcm	Convective adjustment subroutine of **convec.f**. It entrains water lighter than mixed-layer water into the mixed layer. **Calling Sequence:** subroutine convcm(m,n) **Data Declaration:** integer m,n I/O: stdout **Uses:** mod_xc
diapf1	KPP-style implicit interior diapycnal mixing routine for shear instability, double diffusion, and background internal waves. **Diapfl.f** is basically the KPP mixing model (**mxkpp.f**) with surface boundary layer processes

Routine	Description
	removed. It uses the same tri-diagonal matrix solution of vertical diffusion equation as **mxkpp.f** . Related subroutines are found in Section 6.6. **Calling Sequence:** subroutine diapf1(m,n) **Data Declaration:** integer m,n **I/O:** None **Uses:** mod_xc
diapf2	Isopycnic coordinate explicit interior diapycnal mixing for hybrid coordinates. Related subroutines are found in Section 6.6. **Calling Sequence:** subroutine diapf2(m,n) **Data Declaration:** integer m,n **I/O:** stdout **Uses:** mod_xc
diapf3	Isopycnic coordinate explicit interior diapycnal mixing subroutine for isopycnal coordinates. Related subroutines are found in Section 6.6. **Calling Sequence:** subroutine diapf3(m,n) **Data Declaration:** integer m,n **I/O:** stdout **Uses:** mod_xc
hybgen	This is the hybrid grid generator. See Section 6.7 for associated subroutines. **Calling Sequence:** subroutine hybgen(m,n) **Data Declaration:** integer m,n **I/O:** stdout, write text **Uses:** mod_xc, mod_pipe
hycom	This is an ESMF driver for the stand-alone HYCOM 2.2 ocean model. It contains no arguments or common blocks. **I/O:** None **Uses:** ESMF_Mod, mod_hycom
hycom_cice	This is an ESMF driver for HYCOM 2.2 ocean model and CICE sea-ice model. As of this writing, this subroutine is not used in HYCOM 2.2. It contains no arguments or common blocks. **I/O:** stdout, write unit 6 **Uses:** ESMF_Mod, mod_hycom, ice_kinds_mod, CICE_ComponentMod, CICE_InitMod, CICE_RunMod, mod_OICPL
icloan	This is the “energy loan” ice model. It has no advection, no dynamics, and the ice amount represents energy “loaned” to the water column to prevent wintertime cooling below freezing level. The loan is paid back in summertime. This is a modified version for ice-ocean "coupling", with freeze/melt energy from relaxation to the freezing temperature. The atmosphere/ice surface exchange is applied to the ocean, which was previously done in **thermf.f**.

Routine	Description
	Calling Sequence: subroutine icloan(m,n) **Data Declaration:** integer m,n **I/O:** stdout **Uses:** mod_xc
momtum	Momentum equation. **Calling Sequence:** subroutine momtum (m,n) **Data Declaration:** integer m,n **Common Blocks:** common/momtumr4/ **I/O:** stdout, write text **Uses:** mod_xc, mod_pipe
momtum_hs	Hydrostatic equation with surface stress. Performs initial sea surface height calculation. **Calling Sequence:** subroutine momtum_hs(m,n) **Data Declaration:** integer m,n **Common Blocks:** common/momtumr4/ **I/O:** None **Uses:** mod_xc, mod_pipe, mod_tides
mxkprf	K-profile vertical mixing models. a) Large, McWilliams, Doney KPP vertical diffusion. b) Mellor-Yamada 2.5 vertical diffusion. c) GISS vertical diffusion. See Section 6.10 for vertical mixing subroutines. **Calling Sequence:** subroutine mxkprf(m,n) **Data Declaration:** integer m,n **I/O:** stdout, write text **Uses:** mod_xc, mod_pipe
mxkrta	HYCOM original slab mixed layer. See Section 6.10 for vertical mixing subroutines. **Calling Sequence:** subroutine mxkrta(m,n) **Data Declaration:** integer m,n **I/O:** stdout **Uses:** mod_xc
mxkrtb	Alternative slab mixed layer model. See Section 6.10 for vertical mixing subroutines. **Calling Sequence:** subroutine mxkrtbaj(m,n, j) **Data Declaration:** integer m,n,j **I/O:** None **Uses:** mod_xc
mxkrtm	Mixed layer entrainment/detrainment.

Routine	Description
	Calling Sequence: subroutine mxkrtm(m,n) **Data Declaration:** integer m,n **I/O:** stdout, stdout **Uses:** mod_xc
mxpwp	Price-Weller-Pinkel (PWP) dynamical instability vertical mixing model. **Calling Sequence:** subroutine mxpwp(m,n) **Data Declaration:** integer m,n **I/O:** None **Uses:** mod_xc
profile_lat	This routine returns either pressure as a function of density and latitude or density as a function of pressure and latitude. To return pressure, set press < 0.0. **Calling Sequence:** subroutine profile_lat(theta,press,xlat) **Data Declaration:** real theta,press,xlat **Common Blocks:** common/linepr/ **I/O:** stdout **Uses:** None
tsadvc	Thermodynamic variables for advection and diffusion. See Section 6.1 for corresponding subroutines. **Calling Sequence:** subroutine tsadvc(m,n) **Data Declaration:** integer m, n **I/O:** stdout, text, textu, textv **Uses:** mod_xc, mod_pipe, mod_advem
wtime	This set of real functions return wall time. Real function *wtime_dummy* uses a C routine called **machi_c.c**. Real function *wtime* uses the MPI function mpi_wtime or the Fortran 90 intrinsic *system_clock*. The function will fail if the count is ever negative, but if a clock exists the standard says it is always non-negative. No arguments are called. **I/O:** None **Uses:** None

6.1 Advection and Diffusion Subroutines

Subroutine	Description
advem	Wrapper for advection schemes. **Calling Sequence:** subroutine advem(advtyp,fld,fldc,u,v,fco,fcn,posdef, scal,scali,dt2) **Data Declaration:** integer integer advtyp real posdef, dt2, fld, fldc, u, v, fco, fcn, scal, scali

Subroutine	Description
	I/O: stdout **Uses:** None
advem_fct2	Subroutine for leapfrog 2nd order Flux Corrected Transport (FCT). **Calling Sequence:** subroutine advem_fct2(fld,fldc,u,v,fco,fcn,scal,scali,dt2) **Data Declaration:** real dt2, fld, fldc, u, v, scal, scali, fco, fcn **Common Blocks:** common/testpt/ **I/O:** stdout **Uses:** None
advem_fct4	Subroutine for leapfrog 4th order Flux Corrected Transport (FCT). **Calling Sequence:** subroutine advem_fct4(fld,fldc,u,v,fco,fcn,scal,scali,dt2) **Data Declaration:** real dt2, fld, fldc, u, v, scal, scali, fco, fcn **Common Blocks:** common/testpt/ **I/O:** stdout **Uses:** None
advem_mpdata	Subroutine for leapfrog 2nd order MPDATA with a combined monotone scheme. **Calling Subroutine:** subroutine advem_mpdata(fld,u,v,fco,fcn, posdef,scal, scali,dt2) **Data Declaration:** real posdef, dt2, fld, u, v, fco, fcn, scal, scali **Common Blocks:** common/testpt/ **I/O:** None **Uses:** None
advem_pcm	Piecewise Constant Method (Donor Cell, Upwind). It is spread over two time steps and may only require half the normal time step for stability. **Calling Sequence:** subroutine advem_pcm(fld,u,v,fco,fcn,scal,scali,dt2) **Data Declaration:** real dt2, fld, u, v, scal, scali, fco, fcn **Common Blocks:** common/testpt/ **I/O:** stdout **Uses:** None
tsdff_1x	Laplacian diffusion for a single scalar field **Calling Sequence:** subroutine tsdff_1x(fld1) **Data Declaration:** real fld1 **I/O:** stdout, text, textu, textv **Uses:**
tsdff_2x	Laplacian diffusion for two scalar fields. **Calling Sequence:** subroutine tsdff_2x(fld1,fld2) **Data Declaration:** real fld1, fld2 **I/O:** stdout, write text, textu, textv **Uses:** None

6.2 Atmospheric Forcing Subroutines

Subroutine	Description
dpthuv	Defines water depth (bottom pressure) at u, v points and barotropic potential vorticity. It calls no arguments or common blocks. **I/O:** stdout **Uses:** mod_xc
dpudpv	Defines layer depth at u, v points with a halo out to *margin_dpudpv*. **Calling Sequence:** subroutine dpudpv(dpu,dpv, p, depthu, depthv, margin_dpudpv) **Data Declaration:** integer margin_dpudpv real dpu, dpv, p, depthu, depthv **I/O:** stdout **Uses:** mod_xc
dpudpvj	Defines layer depth at u, v points in a single row. **Calling Sequence:** subroutine dpudpvj(dpu,dpv, p,depthu,depthv, margin_dpudpv, j) **Data Declaration:** integer margin_dpudpv, j real dpu, dpv, p, depthu, depthv **I/O:** None **Uses:** mod_xc
forday	Converts model day to "calendar" date (year, ordinal-day, hour). **Calling Sequence:** subroutine forday(dtime,yrflag, iyear,iday,ihour) **Data Declaration:** real dtime integer yrflag, iyear, iday, ihour **I/O:** stdout **Uses:** mod_xc
forfuna	Initializes input of atmospheric forcing fields. All input fields must be defined at all grid points. It requires no arguments. (I/O and array units 900-910 are reserved for the entire run.) **Common Blocks:** common/rdforfi/ **I/O:** stdout open, read units 900-910, 916 **Uses:** mod_xc, mod_za
forfundf	Initializes spatially varying *veldf2, veldf4,* and *thkdf4*, if necessary, and computes Laplacian and biharmonic diffusion coefficients. All units are m/s, and all are on the *p* grid. I/O unit 923 is not reserved. There are no arguments for this subroutine. **I/O:** stdout

Subroutine	Description
	open, close unit uoff+923 **Uses:** mod_xc, mod_za
forfunh	Processes high frequency atmospheric forcing fields. **Calling Sequence:** subroutine forfunh(dtime) **Data Declaration:** real dtime **I/O:** stdout open, read units uoff+900-910, 916 read cline rewind unit uoff+901 **Uses:** mod_xc, mod_za
forfunk	This subroutine initializes input of the *kpar* forcing field. Units of *akpar* are 1/m and *akpar* is always on the *p* grid. All input fields must be defined at all grid points. I/O and I/O unit 919 is reserved for the entire run. There are no arguments called in this subroutine. **Common Blocks:** common/rdforfi/ **I/O:** stdout open, read, close unit uoff+919 **Uses:** mod_xc, mod_za
forfunp	Initializes input of river (precipitation is bogus) forcing field. Units of rivers are m/s (positive into the ocean) and are always on the *p* grid. All input fields must be defined at all grid points. I/O and I/O unit 918 is reserved for the entire run. There are no arguments called. **Common Blocks:** common/rdforfi/ **I/O:** stdout open, read, close unit uoff+918 **Uses:** mod_xc, mod_za
forfunr	Initializes input of thermal/tracer relaxation forcing fields. I/O and array I/O units 911-914 are reserved for the entire run. I/O unit 915 is not reserved. All input fields must be defined at all grid points. No arguments are called. **Common Blocks:** common/rdforfi/ **I/O:** stdout open, read units uoff+911-915 **Uses:** mod_xc, mod_za
forfunt	Initializes spatially varying minimum depth for isopycnal layers. Units of *isotop* are m on the *p* grid. I/O and I/O unit 924 are not reserved. All input fields must be defined at all grid points. **I/O:** stdout open,close unit uoff+924 **Uses:** mod_xc, mod_za
forfunv	This routine initializes spatially varying isopycnal target densities. Units of

Subroutine	Description
	sigma are sigma-theta or sigma^2. Sigma is on the *p* grid. I/O unit 922 is not reserved. All input fields must be defined at all grid points. No arguments are called. **I/O:** stdout open, close unit uoff+922 **Uses:** mod_xc, mod_za
preambl_print	Prints non-blank lines of *preambl*. **Calling Sequence:** subroutine preambl_print(preambl) **Data Declaration:** character preambl **I/O:** stdout **Uses:** mod_xc, mod_za
rdbaro	Processing for baroclinic velocity nesting archive input. I/O and array I/O unit 921 is reserved for the entire run. **Calling Sequence:** subroutine rdbaro(dtime) **Data Declaration:** real dtime **I/O:** stdout **Uses:** mod_xc, mod_za
rdbaro_in	Input barotropic fields from archive on model day *dtime*. I/O and array I/O unit 921 is reserved for the entire run. **Calling Sequence:** subroutine rdbaro_in(dtime,lslot) **Data Declaration:** real dtime integer lslot **I/O:** stdout, write flnm open, read, close unit uoff+921, read cline **Uses:** mod_xc, mod_za
rdrivr	Reads river forcing for one month. **Calling Sequence:** subroutine rdrivr(mnth,lslot) **Data Declaration:** integer lslot,mnth **Common Blocks:** common/rdforfi/ **I/O:** stdout read unit iunit **Uses:** mod_xc, mod_za
rdforf	Reads forcing functions for one month. **Calling Sequence:** subroutine rdforf(mnth,lslot) **Data Declaration:** integer lslot,mnth **Common Blocks:** common/rdforfi/ **I/O:** stdout read, rewind unit uoff+iunit **Uses:** mod_xc, mod_za

Subroutine	Description
rdkpar	Reads *kpar* forcing for one month. **Calling Sequence:** subroutine rdkpar(mnth,lslot) **Data Declaration:** integer lslot,mnth **Common Blocks:** common/rdforfi/ **I/O:** stdout read unit uoff+iunit **Uses:** mod_xc, mod_za
rdmonth	Reads a single array field from unit *iunit* (see also *rdmonthck*). Ignores the input month if it falls between 1 and 12. iunit=900-910; atmospheric forcing field iunit=911-914; relaxation forcing field iunit=915; relaxation time scale field iunit=918; river forcing field iunit=919; *kpar* forcing field iunit=922; isopycnal target density field iunit=923; Laplacian or biharmonic diffusion velocity field iunit=924; minimum depth for isopycnal layers **Calling Sequence:** subroutine rdmonth (field, iunit) **Data Declaration:** integer iunit real field **I/O:** None **Uses:** mod_xc, mod_za
rdmonthck	This subroutine reads a single array field from unit iunit (see also *rdmonth*). It checks the input month against *mnthck*, if *mnthck*>0. **Calling Sequence:** subroutine rdmonthck(field, iunit, mnthck) **Data Declaration:** integer iunit, mnthck real field **I/O:** stdout read cline **Uses:** mod_xc, mod_za
rdnest	This is a 3D nesting archive input processing routine. I/O and array I/O unit 915 is used for *rmun[pv]* only (not reserved). I/O and array I/O unit 920 is reserved for the entire run. **Calling Sequence:** subroutine rdnest(dtime) **Data Declaration:** real dtime **I/O:** stdout open, read, close unit 915 **Uses:** mod_xc, mod_za
rdnest_in	This subroutine inputs 3D nesting fields from archive on model day *dtime*.

Subroutine	Description
	I/O unit 920 is reserved for the entire run. **Calling Sequence:** subroutine rdnest_in(dtime,lslot) **Data Declaration:** real dtime integer lslot **I/O:** stdout, write flnm open, read, close unit uoff+920, read cline **Uses:** mod_xc, mod_za
rdpall	This subroutine copies slot 2 into slot 1 and reads a set of high frequency forcing fields into slot 2. **Calling Sequence:** subroutine rdpall(dtime0,dtime1) **Data Declaration:** real dtime0, dtime1 **I/O:** stdout **Uses:** mod_xc
rdpall1	This subroutine copies *field*(:,:,2) into *field* (:,:,1) and reads a high frequency forcing field into *field* (:,:,2). On exit, *dtime* is the time (wind day) of the forcing field. **Calling Sequence:** subroutine rdpall1(field,dtime,iunit,lprint) **Data Declaration:** real field, dtime integer iunit logical lprint **I/O:** stdout, write cline read iunit, cline **Uses:** mod_xc, mod_za
rdrlax	This subroutine reads relaxation fields for one month, with monthly (*clmflg*==12) or bimonthly (*clmflg*==6) data. **Calling Sequence:** subroutine rdrlax(month,lslot) **Data Declaration:** integer lslot, month **Common Blocks:** common/rdforfi/ **I/O:** stdout read unit iunit **Uses:** mod_xc, mod_za
rd_archive	This subroutine reads a single archive array field from unit *iunit*. **Calling Sequence:** subroutine rd_archive(field, cfield, iunit) **Data Declaration:** character cfield integer iunit real field **I/O:** stdout read cline **Uses:** mod_xc, mod_za

Subroutine	Description
skmonth	This subroutine skips a single array field from unit *iunit*. **Calling Sequence:** subroutine skmonth(iunit) **Data Declaration:** integer iunit **I/O:** stdout read unit iunit **Uses:** mod_xc, mod_za
str2spd	This subroutine calculates wind speed from wind stress. Speed-dependent scale factor from stress to speed is based on the Kara (neutral) wind-speed dependent drag coefficient. **Calling Sequence:** subroutine str2spd(wspd, tx,ty) **Data Declaration:** real wspd, tx,ty **I/O:** None **Uses:** mod_xc, mod_za

6.3 Bathymetry Subroutines

Subroutine	Description
bigrid	This subroutine sets loop bounds for irregular basins in C-grid configuration. The variables *q*, *u*, *v*, and *p* are vorticity, u-velocity, v-velocity and mass points, respectively. The argument '*depth*' is a basin depth array and zero values indicate land. **Calling Sequence:** subroutine bigrid(depth, mapflg, util1,util2,util3) **Data Declaration:** real depth,util1,util2,util3 integer mapflg **I/O:** stdout write char3 **Uses:** mod_xc
geopar	This subroutine sets up model parameters related to geography. It calls no arguments or common blocks. **I/O:** stdout, write unit 6 open, read, close unit=uoff+9 read cline **Uses:** mod_xc, mod_za
indxi	Determines the *i* loop index corresponding to a land/sea mask. **Calling Sequence:** subroutine indxi(ipt,if,il,is) **Data Declaration:** integer ipt, if, il, is **I/O:** stdout **Uses:** mod_xc
indxj	Determines the *j* loop index corresponding to a land/sea mask.

Subroutine	Description
	Calling Sequence: subroutine indxj(jpt,jf,jl,js) **Data Declaration:** integer jpt, jf, jl, js **I/O:** stdout **Uses:** mod_xc

6.4 Communication Subroutines

There are two versions of the communication subroutines. The message passing interface (MPI) version (**mod_xc_mp.h**), which will be used at NAVOCEANO, is documented below. There is also a single shared memory version (**mod_xc_sm.h**) that uses the same application specific programming interface. The subroutines are virtually identical in each, with the exception of a few arguments that may have no effect in the single shared memory version.

Subroutine	Description
xcaget	Converts an entire 2D array from tiled to non-tiled layout. **Calling Sequence:** subroutine xcaget(aa, a, mnflg) **Data Declaration:** integer mnflg (node source flag) real aa (non-tiled source array, a (tiled source array) **Common Blocks:** common/xcmpii/, common/xcagetr/ **I/O:** None **Uses:** None
xcaget4	Converts an entire 2D array from tiled to non-tiled layout. This is a special version for *zaiord* and *zaiowr* only. Arrays are real*4 and tiled array has no halo. **Calling Sequence:** subroutine xcaget4(aa, a, mnflg) **Data Declaration:** integer mnflg real*4 aa, a **Common Blocks:** common/xcmpii/, common/xcagetr/ **I/O:** None **Uses:** None
xcaput	Converts an entire 2D array from non-tiled to tiled layout. The parameter *mnflg* selects which nodes must contain the non-tiled array: = 0; all nodes = n; node number n (*mnproc*=n) If *mnflg*.ne.0 the array *aa* may be broadcast to all nodes, therefore *aa* must exist and be overwritable on all nodes. **Calling Sequence:** subroutine xcaput(aa, a, mnflg) **Data Declaration:** integer mnflg real*4 aa, a

Subroutine	Description
	Common Blocks: common/xcmpii/, common/xcagetr/ **I/O:** stdout **Uses:** None
xcaput4	This subroutine converts an entire 2D array from non-tiled to tiled layout. This is a special version for *zaiord* and *zaiowr* only. Arrays are real*4 and the tiled array, *a,* has no halo. **Calling Sequence:** subroutine xcaput4(aa, a, mnflg) **Data Declaration:** integer mnflg real*4 aa, a **Common Blocks:** common/xcmpii/, common/xcagetr/ **I/O:** stdout **Uses:** None
xcastr	This subroutine broadcasts the array *a* to all tiles. **Calling Sequence: subroutine xcastr(a, mnflg)** **Data Declaration:** integer mnflg real a **Common Blocks:** common/xcmpii/, common/ xcmaxr4/ **I/O:** None **Uses:** None
xceget	This subroutine finds the value of *a*(*ia,ja*) on the non-tiled 2D grid. **Calling Sequence:** subroutine xceget(aelem, a, ia,ja) **Data Declaration:** integer ia, ja real aelem, a **Common Blocks:** common/xcmpii/, common/ xcegetr/ **I/O:** None **Uses:** None
xcgetc	This is a machine specific routine for broadcasting *iline*. It is only used in *zagetc*. **Calling Sequence:** subroutine xcgetc(iline) **Data Declaration:** integer integer iline **Common Blocks:** common/xcmpii/ **I/O:** None **Uses:** None
xceput	This subroutine fills a single element in the non-tiled 2D grid. **Calling Sequence:** subroutine xceput(aelem, a, ia,ja) **Data Declaration:** : integer ia, ja real aelem, a **Common Blocks:** common/xcmpii/ **I/O:** None

Subroutine	Description
	Uses: None
xchalt	This subroutine stops all processes. Only one process needs to call this routine because it is for emergency stops. Use *xcstop* for ordinary stops called by all processes. **Calling Sequence:** subroutine xchalt(cerror) **Data Declaration:** character cerror **Common Blocks:** common/xcmpii/ **I/O:** stdout, stdout **Uses:** None
xclget	This subroutine extracts a line of elements from the non-tiled 2D grid. **Calling Sequence:** subroutine xclget(aline,nl, a, i1,j1,iinc,jinc, mnflg) **Data Declaration:** integer nl, il, jl, iinc, jinc, mnflg real aline, a **Common Blocks:** common/xcmpii/, common/ xclgetr/ **I/O:** None **Uses:** None
xclput	This subroutine fills a line of elements in the non-tiled 2D grid. **Calling Sequence:** subroutine xclput(aline,nl, a, i1,j1,iinc,jinc) **Data Declaration:** integer nl, il, jl, iinc, jinc real aline, a **I/O:** None **Uses:** None
xclput4	This subroutine fills a line of elements in the non-tiled 2D grid. This is a special version for *xcaput4* only. **Calling Sequence:** subroutine xclput4(aline,nl, a, i1,j1,iinc,jinc) **Data Declaration:** integer nl, il, jl, iinc, jinc real aline, a **I/O:** None **Uses:** None
xcmaxr_0	This subroutine replaces scalar *a* with its element-wise maximum over all tiles. **Calling Sequence:** subroutine xcmaxr_0(a, mnflg) **Data Declaration:** integer mnflg real a **I/O:** None **Uses:** None
xcmaxr_0o	This subroutine replaces scalar *a* with its element-wise maximum over all tiles. **Calling Sequence:** subroutine xcmaxr_0o(a) **Data Declaration:** real a

Subroutine	Description
	I/O: None **Uses:** None
xcmaxr_1	This subroutine replaces array *a* with its element-wise maximum over all tiles. **Calling Sequence:** subroutine xcmaxr_1(a, mnflg) **Data Declaration:** integer mnflg real a **Common Blocks:** common/xcmpii/, common/ xcmaxr4/ **I/O:** None **Uses:** None
xcmaxr_1o	Replaces array *a* with its element-wise maximum over all tiles. **Calling Sequence:** subroutine xcmaxr_1o(a) **Data Declaration:** real a **I/O:** None **Uses:** None
xcminr_0	This subroutine replaces scalar *a* with its element-wise minimum over all tiles. **Calling Sequence:** subroutine xcminr_0(a, mnflg) **Data Declaration:** integer mnflg real a **I/O:** None **Uses:** None
xcminr_0o	This subroutine replaces scalar *a* with its element-wise minimum over all tiles. **Calling Sequence:** subroutine xcminr_0o(a) **Data Declaration:** real a **I/O:** None **Uses:** None
xcminr_1	Replaces array *a* with its element-wise minimum over all tiles. **Calling Sequence:** subroutine xcminr_1(a, mnflg) **Data Declaration:** integer mnflg real a **Common Blocks:** common/xcmpii/, common/xcmaxr4/ **I/O:** None **Uses:** None
xcminr_1o	Replaces array *a* with its element-wise minimum over all tiles. **Calling Sequence:** subroutine xcminr_1o(a) **Data Declaration:** real a **I/O:** None **Uses:** None
xcspmd	This routine initializes data structures that identify the tiles. All data structures are based on the processor number and the patch

Subroutine	Description
	distribution file, **patch.input**. A table of the data structures is found in Appendix C. **Calling Sequence:** subroutine xcspmd(mpi_comm_vm) **Data Declaration:** integer mpi_comm_vm **Common Blocks:** common/xcmpii/ **I/O:** stdout, open lp, unit 99 read unit 99 **Uses:** None
xcstop	This subroutine stops all processes. All processes must call this routine. Emergency stops use *xchalt*. **Calling Sequence:** subroutine xcstop(cerror) **Data Declaration:** character cerror **Common Blocks:** common/xcmpii/ **I/O:** stdout **Uses:** None
xcsum	This subroutine sums a 2D array, where mask = 1. **Calling Sequence:** subroutine xcsum(sum, a, mask) **Data Declaration:** real*8 sum, real a integer mask **Common Blocks:** common/xcmpii/, /xcsum8/ **I/O:** stdout **Uses:** None
xcsumj	Row sum of a 2D array, where mask ==1 on the first processor only. **Calling Sequence:** subroutine xcsumj(sumj, a,mask) **Data Declaration:** real*8 sumj, real a integer mask **Common Blocks:** common/xcmpii/, /xcsum8/ **I/O:** stdout **Uses:** None
xcsync	This routine is a barrier, with no processor exits until all arrive and flush stdout. Some MPI implementations only flush stdout as a collective operation, explaining the lflush=.true. option to flush stdout. Usually this is just a wrapper to the "Barrier" macro. **Calling Sequence:** subroutine xcsync(lflush) **Data Declaration:** logical lflush **Common Blocks:** common/xcmpii/ **I/O:** None **Uses:** None
xctbar	This routine syncs with processors *ipe1* and *ipe2*. It is a global collective

Subroutine	Description
	operation, and the calls on *ipe1*and *ipe2* must list this processor as one of the two targets. This routine is used in place of a global barrier in halo operations, but it only provides synchronization of one or two processors with the local processor. The parameters *ipe1* and/or *ipe2* can be null_tile to indicate no processor. **Calling Sequence:** subroutine xctbar(ipe1,ipe2) **Data Declaration:** integer ipe1, ipe2 **Common Blocks:** common/halobp/ **I/O:** None **Uses:** None
*xctilr**	Updates the tile overlap halo of a real array. This is a recursive subroutine. **Calling Sequence:** recursive subroutine xctilr(a,l1,ld,mh,nh,itype) **Data Declaration:** integer l1, ld, mh, ng, itype real a **Common Blocks:** common/xctilr4/, common/xctilra, common/xcmpii **I/O:** stdout **Uses:** None
*xctmr**	This subroutine updates the tile overlap halo of a real array. This is a recursive subroutine. **Calling Sequence:** recursive subroutine xctilr(a,l1,ld,mh,nh,itype) **Data Declaration:** integer l1, ld, mh, nh, itype real a **Common Blocks:** common/xctilr4/, common/xcmpii **I/O:** stdout **Uses:** None
xctmri	This subroutine initializes timers. It times every 50^{th} event above 1000. No arguments are called. **I/O:** None **Uses:** None
xctmr0	This subroutine starts timer n. **Calling Sequence:** subroutine xctmr0(n) **Data Declaration:** integer n **I/O:** stdout **Uses:** None
xctmr1	Adds time to timer n since call to *xctim0*. Times every 50^{th} event above 1000. **Calling Sequence:** subroutine xctmr1(n) **Data Declaration:** integer n **I/O:** stdout **Uses:** None

Subroutine	Description
xctmrn	This subroutine registers the name of timer n. **Calling Sequence:** subroutine xctmrn(n,cname) **Data Declaration:** integer n character cname **I/O:** stdout **Uses:** None
xctmrp	Prints all active timers on all processors. Upon exit, all timers are reset to zero. There are no arguments called. **Common Blocks:** common/xcmpii **I/O:** stdout **Uses:** None

6.5 Diagnostic Output Subroutines

Subroutine	Description
overtn	Diagnoses meridional heat flux in the basin model. **Calling Sequence:** subroutine overtn(dtime,dyear) **Data Declaration:** real dtime, dyear **I/O:** stdout, write noo print 999 **Uses:** mod_xc

6.6 Diapycnal Mixing Subroutines (diapfl.f)

Subroutine	Description
diapf1	KPP-style implicit interior diapycnal mixing for shear instability, double diffusion, and background internal waves. **Calling Sequence:** subroutine diapf1(m,n) **Data Declaration:** integer m,n **I/O:** None **Uses:** mod_xc
diapflaj	**Calling Sequence:** subroutine diapf1aj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** None **Uses:** mod_xc
diapflbj	**Calling Sequence:** subroutine diapf1bj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** None

Subroutine	Description
	Uses: mod_xc
diapf1aij	KPP-style implicit interior diapycnal mixing subroutine, with a single i,j point (part A). **Calling Sequence:** subroutine diapf1aij(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** None **Uses:** mod_xc
diapf1uij	Diapycnal mixing subroutine, with a single i,j point and momentum at u grid points. **Calling Sequence:** subroutine diapf1uij(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** None **Uses:** mod_xc
diapf1vij	Diapycnal mixing subroutine, with a single i,j point and momentum at v grid points. **Calling Sequence:** subroutine diapf1vij (m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** None **Uses:** mod_xc
diapf2j	Diapycnal mixing subroutine, with a single j-row. **Calling Sequence:** subroutine diapf2j(m,n, j) **Data Declaration:** integer m,n,j **I/O:** stdout **Uses:** mod_xc
diapf3j	Diapycnal mixing routine with a single j-row. **Calling Sequence:** subroutine diapf3j(m,n, j) **Data Declaration:** integer m,n,j **I/O:** stdout **Uses:** mod_xc

6.7 Hybrid Grid Generation Subroutines (hybgen.f)

Subroutine	Description
hybenaj	Part A of the hybrid grid generator, single j-row. **Calling Sequence:** subroutine hybgenaj(m,n,j) **Data Declaration:** integer m,n, j **I/O:** stdout, write cinfo **Uses:** mod_xc

Subroutine	Description
hybgenbj	Part A of the hybrid grid generator, single j-row. **Calling Sequence:** subroutine hybgenbj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** stdout **Uses:** mod_xc

6.8 Incremental Updating (for Data Assimilation) Subroutines (mod_incupd.f)

Subroutine	Description
incupd	Updates HYCOM 2.2 variables with increments. **Calling Sequence:** subroutine incupd(n) **Data Declaration:** integer n **I/O:** stdout **Uses:** None
incupd_init	Subroutine used to calculate the increment field for incremental updating. **Calling Sequence:** subroutine incupd_init(dtime0) **Data Declaration:** real dtime0 **I/O:** None **Uses:** None
incupd_read	Inputs 3D HYCOM fields (from an archive file) on model day *dtime*. It calculates the increment between the input and the initial state. **Calling Sequence:** subroutine incupd_read(dtime) **Data Declaration:** real dtime **I/O:** stdout, write flnm open, read, close unit uoff+925, read cline **Uses:** mod_za
incupd_rd	Subroutine used to calculate the increment field for incremental updating. **Calling Sequence:** subroutine incupd_rd(dtime0) **Data Declaration:** real dtime0 **I/O:** stdout **Uses:** None

6.9 Initialization and Restart Subroutines

Subroutine	Description
blkdat	Initializes common variables. It calls no arguments. **I/O:** stdout open, read, close unit uoff+99

Subroutine	Description
	Uses: mod_xc, mod_incupd, mod_floats, mod_tides
blkin8	Reads in one real*8 value on unit 99. **Calling Sequence:** subroutine blkin8(rvar,cvar,cfmt) **Data Declaration:** real rvar character cvar, cfmt **I/O:** stdout read unit uoff+99 **Uses:** mod_xc
blkini	Reads in one named integer value on unit 99. **Calling Sequence:** subroutine blkini(ivar,cvar) **Data Declaration:** integer ivar character cvar **I/O:** stdout read unit uoff+99 **Uses:** mod_xc
blkinl	Reads in one named logical value on unit 99. **Calling Sequence:** subroutine blkinl(lvar,cvar) **Data Declaration:** logical lvar character cvar **I/O:** stdout read in an integer 0= F, 1= T, read unit uoff+99 **Uses:** mod_xc
blkinr	This subroutine reads in one named real value on unit 99. **Calling Sequence:** subroutine blkinr(rvar,cvar,cfmt) **Data Declaration:** real rvar character cvar, cfmt **I/O:** stdout read unit uoff+ 99 **Uses:** mod_xc
inicon	This subroutine initializes all fields (except tracers. See *initrc*). **Calling Sequence:** subroutine inicon(mnth) **Data Declaration:** integer mnth **I/O:** stdout, write ptxt, utxt, vtxt, text, unit 6 **Uses:** mod_xc, mod_pipe
inigiss	This subroutine initializes NASA GISS vertical mixing scheme. No arguments are called. **I/O:** stdout open, close unit uoff+98 **Uses:** mod_xc

Subroutine	Description
inikpp	Initializes the Large, McWilliams, and Doney KPP vertical mixing scheme (see Section 5.2.7.1). No arguments are called. **Common Blocks:** common/kppltr/ **I/O:** None **Uses:** mod_xc
inimy	Initializes the Mellor-Yamada level 2.5 vertical mixing scheme (see Section 5.2.7.2). No arguments are called. **I/O:** None **Uses:** mod_xc
restart_in	This subroutine reads in a restart file. **Calling Sequence:** subroutine restart_in(nstep0, dtime0) **Data Declaration:** integer nstep0 real dtime0 **I/O:** open unit uoff+11 read cline rewind unit uoff+11 stdout **Uses:** mod_xc, mod_za
restart_in3d	This subroutine reads a single restart 3D array field. **Calling Sequence:** subroutine restart_in3d(field,l, mask, cfield) **Data Declaration:** integer l, mask real field character cfield **I/O:** read cline stdout **Uses:** mod_xc, mod_za
restart_out	Writes out in a restart file on unit 12 or 22 (and a flux file on 25). **Calling Sequence:** subroutine restart_out(nstepx, dtimex, last) **Data Declaration:** integer nstepx logical last real dtimex **I/O:** stdout, write iunit, cline, unit uoff+25 open iunit, unit uoff+25 rewind iunit **Uses:** mod_xc, mod_za

6.10 K-Profile Vertical Mixing Subroutines

Subroutine	Description
mlbdep	This subroutine diagnoses the depth of the PWP mixed layer base and homogenizes to that depth. **Calling Sequence:** subroutine mlbdep(t1d,s1d,th1d,tr1d,u1d,v1d, p1d, dp1d, kmlb,kmax) **Data Declaration:** integer kmlb, kmax real t1d, s1d, th1d, tr1d, u1d, v1d, p1d, dp1d **I/O:** None **Uses:** mod_xc
mxkprfaj	This subroutine calculates viscosity and diffusivity. **Calling Sequence:** subroutine mxkprfaj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** None **Uses:** mod_xc
mxkprfbj	This subroutine performs final mixing at *p* points. **Calling Sequence:** subroutine mxkprfbj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** None **Uses:** mod_xc
mxkprfcj	This subroutine updates HYCOM 2.2 variables with increments. **Calling Sequence:** subroutine mxkprfcj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** None **Uses:** mod_xc
mxkppaij	KPP vertical diffusion, single j-row (part A). The vertical coordinate is *z* negative below the ocean surface. Quadratic interpolation is implemented by three-point collocation. **Calling Sequence:** subroutine mxkppaij(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** stdout **Uses:** mod_xc
mxmyaij	This subroutine performs Mellor-Yamada 2.5 vertical diffusion, single j-row (part A). The vertical coordinate is *z* negative below the ocean surface. **Calling Sequence:** subroutine mxmyaij(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** stdout **Uses:** mod_xc
mxgissaij	This routine performs the NASA GISS vertical mixing model, single j-row (part A). This is a level 2 turbulence model. **Calling Sequence:** subroutine mxgissaij(m,n, i,j)

Subroutine	Description
	Data Declaration: integer m,n, i,j **Common Blocks:** common/mxgissij_b/ **I/O:** stdout, write unit 6 **Uses:** mod_xc
mxkprfbij	This subroutine performs K-profile vertical diffusion, single j-row (part B). The vertical coordinate is *z* negative below the ocean surface. It performs the final vertical mixing at *p* points. **Calling Sequence:** subroutine mxkprfbij(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** stdout **Uses:** mod_xc
mxkprfciju	K-profile vertical diffusion subroutine, single j-row, with momentum at u grid points. The vertical coordinate is *z* negative below the ocean surface. **Calling Sequence:** subroutine mxkprfciju(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** stdout **Uses:** mod_xc
mxkprfcijv	K-profile vertical diffusion subroutine, single j-row, with momentum at v grid points. The vertical coordinate is *z* negative below the ocean surface. **Calling Sequence:** subroutine mxkprfcijv(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** stdout **Uses:** mod_xc
mxkrtaaj	HYCOM 2.2 slab mixed layer model, single row, part A. **Calling Sequence:** subroutine mxkrtaaj(m,n, j, depnew) **Data Declaration:** integer m,n, j real depnew **I/O:** stdout **Uses:** mod_xc
mxkrtabj	HYCOM 2.2 slab mixed layer model, single row, part B. **Calling Sequence:** subroutine mxkrtabj(m,n, j, depnew) **Data Declaration:** integer m,n, j real depnew **I/O:** None **Uses:** mod_xc
mxkrtbaj	Alternative slab mixed layer model, single row, part A. **Calling Sequence:** subroutine mxkrtbaj(m,n, j) **Data Declaration:** integer m,n **I/O:** stdout

Subroutine	Description
	Uses: mod_xc
mxkrtbbj	Alternative slab mixed layer model, single row, part B. **Calling Sequence:** subroutine mxkrtbbj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** stdout, unit 100 **Uses:** mod_xc
mxkrtmaj	**Calling Sequence:** subroutine mxkrtmaj(m,n, sdot, j) **Data Declaration:** **I/O:** None **Uses:** mod_xc
mxkrtmbj	**Calling Sequence:** subroutine mxkrtmbj(m,n, sdot, j) **Data Declaration:** integer m,n, j real sdot **I/O:** stdout **Uses:** mod_xc
mxpwpaj	This subroutine calculates viscosity and diffusivity. **Calling Sequence:** subroutine mxpwpaj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** None **Uses:** mod_xc
mxpwpbj	Final velocity mixing at *u,v* points. **Calling Sequence:** subroutine mxpwpbj(m,n, j) **Data Declaration:** integer m,n, j **I/O:** None **Uses:** mod_xc
mxpwpaij	PWP vertical mixing, single j-row (part A). **Calling Sequence:** subroutine mxpwpaij(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** stdout, write unit 6 **Uses:** mod_xc
mxpwpbiju	PWP vertical diffusion, single j-row (part A), momentum at u grid points. **Calling Sequence:** subroutine mxpwpbiju(m,n, i,j) **Data Declaration:** integer m,n, i,j **I/O:** None **Uses:** mod_xc
mxpwpbijv	PWP vertical diffusion, single j-row (part A), momentum at v grid points. **Calling Sequence:** subroutine mxpwpbijv(m,n, i,j) **Data Declaration:** integer m,n, i,j

Subroutine	Description
	I/O: None **Uses:** mod_xc
wscale	This subroutine computes turbulent velocity scales for the KPP mixing scheme. The vertical coordinate is *z* negative below the ocean surface. **Calling Sequence:** subroutine wscale(i,j,zlevel,dnorm,bflux,wm,ws,isb) **Data Declaration:** integer i,j, isb real zlevel, dnorm, bflux, wm, ws **Common Blocks:** common/kppltr/ **I/O:** None **Uses:** mod_xc

6.11 Lateral Boundary Condition Subroutines (latbdy.f)

Subroutine	Description
latbdf	Applies lateral boundary conditions to the barotropic flow field. This is a port flow version and is not related to the standard HYCOM 2.2 open boundary condition. This version uses algorithms based on a 1 invariant Flather boundary condition (sets the gradient of the incoming characteristic to zero). The tangential velocity is not constrained. Note that East, West, North and South refer to the grid (i.e., i,j) points and not the geographic East, West North and South. **Calling Sequence:** subroutine latbdf(n,lll) **Data Declaration:** integer n,lll **I/O:** open, read, close unit uoff+99 stdout, write char3, fmt **Uses:** mod_xc, mod_tides
latbdp	Applies lateral boundary conditions to the barotropic flow field. This is a port flow version that is similar to the standard "Browning and Kreiss (1982)" HYCOM 2.2 open boundary condition, except that the exterior normal velocity is constant in time and exterior pressure equals interior pressure. Tangential velocity is not constrained. **Calling Sequence:** subroutine latbdp(n) **Data Declaration:** integer n **I/O:** stdout, write char3, fmt open, close unit uoff+99 **Uses:** mod_xc
latbdt	Applies lateral boundary conditions to the barotropic flow field. This is the nested sub-region version and uses the "Browning and Kreiss (1982)" open boundary condition. **Calling Sequence:** subroutine latbdt(n,lll)

Subroutine	Description
	Data Declaration: integer n,lll **I/O:** stdout, write char3, fmt open, close unit uoff+99 **Uses:** mod_xc

6.12 Machine Dependent I/O Subroutines

There are two versions of machine dependent I/O subroutines. The message passing interface (MPI) version (**mod_za_mp1.h**), which will be used at NAVOCEANO, is documented below. There is also a single shared memory version (**mod_za_sm.h**) that uses the same application-specific programming interface. The subroutines are virtually identical in each, with the exception of a few arguments that may have no effect in the single shared memory version.

Subroutine	Description
flush	This is a wrapper for flush system call under AIX. It is also used to disable the flush system call under Intel's IFC compiler (if defined IFC). **Calling Sequence:** subroutine flush(iunit) **Data Declaration:** integer iunit **I/O:** None **Uses:** None
getenv	This subroutine provides *getenv* functionality on the T3E, using *pxfgetenv*. **Calling Sequence:** subroutine getenv(cname, cvalue) **Data Declaration:** character cname, cvalue **I/O:** None **Uses:** None
ieee_retrospective	This is a dummy routine to turn off IEEE warning messages on a Sun system. No arguments are called. **I/O:** None **Uses:** None
machine	This is a machine specific initialization routine called at the beginning of the program. No arguments are called. **I/O:** None **Uses:** None
x1flush	This subroutine a wrapper for the flush system call on the Cray X1. **Calling Sequence:** subroutine x1flush(iunit) **Data Declaration:** integer iunit **I/O:** None **Uses:** None
zagetc	This is a machine specific routine for reading one-text lines from a file. The

Subroutine	Description
	read is performed on the first processor only. **Calling Sequence:** subroutine zagetc(cline, ios, iunit) **Data Declaration:** character cline integer ios, iunit **Common Blocks:** common/ czgetc/ **I/O:** read inuit, ios **Uses:** None
zaio_endian	This subroutine swaps the Endian-ness of the array. It assumes integer(kind=1) and integer(kind=4) occupy one and four bytes, respectively. **Calling Sequence:** subroutine zaio_endian(a,n) **Data Declaration:** integer a, n **I/O:** None **Uses:** None
zaiocl	This subroutine performs array I/O file closing. **Calling Sequence:** subroutine zaiocl(iaunit) **Data Declaration:** integer iaunit **Common Blocks:** common/czioxx/ **I/O:** stdout **Uses:** None
zaiofl	This subroutine performs array I/O buffer flushing. **Calling Sequence:** subroutine zaiofl(iaunit) **Data Declaration:** integer iaunit **Common Blocks:** common/czioxx/ **I/O:** stdout open unit iaunit+uaoff **Uses:** None
zaioiq	Machine specific routine for array I/O inquiry. **Calling Sequence:** subroutine zaioiq(iaunit, irec) **Data Declaration:** integer iaunit, irec **Common Blocks:** common/czioxx/ **I/O:** None **Uses:** None
zaiope	This subroutine opens a file for array I/O. **Calling Sequence:** subroutine zaiope(cenv,cstat, iaunit) **Data Declaration:** character cenv, cstat integer iaunit **Common Blocks:** common/czioxx/, common/czioxw/ **I/O:** stdout open unit iaunit+uaoff

Subroutine	Description
	Uses: None
zaiopf	Routine for opening a file for array I/O. **Calling Sequence:** subroutine zaiopf(cfile,cstat, iaunit) **Data Declaration:** character cfile, cstat integer iaunit **Common Blocks:** common/czioxx/, common/czioxw/ **I/O:** stdout open unit iaunit+uaoff **Uses:** None
zaiopi	This subroutine looks to see if an array I/O is already open. **Calling Sequence:** subroutine zaiopi(lopen, iaunit) **Data Declaration:** integer iaunit logical lopen **Common Blocks:** common/czioxx/ **I/O:** None **Uses:** None
zaiopn	Routine for opening a file for array I/O. Array I/O is Fortran real*4 direct access I/O to unit iaunit+uaoff. **Calling Sequence:** subroutine zaiopn(cstat, iaunit) **Data Declaration:** character cstat integer iaunit **Common Blocks:** common/czioxx/, common/czioxw/ **I/O:** stdout, write cenv, cfile, unit 6, iaunit open iaunit + uaoff **Uses:** None
zaiordd	This subroutine performs a direct access read of a single record. It is expressed as a subroutine because I/O with implied do loops can be slow on some machines. **Calling Sequence:** subroutine zaiordd(a,n, iunit,irec,ios) **Data Declaration:** integer n, iunit, irec, ios real a **I/O:** read unit iunit, irec, ios **Uses:** None
zaiord	This routine performs array reading. **Calling Sequence:** subroutine zaiord(h, mask,lmask, hmin,hmax, iaunit) **Data Declaration:** integer iaunit, mask real hmin, hmax, h logical lmask **Common Blocks:** common/czioxx/, common/czioxw/, common/czioxr/

Subroutine	Description
	I/O: stdout inquire unit iaunit+uaoff **Uses:** None
zaiord3	This routine performs 3D array reading. **Calling Sequence:** subroutine zaiord3(h, l, mask,lmask, hmin,hmax, iaunit) **Data Declaration:** integer l, iaunit, mask logical lmask real hmin, hmax, h **I/O:** None **Uses:** None
zaiorw	Routine for array I/O file rewinding. **Calling Sequence:** subroutine zaiorw(iaunit) **Data Declaration:** integer iaunit **Common Blocks:** common/czioxx/ **I/O:** stdout **Uses:** None
zaiosk	This is a routine for skipping an array read. **Calling Sequence:** subroutine zaiosk(iaunit) **Data Declaration:** integer iaunit **Common Blocks:** common/czioxx/ **I/O:** stdout **Uses:** None
zaiost	This subroutine initializes array I/O. No arguments are called. **I/O:** stdout **Uses:** None
zaiowrd	This subroutine performs a direct access write for a single record. **Calling Sequence:** subroutine zaiowrd(a,n, iunit,irec,ios) **Data Declaration:** integer n, iunit, irec, ios real a **I/O:** write unit iunit, irec, ios **Uses:** None
zaiowr	**Calling Sequence:** subroutine zaiowr(h, mask,lmask, hmin,hmax,iaunit, lreal4) **Data Declaration:** integer iaunit, mask logical lmask, lreal4 real hmin, hmax, h **Common Blocks:** common/czioxx/, common/czioxw/, common/czioxr/ **I/O:** stdout inquire unit iaunit+uaoff

Subroutine	Description
	Uses: None
zaiowr3	This subroutine performs 3D array writing. **Calling Sequence:** subroutine zaiowr3(h, l, mask,lmask, hmin,hmax, iaunit, lreal4) **Data Declaration:** integer l, iaunit, mask logical lmask, lreal4 real hmin, hmax, h **I/O:** None **Uses:** None

6.13 Matrix Inversion Subroutines

These are matrix inversion subroutines for the implicit solution of the vertical diffusion equation for a tri-diagonal matrix.

Subroutine	Description
f_invmtx	Matrix inversion code used in the parameter matrix objective analysis algorithm of Mariano and Brown (1994), specifically in the code to compute the large scale 2D trend surface. Found in **mod_floats.f**. **Calling Sequence:** subroutine f_invmtx (a,na,v,nv,n,d,ip,ier) **Data Declaration:** integer na,nv,n,ip(2*n),ier dimension a(na,n),v(nv,n) **I/O:** print 115, 116 **Uses:** None
tridcof	Computes coefficients for tridiagonal matrix (dimension=*kdm*). Note: *tcu*(1) = 0 and *tcl*(*kdm*+1) = 0 are necessary conditions. **Calling Sequence:** subroutine tridcof(diff,tri,nlayer,tcu,tcc,tcl) **Data Declaration:** real diff, tri, tcu, tcc, tcl integer nlayer **I/O:** None **Uses:** mod_xc
tridrhs	Computes the right hand side of tridiagonal matrix for scalar fields: = yo (old field) + flux-divergence of *ghat,* and + flux-divergence of non-turbulant fluxes **Calling Sequence:** subroutine tridrhs(h,yo,diff,ghat,ghatflux,tri,nlayer,rhs) **Data Declaration:** real h, yo, diff, ghat, ghatflux integer nlayer **I/O:** None

Subroutine	Description
	Uses: mod_xc
tridmat	Solves the tridiagonal matrix for new vector *yn*, given right hand side vector *rhs*. Note: If the surface and bottom fluxes are nonzero, the following must apply: -surface layer needs +*delt1***surfaceflux*/(h(1)*bet), and -bottom layer needs +*tri*(*nlayer*,1)**diff*(*nlayer*+1)**yo*(*nlayer*+1))/*bet* **Calling Sequence:** subroutine tridmat(tcu,tcc,tcl,nlayer,h,rhs,yo,yn,diff) **Data Declaration:** real tcu, tcc, tcl, h, rhs, yo, diff, gam, bet integer nlayer **I/O:** stdout **Uses:** mod_xc

6.14 HYCOM 2.2 Module Subroutines (module mod_hycom.f, mod_OICPL.f)

Subroutine	Description
Archive_ESMF	Creates a HYCOM 2.2 "archive-like" file from Import/Export state. The Import state may not be at the same time as the Export state. **Calling Sequence:** subroutine Archive_ESMF(iyear,iday,ihour) **Data Declaration:** integer iyear,iday,ihour **I/O:** open, write, close nop, write cfile **Uses:** None
Export_ESMF	This subroutine fills the export state and calculates SSFI “in place”. No arguments are called. **I/O:** None **Uses:** None
HYCOM_Final	This subroutine creates a report, destroys the internal ocean clock, and prints active timers at the end of the HYCOM 2.2 run. **Calling Sequence:** subroutine HYCOM_Final (gridComp, impState,expState, extClock, rc) **Data Declaration:** type gridComp, impState, expState, extClock integer rc **I/O:** write nod **Uses:** None
HYCOM_Init	Initializes HYCOM 2.2 before the first time step. No arguments are called. **I/O:** stdout, write msg, intvl, nod open, read, close unit uoff+99, nod **Uses:** None
HYCOM_SetServices	**Calling Sequence:** subroutine HYCOM_SetServices(gridComp, rc) **Data Declaration:** type gridComp

Subroutine	Description
	integer rc **I/O:** None **Uses:** None
HYCOM_Run	This subroutine executes a single timestep. **I/O:** stdout, write nod, intvl **Uses:** None
Import_ESMF	This subroutine extracts the import state. No arguments are called. **I/O:** None **Uses:** None
mod_hycom	HYbrid Coordinate Ocean Model version 2.2. **I/O:** None **Uses:** ESMF_Mod, mod_xc, mod_za, mod_pipe, mod_incupd, mod_floats, mod_tides
OICPL_Final	**Calling Sequence:** subroutine OICPL_Final(cplComp, impState, expState, extClock, rc) **Data Declaration:** type cplComp, impState, expState, extClock integer rc **I/O:** None **Uses:** None
OICPL_Init	Initializes the OICPL structure for the ESMF framework. **Calling Sequence:** subroutine OICPL_Init(cplComp, impState, expState, extClock, rc) **Data Declaration:** type cplComp, impState, expState, extClock integer rc **I/O:** None **Uses:** None
OICPL_Run_I2O	OICPL run routine for transferring from the ice state to the ocean state. **Calling Sequence:** subroutine OICPL_Run_I2O(cplComp, impState,expState, extClock, rc) **Data Declaration:** type cplComp, impState, expState, extClock integer rc **I/O:** write msg **Uses:** None
OICPL_Run_O2I	OICPL run routine for transferring from the ocean state to the ice state. **Calling Sequence:** subroutine OICPL_Run_O2I(cplComp, impState,expState, extClock, rc) **Data Declaration:** type cplComp, impState, expState, extClock integer rc **I/O:** write msg

Subroutine	Description
	Uses: None
OICPL_SetServices	**Calling Sequence:** subroutine OICPL_SetServices(cplComp, rc) **Data Declaration:** type cplComp integer rc **I/O:** None **Uses:** None
Setup_ESMF	This subroutine sets up ESMF data structures for HYCOM 2.2. **Calling Sequence:** subroutine Setup_ESMF(gridComp, impState, expState, extClock, rc) **Data Declaration:** type gridComp, impState, expState, extClock integer rc **I/O:** None **Uses:** None

6.15 Pipe Comparison Subroutines (mod_pipe.f)

These routines represent the HYCOM 2.2 pipe based debugging interface. They facilitate output comparisons from two HYCOM 2.2 versions running concurrently. One model, the 'slave', writes its output into a named pipe. The other model, the 'master', reads from the pipe and makes comparisons. Differences are recorded in **PIPE_base.out**.

Subroutine	Description
pipe_init	This subroutine is called at the start of main program. It allocates arrays for comparison and opens the pipe and some output files. No arguments are called. **I/O:** stdout open lpunit, ipunit, open, read, close unit 17 inquire iolength=irecl **Uses:** None
pipe_compare	This routine may be called from anywhere in the code (from both versions, of course) to check whether data stored in a single array *'field'* are identical. **Calling Sequence:** subroutine pipe_compare(field,mask,what) **Data Declaration:** real field integer mask character what **I/O:** stdout, write lpunit, ipunit read ipunit **Uses:** mod_xc
pipe_comparall	**Calling Sequence:** subroutine pipe_comparall(m,n, cinfo)

Subroutine	Description
	Data Declaration: integer m, n character cinfo **I/O:** stdout, write lpunit, cformat, txt1, txt2 **Uses:** mod_xc
pipe_compare_notneg	This routine is called from anywhere in the model code to check whether data stored in the single array *field* is non-negative. This is typically a tracer field. This routine is only active in PIPE_TRACER mode. **Calling Sequence:** subroutine pipe_compare_notneg(field,mask,what) **Data Declaration:** real field integer mask character what **I/O:** stdoutunit **Uses:** mod_xc
pipe_compare_sym1	This routine is called from anywhere in the model code to check whether data stored in the single array *field* is symmetric. This is passed through to *pipe_compare* when in master/slave mode. **Calling Sequence:** subroutine pipe_compare_sym1(field,mask,what) **Data Declaration:** real field integer mask character what **I/O:** write lpunit **Uses:** mod_xc
pipe_compare_sym2	**Calling Sequence:** subroutine pipe_compare_sym2(field_u,mask_u,what_u, field_v,mask_v,what_v) **Data Declaration:** real field_u, field_v integer mask_u, mask_v character what_u, what_v **I/O:** write lpunit **Uses:** mod_xc
pipe_compare_same	This subroutine is called from anywhere in the code to check whether data stored in *fielda* and *fieldb* are identical. Usually fielda is temperature and fieldb is a temperature tracer. This subroutine is only active in PIPE_TRACER mode. **Calling Sequence:** subroutine pipe_compare_same(fielda,fieldb,mask,what) **Data Declaration:** real fielda, fieldb integer mask character what **I/O:** write lpunit **Uses:** mod_xc

6.16 Plotting Subroutines

Subroutine	Description
prtmsk	Deletes *array* elements outside *mask*. Then breaks *array* into sections, each *nchar* characters wide, for printing. **Calling Sequence:** subroutine prtmsk(mask,array,work, idm,ii,jj,offset,scale, title) **Data Declaration:** integer idm, ii, jj, mask real array, work, offset, scale character title **Common Blocks:** common/linepr/ **I/O:** stdout **Uses:** None
psmooth	A ragged boundary version of the basic 9-point smoothing routine. This routine is set up to smooth data carried at *p* points. **Calling Sequence:** subroutine psmooth(a,margin_smooth) **Data Declaration:** integer margin_smooth real a **I/O:** None **Uses:** mod_xc
psmooth_dif	This routine returns the maximum of the original and smoothed value and ignores locations where $k > aklist(i,j)$. **Calling Sequence:** subroutine psmooth_dif(a,aklist,k,margin_smooth) **Data Declaration:** integer k, margin_smooth real a, aklist **I/O:** None **Uses:** mod_xc
psmooth_ice	This routine smoothes *covice*=1.0 and *covice*=0.0 regions separately, leaving areas with fractional *covice* untouched. Note that *covice* must be valid in the halo out to *margin_smooth*. **Calling Sequence:** subroutine psmooth_ice(a,margin_smooth) **Data Declaration:** integer margin_smooth real a **I/O:** None **Uses:** mod_xc
psmooth_new	A ragged boundary version of the basic 9-point smoothing routine. Input is in *a*, output is in *b*. The variables *a* and *b* must not be the same array. **Calling Sequence:** subroutine psmooth_new(a,b,margin_smooth) **Data Declaration:** integer margin_smooth

Subroutine	Description
	real a, b **I/O:** None **Uses:** mod_xc

6.17 Synthetic Float, Drifter and Mooring Subroutines (module mod_floats.f)

Subroutine	Description
floats	HYCOM 2.2 synthetic floats, drifters and moorings. Optionally samples time series of dynamical/thermodynamical variables at the location of each float. See Section 5.2.4 for more information. **Calling Sequence:** subroutine floats(m,n,timefl,ioflag) **Data Declaration:** integer m,n, ioflag real timefl **I/O:** stdout open, write unit 801 **Uses:** None
floats_init	Reads initial float data and initializes parameters and arrays required for floats. **Calling Sequence:** subroutine floats_init(m,n,time0) **Data Declaration:** integer m, n real time0 **I/O:** stdout open, read unit uoff+99 **Uses:** None
floats_restart	This subroutine outputs a float restart file. It calls no arguments. **I/O:** stdout, open, write, close unit uoff+802 **Uses:** None
f_stat	Computes mean, variance, and standard deviation of a data sequence. **Calling Sequence:** subroutine f_stat(ser,ls,amean,var,std) **Data Declaration:** dimension ser **I/O:** None **Uses:** None
intrph	This is a 2D polynomial or nearest neighbor interpolation. **Calling Sequence:** subroutine intrph(varb2d,ptlon,ptlat,fllon,fllat,maskpt, ngood,ngrid,intpfl,radian,vrbint,ier) **Data Declaration:** real varb2d, ptlon, ptlat, fllon, fllat, vrbint, radian integer ngood, ngrid, intpfl, ier logical maskpt

Subroutine	Description
	I/O: write unit 6 **Uses:** None
rantab	Every time this routine is called one of the 200 records in *rtab* is randomly selected and new random numbers are inserted. **Calling Sequence:** subroutine rantab (rtab,iseed1,iseed2,numran) **Data Declaration:** real rtab integer iseed1, iseed2 **I/O:** None **Uses:**
rantab_ini	Retrieves the two seeds, with *iseed1* representing the 'usual' random number generator (*udev_rt*) and *iseed2* representing the 'Gaussian' random number generator (*grand_rt*). **Calling Sequence:** subroutine rantab_ini(rtab,iseed1,iseed2,iflag) **Data Declaration:** real rtab integer iseed1, iseed2, iflag **I/O:** stdout **Uses:** None

6.18 Thermal Forcing Subroutines

Subroutine	Description
thermf	Thermal forcing subroutine. Upon exiting, the flux is for the ocean fraction of each grid cell. **Calling Sequence:** subroutine thermf(m,n) **Data Declaration:** integer m, n **I/O:** stdout **Uses:** mod_xc
thermf_oi	Thermal forcing subroutine with optimal interpolation. It combines ocean and sea ice surface fluxes and completes surface salinity forcing. **Calling Sequence:** subroutine thermf_oi(m,n) **Data Declaration:** integer m, n **I/O:** None **Uses:** mod_xc
thermfj	Thermal forcing subroutine for the ocean surface (positive into ocean). **Calling Subroutine:** subroutine thermfj(m,n, j) **Data Declaration:** integer m, n, j **I/O:** stdout **Uses:** mod_xc

6.19 Tides Subroutines (module mod_tides.f)

Subroutine	Description
tidal_arguments	Calculates tidal arguments. It is a kernel routine for subroutine *hat53*. **Calling Sequence:** subroutine tidal_arguments(time1, arg, f, u) **Data Declaration:** real time1, arg, f, u **I/O:** None **Uses:** None
tides_astrol	Computes the basic astronomical mean longitudes s, h, p, and N. Note that N is not 'N', i.e., N is decreasing with time. These formulae are for the period 1990 - 2010, and were derived by David Cartwright (personal comm., Nov. 1990). Time is UTC in decimal MJD. All longitudes returned in degrees (R. D. Ray, Dec. 1990). **Calling Sequence:** subroutine tides_astrol(time, shpn) **Data Declaration:** real time, shpn **I/O:** None **Uses:** None
tides_driver	**Calling Sequence:** subroutine tides_driver(z1rall,z1iall,dtime, astroflag,zpredall,start,ijtdm,ndat) **Data Declaration:** integer ijtdm, start, ndat logical astroflag real z1rall, z1iall, zpredall, dtime **Common Blocks:** common/tidef/ **I/O:** None **Uses:** mod_xc
tides_force	This subroutine calculates body tide. **Calling Sequence:** subroutine tides_force(ll) **Data Declaration:** integer ll **I/O:** None **Uses:** mod_xc
tides_mkw	**Calling Sequence:** subroutine tides_mkw(interp,ind,nc,wr) **Data Declaration:** integer nc, ind real wr logical interp **I/O:** None **Uses:** None
tides_nodal	**Calling Sequence:** subroutine tides_nodal **Data Declaration:** None **Common Bocks:** common /tidef/

Subroutine	Description
	I/O: None **Uses:** None
tides_set	Body force tide setup subroutine. The origin (*time_mjd*) is in modified Julian days, i.e. with zero on Nov 17 0:00, 1858. This is updated once per year (Jan 1), with Jan 1 = 1 (day one). The variable *time_ref* is the time from HYCOM 2.2 origin, i.e. from Jan 1, 1901 0:00 to Jan 1 0:00 in the computation year. **Calling Sequence:** subroutine tides_set(flag) **Data Declaration:** integer flag **I/O:** stdout **Uses:** None

6.20 Tracer Specific Subroutines

Subroutine	Description
initrc	Initializes all tracers. **Calling Sequence:** subroutine initrc(mnth) **Data Declaration:** integer month **I/O:** stdout, ptxt, cformat open, close unit uoff+99 **Uses:** mod_xc, mod_pipe
pcmtrc	Remaps from one set of vertical cells to another. This is accomplished through piecewise constant across each input cell. The output is the average of the interpolation profile across each output cell. **Calling Sequence:** subroutine pcmtrc(si,pi,ki,ks, so,po,ko) **Data Declaration:** integer ki, ks, ko real si ,pi, so, po **I/O:** stdout, write unit 6 **Uses:** None
plmtrcs	Generates slopes for monotonic piecewise linear distribution. **Calling Sequence:** subroutine plmtrcs(rl,rc,rr,a,s,n) **Data Declaration:** integer n real rl, rc, rr, a, s **I/O:** None **Uses:** None
plmtrcx	Generates a monotonic PLM interpolation of a layered field. **Calling Sequence:** subroutine plmtrcx(pt, s,ss,ki,ks) **Data Declaration:** integer ki,ks real pt, s, ss **I/O:** None

Subroutine	Description
	Uses: None
trcupd	Tracer specific operations (side-wall relaxation in *thermf*) **Calling Sequence:** subroutine trcupd(m,n) **Data Declaration:** integer m, n **I/O:** None **Uses:** mod_xc
trcupd_903	Tracer-specific operations for Franks NPZ biology. **Calling Subroutine:** subroutine trcupd_903(m,n, ibio) **Data Declaration:** integer m, n, ibio **I/O:** stdout **Uses:** mod_xc
trcupd_904	Tracer-specific operations for Lima/Idrisi NPZD biology. **Calling Sequence:** subroutine trcupd_904(m,n, ibio) **Data Declaration:** integer m,n,ibio **I/O:** stdout **Uses:** mod_xc

6.21 Turbulence Function Subroutines (inigiss.f)

Subroutine	Description
interp2d_expabs	Subroutine for a modular interpolation calculation. It provides a faster interpolation calculation in the *ifexpabstable*=1 case. **Calling Sequence:** subroutine interp2d_expabs(ri,rid,slq2,sm,sh,ss, m,m0, delta,rat) **Data Declaration:** real ri,rid,slq2,sm,sh,ss,delta,rat integer m,m0 **I/O:** stdout **Uses:** mod_xc
oursal2_1a	Calculates turbulence functions $(Sl/q)^2$ and S_M, S_H, S_S of Ri(=Ri_T+Ri_C) and Ri_d(=Ri_T-Ri_C) in our NCAR turbulence module. Generates contour plots and data for turbulence function plots. **Calling Sequence:** subroutine oursal2_1a(ri,rid,slq2,sm,sh,sc,c_y0, c_y00, iri, irid) **Common Blocks:** common/bb/ **Data Declaration:** real ri,rid,slq2, sm, sh, sc, c_y0, c_y00 integer iri, irid, iend, ier **I/O:** stdout open, write unit uoff+98 **Uses:** mod_xc

Subroutine	Description
rtwi	This subroutine solves general nonlinear equations of the form x=fct_sal(x) by means of Wegsteins iteration method. **Calling Sequence:** subroutine rtwi(xx,val,xst,eps,sm,sh,sc,iend,ier) **Data Declaration:** real xx,val,xst,eps,sm,sh,sc integer iend,ier **Common Blocks:** common/bb/ **I/O:** None **Uses:** mod_xc
smshsc_a3	This routine calculates the “p’s” from the timescale ratios based on *smshsc2*. **Calling Sequence:** subroutine smshsc_a3(yyy,nnn,ccc,sm,sh,sc) **Data Declaration:** real yyy,nnn,ccc,sm,sh,sc **I/O:** stdout **Uses:** mod_xc

7.0 HYCOM 2.2 COMMON BLOCKS

Variable	Type	Description
Common/ bb		
rit	Real	The temperature portion of *ri.*
ric	Real	The concentration portion of *ri.*
Common/ consts		
tenm	Real	Pressure thickness values corresponding to 10m, 1m,...
onem	Real	Pressure thickness values corresponding to 10m, 1m,...
qonem	Real	1/*onem.*
g	Real	Gravity acceleration.
thref	Real	Reference value of specific volume (m^3/kg).
qthref	Real	1/*thref.*
spcifh	Real	Specific heat of sea water (J/kg/deg).
epsil	Real	Small nonzero number used to prevent division by zero.
huge	Real	Large number used to indicate land points.
Common/ czgetc		
iline	Integer	
ibuf	Integer	
Common/ czioxr		
htmp	Real	
Common/ czioxw		
w	Real	
wminy	Real	
wmaxy	Real	
Common/ czioxx		
iarec	Integer	
Common/forcing		Atmospheric forcing fields.
taux	Real	Wind stress in *x* direction.
tauy	Real	Wind stress in *y* direction.
wndspd	Real	Wind speed.
airtmp	Real	Air temperature.
vapmix	Real	Atmospheric vapor mixing ratio.
precip	Real	Precipitation.
radflx	Real	Net solar radiation.
swflx	Real	Net shortwave radiation.

Variable	Type	Description
surtmp	Real	Surface temp. used to calculate input lw radiation
seatmp	Real	Best available SST from observations
akpar	Real	Photosynthetically available radiation coeff.
rivers	Real	River inflow bogused to surface precipitation.
offlux	Real	Net heat flux offset.
betard	Real	Red extinction coefficient.
betabl	Real	Blue extinction coefficient.
redfac	Real	Fract. of penetr. red light.
Common/gissi1		NASA GISS variables.
nextrtb10	Integer	
ifexpabstable	Integer	
nextrtbl1	Integer	
nextrtbl	Integer	
nposapprox	Integer	
mt0	Integer	
mt	Integer	
ntbl	Integer	
mt_ra_r	Integer	
n_theta_r_oct	Integer	
nbig	Integer	
Common/gissi2		
irimax	Integer	
nb	Integer	
Common/gissi3		
ifback	Integer	
ifsali	Integer	
ifepson2	Integer	
ifrafgmax	Integer	
ifsalback	Integer	
ifchengcon	Integer	
ifunreal	Integer	
idefmld	Integer	
ifpolartablewrite	Integer	
ifbg_theta_interp	Integer	
Common/gissr1		
deltheta_r	Real	
pidbl	Real	

Variable	Type	Description
rri	Real	
Common/gissr2		
ribtbl(-762:762)	Real	
ridb(-762:762)	Real	
slq2b(-762:762, lgiss:nlgiss)	Real	
dri	Real	
smb(-762:762, nlgiss:nlgiss)	Real	
shb(-762:762,-lgiss:nlgiss)	Real	
ssb(-762:762,-nlgiss:nlgiss)	Real	
back_ra_r(-39:117)	Real	
sisamax(-39:117)	Real	
ra_rmax(-39:117)	Real	
c_y_r0(-39:117)	Real	
sm_r1(-39:117)	Real	
sh_r1(-39:117)	Real	
ss_r1(-39:117)	Real	
slq2_r1(-39:117)	Real	
b1,visc_cbu_limit,diff_cbt_limit	Real	
theta_rcrp,theta_rcrn	Real	
Common/gissr3		For bottom enhanced and latitude dependent mixing.
back_ph_0	Real	
adjust_gargett	Real	
back_k_0	Real	
back_del_0	Real	
back_s2	Real	
ri0	Real	
ebase	Real	
epson2_ref	Real	
eps_bot0	Real	
scale_bot	Real	
eplatidepmin	Real	
wave_30	Real	
deltemld, delrhmld	Real	
back_sm2	Real	

Variable	Type	Description
v_back0	Real	
t_back0	Real	
s_back0	Real	
ri_internal	Real	
backfrac	Real	
backfact	Real	
ako	Real	
tpvot0	Real	
sgmt	Real	
tptot0	Real	
tpcot0	Real	
ttot0	Real	
tcot0	Real	
tctot0	Real	
tpvot	Real	
tptot	Real	
tpcot	Real	
ttot	Real	
tcot	Real	
tctot	Real	
back_1_0	Real	
Common/ halobp		
ipb	Integer	
Common/hycom1c		
ctitle	Character	Four lines describing the simulation.
Common/hycom1i		
kapi	Integer	Thermobaric reference state index (1 or 3).
Common/hycom1r		
u,v	Real	Velocity components.
dp, dpu, dpv	Real	Layer thickness.
p, pu, pv	Real	Interface pressure.
temp	Real	Temperature.
saln	Real	Salinity.
th3d	Real	Potential density.
thstar	Real	Virtual potential density.
montg	Real	Montgomery potential.
tracer	Real	Inert tracers.
dpold, dpoldm	Real	Layer thickness.

Variable	Type	Description
theta	Real	Isopycnal layer target densities- *thbase.*
diaflx	Real	Time integral of diapycnal flux.
corio	Real	Coriolis parameter.
potvor	Real	Potential vorticity.
psikk	Real	Montgomery potential in bottom layer.
thkk	Real	Virtual potential density in bottom layer.
Common/hycom2r		
montg	Real	Montgomery potential.
uflx,vflx,	Real	Mass fluxes.
uflxav,vflxav	Real	Average fluxes.
dpav	Real	Average fluxes.
ubavg,vbavg	Real	Barotropic velocity.
pbavg	Real	Barotropic pressure.
defor1,defor2	Real	Deformation components.
ubrhs, vbrhs	Real	Rhs of barotropic *u,v* equations.
utotm, vtotm	Real	Total (barotrop.+baroclin.).
utotn, vtotn	Real	Velocities at two time levels.
uflux, vflux	Real	Horizontal mass fluxes.
uflux1, vflux1	Real	More mass fluxes.
uflux2, vflux2	Real	More mass fluxes.
uflux3, vflux3	Real	More mass fluxes.
Common/hycom3r		
util1, util2	Real	Arrays for temporary storage.
util3, util4	Real	Arrays for temporary storage.
util5, util6	Real	Arrays for temporary storage.
plon, plat	Real	Lon,lat at *p* pts.
ulon, ulat	Real	Lon,lat at *u* pts.
vlon, vlat	Real	Lon,lat at *v* pts.
scux, scuy	Real	Mesh size at *u* pts in *x,y* dir.
scvx, scvy	Real	Mesh size at *v* pts in *x,y* dir.
scpx, scpy	Real	Mesh size at *p* pts in *x,y* dir.
scqx, scqy	Real	Mesh size at *q* pts in *x,y* dir.
scu2, scv2	Real	Grid box area at *u,v* pts.
scp2, scq2	Real	Grid box area at *p,q* pts.
scp2i,scq2i	Real	Inverses of *scp2*,*scq2*.
scuxi,scvyi	Real	Inverses of *scux*, *scvy*.
aspux,aspuy	Real	U-grid aspect ratios for diffusion.
aspvx,aspvy	Real	V-grid aspect ratios for diffusion.

Variable	Type	Description
veldf2u	Real	U-grid Laplacian diffusion coefficient.
veldf2v	Real	V-grid Laplacian diffusion coefficient.
veldf4u	Real	U-grid biharmonic diffusion coefficient.
veldf4v	Real	V-grid biharmonic diffusion coefficient.
thkdf4u	Real	U-grid biharmonic diffusion coefficient.
thkdf4v	Real	V-grid biharmonic diffusion coefficient.
pgfx, pgfy	Real	Horizontal pressure gradient.
gradx,grady	Real	Horizontal pressure gradient.
depthu,depthv	Real	Bottom pressure at *u,v* points.
pvtrop	Real	Potential vorticity of barotropic flow.
depths	Real	Water depth.
drag	Real	Bottom drag.
topiso	Real	Shallowest depth for isopycnal layers (pressure units).
Common/hycom4i		
klist	Integer	K-index.
jerlov	Integer	Jerlov water type 1-5.
Common/hycom4r		
uja, ujb	Real	Velocities at lateral.
via, vib	Real	Neighbor points.
pbot	Real	Bottom pressure at t=0.
sgain	Real	Salinity changes from diapycnic mixing.
surtx	Real	Surface net x-stress on *p*-grid.
surty	Real	Surface net y-stress on *p*-grid.
surflx	Real	Surface net thermal energy flux.
sswflx	Real	Surface swv thermal energy flux.
mixflx	Real	Mixed layer thermal energy flux.
sstflx	Real	Surface thermal flux from SST relaxation.
sssflx	Real	Surface salinity flux from SSS relaxation.
salflx	Real	Surface salinity flux.
buoflx	Real	Mixed layer buoyancy flux.
bhtflx	Real	Mixed layer buoyancy flux from heat.
turgen	Real	Turbulent kinetic energy generation.
thkice	Real	Grid-cell avg. ice thickness (m).
covice	Real	Ice coverage (rel. units).
temice	Real	Ice surface temperature.
sic	Real	Ice concentration on *p*-grid from coupler.
sih	Real	Ice thickness on *p*-grid from coupler.

Variable	Type	Description
uice	Real	Ice *u*-velocity on *p*-grid from coupler.
vice	Real	Ice *v*-velocity on *p*-grid from coupler.
txice	Real	X-stress under ice on *p*-grid from coupler.
tyice	Real	Y-stress under ice on *p*-grid from coupler.
flxice	Real	Heat flux under ice.
fswice	Real	SWV flux under ice.
sflice	Real	Salt flux under ice.
Common/hycom5i		
nmlb	Integer	Layer containing MLB.
Common/hycom5r		
dpmixl	Real	Mixed layer depth.
t1sav	Real	Upper sublayer temperature.
s1sav	Real	Upper sublayer salinity.
tmlb	Real	Temperature in l yr. containing MLB.
smlb	Real	Salinity in l yr. containing MLB.
hekman	Real	Ekman layer thickness.
hmonob	Real	Monin-Obukhov length.
dpbl	Real	Turbulent boundary layer depth.
dpbbl	Real	Bottom turbulent boundary layer depth.
dpmold	Real	Mixed layer depth at old time step.
tmix	Real	Mixed layer temperature.
smix	Real	Mixed layer salinity.
thmix	Real	Mixed layer potential density.
umix, vmix	Real	Mixed layer velocity.
dp0k	Real	Minimum deep *z*-layer separation.
ds0k	Real	Minimum shallow *z*-layer separation.
dssk	Real	Sigma depth scale factor.
dp0kp	Real	Dp0k + 1m.
Common/ iovars		
flnmdep	Character	
flnmarc	Character	
flnmgrd	Character	
flnmrsi	Character	
flnmrso	Character	
flnmflx	Character	
flnmovr	Character	
flnmfor	Character	
flnmforw	Character	

Variable	Type	Description
Common/kppi		
niter	Integer	KPP: iterations for semi-implicit solution (2 recommended).
hblflg	Integer	KPP: boundary layer interpolation flag (0=con.1=lin., 2=quad.).
Common/ kppltr		
wmt	Real	
wst	Real	
Common/kppr		KPP variables.
zgrid	Real	Grid levels in meters.
vcty	Real	Vertical viscosity coefficient.
difs	Real	Vertical scalar diffusivity.
dift	Real	Vertical temperature diffusivity.
ghats	Real	Nonlocal transport.
vonk	Real	Von Karman constant.
zmin,zmax	Real	Zehat limits for table.
epsilon	Real	Vertical coordinate scale factor.
cmonob	Real	KPP: scale factor for Monin-Obukov length.
cekman	Real	KPP: scale factor for Ekman depth.
qrinfy	Real	KPP: 1/max grad. Richardson No. for shear instability.
difm0	Real	KPP: max viscosity due to shear instability.
difs0	Real	KPP: max diffusivity due to shear instability.
difmiw	Real	KPP: background/internal wave viscosity (m^2/s).
difsiw	Real	KPP: background/internal wave diffusivity (m^2/s).
qdif0	Real	KPP: difm0 /difs0.
qdifiw	Real	KPP: difmiw/difsiw.
dsfmax	Real	KPP: salt fingering diffusivity factor (m^2/s).
rrho0	Real	KPP: salt fingering rp=(*alpha*delt*)/(*beta*dels*).
ricr	Real	KPP: critical bulk Richardson number.
cs	Real	KPP: value for nonlocal flux term.
cstar	Real	KPP: value for nonlocal flux term.
cv	Real	KPP: buoyancy frequency ratio (0.0 to use a fn. of N).
c11	Real	KPP: value for turbulence velocity scale.
deltaz	Real	Delta *zehat* in table.

Variable	Type	Description
deltau	Real	Delta *ustar* in table.
vtc	Real	Constant for estimating background shear in rib calculation.
cg	Real	Constant for estimating nonlocal flux term of the differential equation.
dp0enh	Real	Dist. for tapering diff. enhancement at interface nbl-1.
Common/ linepr		
lp	Integer	
Common/ momtumr4		
stress	Real	
stresx	Real	
stresy	Real	
dpmx	Real	
thkbop	Real	
vis2u	Real	
vis4u	Real	
vis2v	Real	
vis4v	Real	
vort	Real	
oneta	Real	
wgtia	Real	
wgtib	Real	
wgtja	Real	
wgtjb	Real	
dl2u	Real	
dl2uja	Real	
dl2ujb	Real	
dl2v	Real	
dl2via	Real	
dl2vib	Real	
Common/ mxgissij_b		
jtheta_r0	Integer	
jtheta_r1	Integer	
itheta_r0	Integer	
itheta_r1	Integer	
Common/myr		Mellor-Yamada 2.5 variables.
q2	Real	TKE.

Variable	Type	Description
q2l	Real	TKE * turbulent length scale.
difqmy	Real	TKE diffusivity.
vctymy	Real	Viscosity on Mellor-Yamada vertical grid.
diftmy	Real	Temperature diffusivity on Mellor-Yamada vertical grid.
ghc	Real	Constant for calculating TKE production.
sef	Real	Constant for calculating TKE production.
smll	Real	Constant for calculating TKE.
const1	Real	Coefficient for estimating surface and bottom boundary conditions.
coef4	Real	Coefficient for calculating viscosity/diffusivity.
coef5	Real	Coefficient for calculating viscosity/diffusivity.
a1my	Real	Coefficient for calculating viscosity/diffusivity.
b1my	Real	Coefficient for calculating viscosity/diffusivity.
a2my	Real	Coefficient for calculating viscosity/diffusivity.
b2my	Real	Coefficient for calculating viscosity/diffusivity.
c1my	Real	Coefficient for calculating viscosity/diffusivity.
e1my	Real	Coefficient for calculating viscosity/diffusivity.
e2my	Real	Coefficient for calculating viscosity/diffusivity.
e3my	Real	Coefficient for calculating viscosity/diffusivity.
Common/parms1i		Integer parameters.
tsofrq	Integer	Number of time steps between anti-drift offset calculations.
mixfrq	Integer	KT: number of time steps between diapycnal mixing calculations.
icefreq	Integer	Number of time steps between sea-ice updates.
nhybrd	Integer	Number of hybrid levels (0=all isopycnal).
nsigma	Integer	Number of sigma levels (nhybrd-nsigma *z*-levels).
hybflg	Integer	Hybrid generator flag (0=T&S).
advflg	Integer	Thermal advection flag (0=T&S).
advtyp	Integer	Scalar advection type (0=PCM).
kapref	Integer	Thermobaric reference state (1=input, 0=none, 1,2,3=constant).
kapnum	Integer	Number of thermobaric reference states (1 or 2).
ntracr	Integer	Number of tracers (<=*mxtrcr*).
trcflg	Integer	Tracer type flag (one per tracer).
clmflg	Integer	Climatology frequency flag. (6=bimonthly;12=monthly).

Variable	Type	Description
dypflg	Integer	KT: diapycnal mixing flag (0=none;1=KPP; 2=explicit).
iniflg	Integer	Initial state flag (0=level;1=zonal; 2=climatology).
lbflag	Integer	Lateral barotropic boundary flag (0=none; 1=port; 2=nest; 3=Flather).
mapflg	Integer	Map flag (0=Mercator; 2=uniform; 3=beta-plane; 4=input).
yrflag	Integer	Days in year flag (0=360; 1=366).
iversn	Integer	HYCOM 2.2 version number x10.
iexpt	Integer	Experiment number x10.
jerlv0	Integer	Initial Jerlov water type (1 to 5; 0 to use *kpar*).
iceflg	Integer	Sea ice model flag (0=none; 1=energy loan; 2=coupled/ESMF).
icmflg	Integer	Ice mask flag (0=none; 1=clim; 2=atmos; 3=obs).
wndflg	Integer	Wind str. input flag (0=none; 1=on u/v grid; 2,3=on *p* grid).
flxflg	Integer	Thermal forcing flag (0=none; 3=net-flux; 1,2,4=SST-based).
empflg	Integer	E-P forcing flag (0=none; 3=net_E-P; 1,2,4=SST-based_E).
lwflag	Integer	Longwave corr. flag (0=none; 1=clim; 2=atmos), SST-based.
sstflg	Integer	SST relaxation flag (0=none; 1=clim; 2=atmos; 3=obs).
sssflg	Integer	SSS relaxation flag (0=none; 1=clim).
Common/parms1r		Real parameters.
sigma	Real	Isopyncnal layer target densities (sigma units).
thbase	Real	Reference density (sigma units).
saln0	Real	Initial salinity value.
baclin	Real	Baroclinic time step.
batrop	Real	Barotropic time step.
qhybrlx	Real	HYBGEN: relaxation coefficient (inverse baroclinic time steps).
visco2	Real	Deformation-dependent Laplacian viscosity factor.
visco4	Real	Deformation-dependent biharmonic viscosity factor.
veldf2	Real	Diffusion velocity (m/s) for Laplacian

Variable	Type	Description
		momentum dissipation.
veldf4	Real	Diffusion velocity (m/s) for biharmonic momentum dissipation.
temdf2	Real	Diffusion velocity (m/s) for Laplacian temp/salinity diffusion.
temdfc	Real	Temp diffusion conservation (0.0 all density; 1.0 all temp).
thkdf2	Real	Diffusion velocity (m/s) for Laplacian thickness diffusion.
thkdf4	Real	Diffusion velocity (m/s) for biharmonic thickness diffusion.
vertmx	Real	Diffusion velocity (m/s) for momentum mixing across MLB.
diapyc	Real	Temperature anti-drift offset (°C/century).
tofset	Real	Salinity anti-drift offset (psu/century).
sofset	Real	Diffusion velocity (m/s) for biharmonic thickness diffusion.
dtrate	Real	KT: maximum permitted mixed layer detrainment rate (m/day).
h1	Real	Depth interval used in lateral weighting of horiz. pres. grad.
slip	Real	= +1 for free-slip; = -1 for non-slip boundary conditions.
cb	Real	Coefficient of quadratic bottom friction.
cbar	Real	Rms flow speed (m/s) for linear bottom friction law.
dsurfq	Real	Number of days between model diagnostics at the surface.
diagfq	Real	Number of days between model diagnostics.
meanfq	Real	Number of days between model diagnostics (time averaged).
rstrfq	Real	Number of days between model restart output.
bnstfq	Real	Number of days between baro. nesting archive input.
nestfq	Real	Number of days between 3D nesting archive input.
wuv1	Real	Weights for time smoothing of *u,v* field.
wuv2	Real	Weights for time smoothing of *u,v* field.
wts1	Real	Weights for time smoothing of *t,s* field.
wts2	Real	Weights for time smoothing of *t,s* field.

Variable	Type	Description
wbaro	Real	Weight for time smoothing of barotropic *u,v,p* field.
thkmls	Real	Reference mixed -layer thickness for SSS relaxation (m).
thkmlt	Real	Reference mixed-layer thickness for SST relaxation (m).
thkriv	Real	Nominal thickness of river inflow (m).
thkmin	Real	KT/PWP: minimum mixed-layer thickness (m).
bldmin	Real	KPP: minimum surface boundary layer thickness (m).
bldmax	Real	KPP: maximum surface boundary layer thickness (m).
thkbot	Real	Thickness of bottom boundary layer (m).
thkfrz	Real	Maximum thickness of near surface freezing zone (m).
tfrz_0	Real	ENLN: ice melting point (°C) at S=0 psu.
tfrz_s	Real	ENLN: gradient of ice melting point (°C /psu).
ticegr	Real	ENLN: vertical temperature gradient inside ice (deg/m). (0.0 to get ice surface temperature from atmos. surface temp.)
hicemn	Real	ENLN: minimum ice thickness (m).
hicemx	Real	ENLN: maximum ice thickness (m).
dp00	Real	Deep *z*-level spacing minimum thickness (m).
dp00f	Real	Deep *z*-level spacing stretching factor (1.0=const. *z*).
dp00x	Real	Deep *z*-level spacing maximum thickness (m).
ds00	Real	Shallow *z*-level spacing minimum thickness (m).
ds00f	Real	Shallow *z*-level spacing stretching factor (1.0=const. *z*).
ds00x	Real	Shallow *z*-level spacing maximum thickness (m).
dp00i	Real	Deep isopycnal spacing minimum thickness (m).
isotop	Real	Shallowest depth for isopycnal layers (m), <0 from file.
sigjmp	Real	Minimum density jump across interfaces (theta units).
tmljmp	Real	Equivalent temperature jump across the mixed layer (°C).
salmin	Real	Minimum salinity allowed in an isopycnic layer.
Common/pwpr		PWP variables.

Variable	Type	Description
rigc	Real	PWP: critical gradient Richardson number.
ribc	Real	PWP: critical bulk Richardson number.
Common/ rdforfi		
mreca	Integer	
mrecc	Integer	
mreck	Integer	
mrecr	Integer	
Common/swtchs		If set to .true., then...
diagno	Logical	Output model fields and diagnostic messages.
thermo	Logical	Use thermodynamic forcing (*flxflg*>0).
windf	Logical	Use wind stress forcing (*wndflg*>0).
pcipf	Logical	Use evap-precip surface salinity flux.
epmass	Logical	Treat evap-precip as a mass exchange
priver	Logical	Use river precip bogus.
rivera	Logical	Annual-only river precip bogus.
kparan	Logical	Annual-only *kpar*.
relax	Logical	Activate lateral boundary T/S/p climatological nudging.
srelax	Logical	Activate surface salinity climatological nudging (*sssflg*==1).
trelax	Logical	Activate surface temperature climatological nudging (*sstflg*==1).
trcrlx	Logical	Activate lateral boundary tracer climatological nudging.
relaxf	Logical	Input T/S/p relaxation fields.
relaxs	Logical	Input surface relaxation fields only.
relaxt	Logical	Input tracer relaxation fields.
locsig	Logical	Use locally-referenced potential density for stability.
vsigma	Logical	Use spatially varying target densities.
hybrid	Logical	Use hybrid vertical coordinates.
isopyc	Logical	Use isopycnic vertical coordinates.
icegln	Logical	Use energy loan ice model (*iceflg*==1).
mxlkta	Logical	KT: activate original mixed layer model (*mlflag*==2).
mxlktb	Logical	KT: activate alternative mixed layer model (*mlflag*==3).
mxlkrt	Logical	KT: activate HYCOM 2.2 Kraus-Turner (*mlflag*==2,3).

Variable	Type	Description
pensol	Logical	KT: activate penetrating solar radiation.
mxlkpp	Logical	KPP: activate mixed layer model (*mlflag*==1).
bblkpp	Logical	KPP: activate bottom boundary layer.
shinst	Logical	KPP: activate shear instability mixing.
dbdiff	Logical	KPP: activate double diffusion mixing.
nonloc	Logical	KPP: activate nonlocal boundary layer mixing.
latdiw	Logical	KPROF: activate lateral dependent internal wave mixing.
botdiw	Logical	GISS: activate bottom enhancing internal wave mixing.
difsmo	Logical	KPROF: activate horiz. smooth diff. coeffs.
difout	Logical	KPROF: output visc/diff coeffs in archive.
mxlmy	Logical	MY2.5: activate mixed layer model (*mlflag*==5).
mxlpwp	Logical	PWP: activate mixed layer model (*mlflag*==4).
mxlgis	Logical	GISS: activate mixed layer model (*mlflag*==6).
flxoff	Logical	Add a net heat flux offset.
flxsmo	Logical	Activate smoothing of surface fluxes.
trcrin	Logical	Initialize tracer from restart file.
trcout	Logical	Advect tracer and save results in history/restart file.
dsur1p	Logical	Single point only surface diagnostics.
Common/ testpt		Grid point where detailed diagnostics are desired.
itest	Integer	
jtest	Integer	
ittest	Integer	
jttest	Integer	
Common/ tidef		
amp	Real	
phase	Real	
pu8	Real	
pf8	Real	
arg8	Real	
timeref	Real	
time_mjd	Real	
Common/varblsd		
area	Real	Basing area (m^2).
avgbot	Real	Mean basin depth (m).
watcum	Real	Cumulative heat flux.

Variable	Type	Description
empcum	Real	Cumulative salt flux.
Common/varblsi		
nstep	Integer	Model time step number.
nstep1	Integer	1st time step of this integration.
nstep2	Integer	Last time step of this integration.
1step	Integer	No. of barotropic timesteps/baroclinic time steps.
l0, l1, l2, l3	Integer	Wind indices.
lk0,lk1,lk2,lk3	Integer	*Kpar* indices.
lr0,lr1,lr2,lr3	Integer	River indices.
lc0,lc1,lc2,lc3	Integer	Clim. indices.
ln0,ln1	Integer	Nest indices.
lb0,lb1	Integer	Baro. indices.
Common/varblsr		
time	Real	Model time (days).
delt1	Real	Timestep (seconds).
dlt	Real	ΔT.
w0, w1, w2, w3	Real	Wind interp. scale factors.
wk0, wk1, wk2, wk3	Real	Kpar interp. scale factors.
wr0, wr1, wr2, wr3	Real	River interp. scale factors.
wc0, wc1, wc2, wc3	Real	Climatology interpolation scale factors.
wn0, wn1	Real	Nest interpolation scale factors.
wb0, wb1	Real	Baro. interpolation scale factors.
Common/wall1i		
maskbc	Integer	Mask for nested barotropic boundary condition.
Common/wallr		Surface and sidewall and nestwall boundary fields.
pwall	Real	Pressure boundary condition (b.c.) at sidewalls.
swall	Real	Salinity b.c. at sidewalls.
twall	Real	Temperature b.c. at sidewalls.
trwall	Real	Tracer b.c. at sidewalls.
pnest	Real	Pressure b.c. at nestwalls.
snest	Real	Salinity b.c. at nestwalls.
tnest	Real	Temperature b.c. at nestwalls.
unest	Real	U-velocity. b.c. at nestwalls.
vnest	Real	V-velocity b.c. at nestwalls.
ubnest	Real	Barotropic *u*-velocity at nestwalls.
vbnest	Real	Barotropic *v*-velocity at nestwalls.

Variable	Type	Description
ubpnst	Real	Barotropic *u*-velocity at nestwalls on *p*-grid.
vbpnst	Real	Barotropic *v*-velocity at nestwalls on *p*-grid.
pbnest	Real	Barotropic pressure at nestwalls.
rmu	Real	Weights for s.w.b.c. relaxation.
rmunp	Real	Weights for p.n.w.b.c. relaxation.
rmunv	Real	Weights for v.n.w.b.c. relaxation.
rmutra	Real	Weights for tracer b.c. relaxation (maximum of all tracers)
rmutr	Real	Weights for tracer b.c. relaxation.
Common/ xcagetr		
al	Real	
at	Real	
Common/ xclgetr		
al	Real	
Common/ xcmaxr4		
b	Real	
c	Real	
Common/ xcmpii		
mpierr	Integer	
mpireq	Integer	
mpistat	Integer	
Common/ xcsum8		
sum8t	Real	
sum8j	Real	
sum8s	Real	
Common/ xctilr4		
ai	Real	
aj	Real	
Common/ xctilra		
aia	Real	

8.0 MODEL INPUT PARAMETERS (BLKDAT.INPUT)

Parameter	Description
iversn	HYCOM version number x10.
iexpt	Experiment number x10.
idm	Longitudinal array size.
jdm	Latitudinal array size.
itest	Grid point where detailed diagnostics are desired.
jtest	Grid point where detailed diagnostics are desired.
kdm	Number of layers.
nhybrd	Number of hybrid levels (0=all isopycnal).
nsigma	Number of sigma levels (*nhybrid-nsigma z*-levels).
dp00	Deep *z*-level spacing minimum thickness (m).
dp00x	Deep *z*-level spacing maximum thickness (m).
dp00f	Deep *z*-level spacing stretching factor (1.0=const. space).
ds00	Shallow *z*-level spacing minimum thickness (m).
ds00x	Shallow *z*-level spacing maximum thickness (m).
ds00f	Shallow *z*-level spacing stretching factor (1.0=const. space).
dp00i	Deep isopycnal spacing minimum thickness (m).
isotop	Shallowest depth for isopycnal layers (m), < 0 from file.
saln0	Initial salinity value (psu), only used for *iniflg* < 2.
locsig	Locally referenced potential density for stability (0=F; 1=T).
kapref	Thermobaric ref. state (-1= input; 0= none; 1,2,3= constant).
thflag	Reference pressure flag (0=Sigma-0; 2=Sigma-2; 4=Sigma-4).
thbase	Reference density (sigma units).
vsigma	Spatially varying isopycnal target densities (0=F,1=T).
sigma	Layers 1-32 isopycnal target density (sigma units).
iniflg	Initial state flag (0=levl; 1=zonl; 2=clim; 3=archv).
jerlv0	Initial Jerlov water type (1 to 5).
yrflag	Days in year flag (0=360; 1=366; 2=366J1; 3=actual).
dsurfq	Number of days between model diagnostics at the surface.
diagfq	Number of days between model diagnostics.
meanfq	Number of days between model diagnostics (time averaged).
rstrfq	Number of days between model restart output.
bnstfq	Number of days between barotropic nesting archive input.
nestfq	Number of days between 3D nesting archive.
cplifq	Number of days (or time steps) between sea ice coupling.
baclin	Baroclinic time step (seconds), integer divisor of 86400.
batrop	Barotropic time step (seconds), integer divisor of *baclin*/2.

Parameter	Description
incflg	Incremental update flag (0=no; 1 = yes; 2 = full-velocity).
incstp	Number of timesteps for full update (1 = direct insertion).
incupf	Number of days of incremental updating input.
hybrlx	HYBGEN: inverse relaxation coefficient (time steps).
hybflg	Hybrid generator flag (0=T&S; 1=TH&S; 2=TH&T).
advflg	Thermal advection flag (0=T&S; 1=TH&S; 2=TH&T).
advtyp	Scalar advection type (0 = PCM; 1 = MPDATA; 2 =FCT2; 4 = FCT4).
slip	= +1 for free-slip, = -1 for non-slip boundary conditions.
visco2	Deformation-dependent Laplacian viscosity factor.
visco4	Deformation-dependent biharmonic viscosity factor.
veldf2	Diffusion velocity (m/s) for Laplacian momentum dissipation.
veldf4	Diffusion velocity (m/s) for biharmonic momentum dissipation.
thkdf2	Diffusion velocity (m/s) for Laplacian thickness diffusion.
thkdf4	Diffusion velocity (m/s) for biharmonic thickness diffusion.
temdf2	Diffusion velocity (m/s) for Laplacian temp/salinity diffusion.
temdfc	Temp diffusion conservation (0.0,1.0- all densities, temp, respectively.)
vertmx	Diffusion velocity (m/s) for momentum at MICOM mixed layer base.
cbar	RMS flow speed (m/s) for linear bottom friction.
cb	Coefficient of quadratic bottom friction.
drglim	Limiter for explicit friction (1.0 none; 0.0 implicit).
drgscl	Scale factor for tidal drag (0.0 for no tidal drag).
thkbot	Thickness of bottom boundary layer (m).
sigjmp	Minimum density jump across interfaces (kg/m^3).
tmljmp	Equivalent temperature jump across mixed-layer (ºC).
thkmls	Reference mixed-layer thickness for SSS relaxation (m).
thkmlt	Reference mixed-layer thickness for SST relaxation (m).
thkriv	Nominal thickness of river inflow (m).
thkfrz	Maximum thickness of near-surface freezing zone (m).
iceflg	Ice model flag (0=none; 1=energy loan; 2=coupled/ESMF).
tfrz_0	ENLN: ice melting point (ºC) at S=0psu.
tfrz_s	ENLN: gradient of ice melting point (ºC /psu).
ticegr	ENLN: temp. grad. inside ice (deg/m); =0 use *surtmp*.
hicemn	ENLN: minimum ice thickness (m).
hicemx	ENLN: maximum ice thickness (m).
ntracr	Number of tracers (0=none; negative to initialize).
trcflg	Tracer flags (one digit per tracer, most sig. replicated).
tsofrq	Number of time steps between anti-drift offset calculations.
tofset	Temperature anti-drift offset (ºC/century).
sofset	Salinity anti-drift offset (psu/century).
mlflag	Mixed layer flag (0=none; 1=KPP; 2-3=KT; 4=PWP; 5=MY; 6=GISS).

Parameter	Description
pensol	KT: activate penetrating solar radiation (0=F,1=T).
dtrate	KT: maximum permitted mixed layer detrainment rate (m/day).
thkmin	KT/PWP: minimum mixed-layer thickness (m).
dypflg	KT/PWP: diapycnal mixing flag (0=none; 1=KPP; 2=explicit).
mixfrq	KT/PWP: number of time steps between diapycnal mixing calculations.
diapyc	KT/PWP: diapycnal diffusivity x buoyancy frequency (m^2/s^2).
rigr	PWP: critical gradient Richardson number.
ribc	PWP: critical bulk Richardson number.
rinfty	KPP: maximum gradient Richardson number (shear inst.).
ricr	KPP: critical bulk Richardson number.
bldmin	KPP: minimum surface boundary layer thickness (m).
bldmax	KPP: maximum surface boundary layer thickness (m).
cekman	KPP/KT: scale factor for Ekman depth.
cmonob	KPP: scale factor for Monin-Obukov depth.
bblkpp	KPP: activate bottom boundary layer (0=F,1=T).
shinst	KPP: activate shear instability mixing (0=F; 1=T).
dbdiff	KPP: activate double diffusion mixing (0=F; 1=T).
nonloc	KPP: activate nonlocal b. layer mixing (0=F; 1=T).
latdiw	K-PROF: activate lat.dep. internal wave mixing (0=F; 1=T).
botdiw	GISS: activate bottom enhanced internal wave mixing (0=F; 1=T).
difsmo	K-PROF: activate horizontal smooth diff coefficients (0=F; 1=T).
difout	K-PROF: output viscosity/diff coeffs in archive (0=F; 1=T).
difm0	KPP: max viscosity due to shear instability (m^2/s).
difs0	KPP: max diffusivity due to shear instability (m^2/s).
difmiw	KPP: background/internal wave viscosity (m^2/s).
difsiw	KPP: background/internal wave diffusivity (m^2/s).
dsfmax	KPP: salt fingering diffusivity factor (m^2/s).
rrho0	KPP: salt fingering *rp*=(alpha*delT)/(beta*delS).
cs	KPP: value for nonlocal flux term.
cstar	Value for nonlocal flux term.
cv	Buoyancy frequency ratio (0.0 to use a function of N).
c11	KPP: value for turbulence velocity scale.
hblflg	KPP: boundary layer interp. flag (0=const.;1=linear; 2=quad.).
niter	KPP: iterations for semi-implicit solution (2 recommended).
fltflg	FLOATS: synthetic float flag (0=no; 1=yes).
nfladv	FLOATS: advect every *nfladv* baroclinic time steps (even, >=4).
nflsam	FLOATS: output (0=every *nfladv* steps; >0=no. of days).
intpfl	FLOATS: horizontal interpolation (0=2nd order+n.n.; 1=n.n. only).
iturbv	FLOATS: add horizontal turbulence advection velocity (0=no; 1=yes).

Parameter	Description
ismpfl	FLOATS: sample water properties at float (0=no; 1=yes).
tbvar	FLOATS: horizontal turbulence velocity variance scale (m^2/s^2).
tdecri	FLOATS: inverse decorrelation time scale (1/day).
lbflag	Lateral barotropic boundary flag (0=none; 1=port; 2=input).
tidflg	TIDES: tidal forcing flag (0=none; 1=open bdy; 2=bdy&body).
tidcon	TIDES: 1 digit per (Q1K2P1N2O1K1S2M2), 0=off; 1=on.
tidsal	TIDES: scalar self attraction and loading factor.
tidgen	TIDES: generic time (0=F; 1=T).
tidrmp	TIDES: ramp time (days).
tid_t0	TIDES: origin for ramp time (model day).
clmflg	Climatology frequency flag (6=bimonthly; 12=monthly).
wndflg	Wind stress input flag (0=none; 1=u/v-grid; 2,3=p-grid).
ustflg	Ustar forcing flag (3 = input; 1, 2 = wndspd; 4 = stress).
flxflg	Thermal forcing flag (0=none; 3=net-flux; 1,2,4=SST-based).
empflg	E-P forcing flag (0=none; 3=net_E-P; 1,2,4=SST-bas_E).
sssflg	SSS relaxation flag (0=none; 1=clim).
lwflag	Longwave (SST) flag (0=none; 1=clim; 2=atmos).
sstflg	SST relaxation flag (0=none; 1=clim; 2=atmos; 3=observed).
icmflg	Ice mask flag (0=none; 1=clim; 2=atmos; 3=observed).
flxoff	Net flux offset flag (0=F; 1=T).
flxsmo	Smooth surface fluxes (0=T; 1=F).
relax	Activate lateral boundary nudging (0=F;1=T).
trcrlx	Activate lateral boundary tracer nudging (0=F; 1=T).
priver	Rivers as a precipitation bogus (0=F; 1=T).
epmass	Treat evap-precip as a mass exchange (0=F; 1=T).

9.0 NOTES

9.1 Acronyms and Abbreviations

Acronym	Definition
ALE	Arbitrary Lagrangian-Eulerian
CICE	Los Alamos National Laboratory Sea Ice Model
COARE	Coupled Ocean Atmosphere Response Experiment
CODINE	Computing in Distributed Network Environments
CPP	C Preprocessor
cp	Copy
ESMF	Earth System Modeling Framework
EVD	Eulerian Vertical Direction
FCT	Flux-Corrected Transport scheme
GDEM	Generalized Digital Environmental Model
GISS	NASA Goddard Institute for Space Studies
GRD	Global Resource Director
HYCOM	HYbrid Coordinate Ocean Model
I/O	Input/Output
KPP	K-Profile Parameterization model setup
KT	Kraus-Turner model setup
LSF	Load Sharing Facility
MKS	Meters Kilograms Seconds measuring system
MLB	Mixed Layer Base
MPDATA	Multidimensional Positive Definite Advection Transport Algorithm
MPI	Message Passing Interface
NetCDF	Network Common Data Format
NCAR	National Center for Atmospheric Research
NCEP	National Center for Environmental Prediction
NCARGF77/90	NCAR Graphics for Fortran 77/90
NFS	Network File System
NOPP	National Oceanographic Partnership Program
NQS	Network Queuing System
NRL	Naval Research Laboratory
ONR	Office of Naval Research
PBS	Portable Batch System
PE	Processor Element or Potential Energy
PLM	Piecewise Linear Method
POM	Princeton Ocean Model
RCP	Remote CoPy
RMS	Root Mean Square
S	Salinity
SDD	Software Design Description

SSH	Sea Surface Height
SSS	Sea Surface Salinity
SST	Sea Surface Temperature
T	Temperature
TKE	Turbulent Kinetic Energy

APPENDIX A

HYCOM 2.2 Utility Commands

Utility	Description
`C`	Used as a "comment" line indicator in model scripts.
`clim_stat`	Lists contents of NRL "native" climatology file.
clim_stat.1	Manual page for `clim_stat`.
clim_stat.f	Source code for `clim_stat_machinetype`.
`echo2`	Echoes arguments to `stderr` (softlink).
echo2.c	Source code for `echo2_machinetype`.
`hycom2raw`	Outputs a raw copy of a HYCOM 2.2 ".a" file (softlink).
hycom2raw.F	Source code for `hycom2raw_machinetype`.
`hycom_alat`	HYCOM 2.2 grid statistics (softlink).
hycom_alat.f	Source code for `hycom_alat_machinetype`.
`hycom_depth`	HYCOM 2.2 *z*-depth statistics (softlink).
hycom_depth.f	Source code for `hycom_depth_machinetype`.
`hycom_expr`	Arithmetic expression of fields (softlink).
hycom_expr.F	Source code for `hycom_expr_machinetype`.
`hycom_ij2lonlat`	Longitude, latitude of an `ip`, `jp` point on the p-grid (softlink).
hycom_ij2lonlat.F	Source code for `hycom_ij2lonlat_machinetype`.
`hycom_lonlat2ij`	Nearest p-grid point to longitude, latitude (softlink).
hycom_lonlat2ij.F	Source code for `hycom_lonlat2ij_machinetype`.
`hycom_mxthrd`	HYCOM 2.2 OpenMP `mxthrd` statistics (softlink).
hycom_mxthrd.F	Source code for `hycom_mxthrd_machinetype`.
`hycom_nest_dates`	Archive dates needed for nesting (softlink).
hycom_nest_dates.f	Source code for `hycom_nest_dates_machinetype`.
`hycom_print`	Prints a sub-array (softlink).
hycom_print.F	Source code for `hycom_print_machinetype.`

Utility	Description
hycom_profile	Vertical profile from an archive (softlink).
hycom_profile.F	Source code for hycom_profile_machinetype.
hycom_range	HYCOM 2.2 ".a" file statistics (softlink).
hycom_range.F	Source code for hycom_range_machinetype.
hycom_range_ij	Utility hycom_range with location information (softlink).
hycom_range_ij.F	Source code for hycom_range_ij_machinetype.
hycom_rivers	Rivers within a longitude/latitude box (softlink).
hycom_rivers.F	Source code for hycom_rivers_machinetype.
hycom_rivers.d	Global database of rivers.
hycom_sea_ok	Same sea points as a bathymetry (softlink).
hycom_sea_ok.F	Source code for hycom_sea_ok_machinetype.
hycom_shift	Identical after a shift. (softlink).
hycom_shift.F	Source code for hycom_shift_machinetype.
hycom_sigma	Illustrates HYCOM 2.2 *z*-Sigma-*z*.
hycom_sigma.f	Source code for hycom_sigma_machinetype.
hycom_sigma.gnu	Gnuplot script used in hycom_sigma.
hycom_wind_date	Model day to yyyy_ddd_hh (softlink).
hycom_wind_date.f	Source code for hycom_wind_date_machinetype.
hycom_yoflat	HYCOM 2.2 Mercator grid latitudes (softlink).
hycom_yoflat.f	Source code for hycom_yoflat_machinetype.
hycom_zonal	Mean and root mean square of zonal extents (softlink).
hycom_zonal.F	Source code for hycom_zonal_machinetype.
mdel	Deletes a sequence of NQS jobs.
mlist	Creates a list of model runs in ./LIST.
msub	Submits a sequence of jobs (softlink to default version).
msub_codine	Submits a CODINE batch job (softlink to msub_grd).
msub_csh	Submits a background csh job.
msub_grd	Submits a GRD batch job.
msub_ll	Submits a LoadLeveler batch job.
msub_lsf	Submits a LSF batch job.
msub_nqs	Submits a NQS batch job.
msub_pbs	Submits a PBS batch job.

Utility	Description
`ncargf77`	NCAR Graphics f77 wrapper (softlink to script).
`ncargf77.4.1.1_SunOS`	Version 4.1.1 script for Solaris.
`ncargf90`	NCAR Graphics f90 wrapper (softlink to script).
`ncargf90.4.1.1_SunOS`	Version 4.1.1 script for Solaris.
`pget`	Gets one file (softlink to default version).
`pget_rcp`	Gets one file using RCP.
`pput`	Puts one file (softlink to default version).
`pput_rcp`	Puts one file using RCP.
`wind_stat`	Content list of NRL "native" wind file.
wind_stat.1	Manual page for `wind_stat`.
wind_stat.f	Source code for `wind_stat_machinetype`.
wind_stat_t3e.f	Source code for `wind_stat_t3e`.

APPENDIX B

HYCOM Data Structures (from subroutine *xcspmd*)

The following data structures are based on processor number and the patch distribution file **patch.input**.

Structure	Description
ipr	1st 2D node dimension (<=*iqr*).
jpr	2nd 2D node dimension (<=*jqr*).
ijpr	1D node dimension (*ipr*jpr*).
mproc	1st 2D node index.
nproc	2nd 2D node index.
mnproc	1D node index.
i0	I0_pe(*mproc,nproc*).
ii	Ii_pe(*mproc,nproc*).
j0	J0_pe(*mproc,nproc*).
jj	Jj_pe(*mproc,nproc*).
margin	Tells how much of the halo is currently valid.
nreg	Region type.
vland	Fill value for land (standard value 0.0).
idproc	2D node addresses with periodic wrap.
idproc1	1D node addresses with periodic wrap.
idhalo	Left and right halo target nodes.
i0_pe	1st dimension tile offsets.
ii_pe	1st dimension tile extents (<=*idm*).
j0_pe	2nd dimension tile offsets.
jj_pe	2nd dimension tile extents (<=*jdm*).
mpe_1	1st node in each row of 2D nodes.
mpe_e	End node in each row of 2D nodes.
mpe_i	Mapping from 1st global dimension to 2D nodes.
npe_j	Mapping from 2nd global dimension to 2D nodes.
i1sum	Local index of 1st partial sum on each tile.
iisum	Number of partial sums on each tile.
m0_top	Tile offset: top neighbors (0:*jpr*-1).
mm_top	Tile extent: top neighbors (<=*jpr*).
i0_st	Halo offsets: send top neighbors.
ii_st	Halo lengths: send top neighbors.
i0_gt	Halo offsets: get top neighbors.

ii_gt	Halo lengths: get top neighbors.
m0_bot	Tile offset: bottom neighbors (0:*jpr*-1).
mm_bot	Tile extent: bottom neighbors (<=*jpr*).
i0_sb	Halo offsets: send bottom neighbors.
ii_sb	Halo lengths: send bottom neighbors.
i0_gb	Halo offsets: get bottom neighbors.
ii_gb	Halo lengths: get bottom neighbors.