Contents

1 Overview of MITgcm **9**

1.1 Introduction . . . 9
1.2 Illustrations of the model in action . . . 13
1.2.1 Global atmosphere: 'Held-Suarez' benchmark . . . 13
1.2.2 Ocean gyres . . . 15
1.2.3 Global ocean circulation . . . 15
1.2.4 Convection and mixing over topography . . . 15
1.2.5 Boundary forced internal waves . . . 19
1.2.6 Parameter sensitivity using the adjoint of MITgcm . . . 21
1.2.7 Global state estimation of the ocean . . . 21
1.2.8 Ocean biogeochemical cycles . . . 22
1.2.9 Simulations of laboratory experiments . . . 22
1.3 Continuous equations in 'r' coordinates . . . 22
1.3.1 Kinematic Boundary conditions . . . 29
1.3.2 Atmosphere . . . 29
1.3.3 Ocean . . . 30
1.3.4 Hydrostatic, Quasi-hydrostatic, Quasi-nonhydrostatic and Non-hydrostatic forms . . . 31
1.3.5 Solution strategy . . . 35
1.3.6 Finding the pressure field . . . 36
1.3.7 Forcing/dissipation . . . 38
1.3.8 Vector invariant form . . . 39
1.3.9 Adjoint . . . 39
1.4 Appendix ATMOSPHERE . . . 39
1.4.1 Hydrostatic Primitive Equations for the Atmosphere in pressure coordinates . . . 39
1.5 Appendix OCEAN . . . 41
1.5.1 Equations of motion for the ocean . . . 41
1.6 Appendix:OPERATORS . . . 46
1.6.1 Coordinate systems . . . 46

2 Discretization and Algorithm **47**
2.1 Time-stepping . . . 47
2.2 Pressure method with rigid-lid . . . 48
2.3 Pressure method with implicit linear free-surface . . . 51
2.4 Explicit time-stepping: Adams-Bashforth . . . 52
2.5 Implicit time-stepping: backward method . . . 53
2.6 Synchronous time-stepping: variables co-located in time . . . 53
2.7 Staggered baroclinic time-stepping . . . 56
2.8 Non-hydrostatic formulation . . . 58
2.9 Variants on the Free Surface . . . 61
2.9.1 Crank-Nickelson barotropic time stepping . . . 62
2.9.2 Non-linear free-surface . . . 64
2.10 Spatial discretization of the dynamical equations . . . 70
2.10.1 Notation . . . 70
2.10.2 The finite volume method: finite volumes versus finite difference . . . 70
2.10.3 C grid staggering of variables . . . 71
2.10.4 Grid initialization and data . . . 72
2.10.5 Horizontal grid . . . 72
2.10.6 Vertical grid . . . 75
2.10.7 Topography: partially filled cells . . . 77
2.11 Continuity and horizontal pressure gradient terms . . . 79
2.12 Hydrostatic balance . . . 80
2.13 Flux-form momentum equations . . . 80
2.13.1 Advection of momentum . . . 81
2.13.2 Coriolis terms . . . 82
2.13.3 Curvature metric terms . . . 82
2.13.4 Non-hydrostatic metric terms . . . 83
2.13.5 Lateral dissipation . . . 84
2.13.6 Vertical dissipation . . . 85
2.13.7 Derivation of discrete energy conservation . . . 86
2.14 Vector invariant momentum equations . . . 86
2.14.1 Relative vorticity . . . 87
2.14.2 Kinetic energy . . . 87
2.14.3 Coriolis terms . . . 88
2.14.4 Shear terms . . . 88
2.14.5 Gradient of Bernoulli function . . . 89
2.14.6 Horizontal dissipation . . . 89
2.14.7 Horizontal dissipation . . . 89
2.14.8 Vertical dissipation . . . 90
2.15 Tracer equations . . . 90
2.15.1 Time-stepping of tracers: ABII . . . 91
2.16 Linear advection schemes . . . 92
2.16.1 Centered second order advection-diffusion . . . 92
2.16.2 Third order upwind bias advection . . . 95
2.16.3 Centered fourth order advection . . . 96

2.16.4 First order upwind advection . . . 96
2.17 Non-linear advection schemes . . . 97
2.17.1 Second order flux limiters . . . 97
2.17.2 Third order direct space time . . . 98
2.17.3 Third order direct space time with flux limiting . . . 99
2.17.4 Multi-dimensional advection . . . 103
2.18 Comparison of advection schemes . . . 103
2.19 Shapiro Filter . . . 105

3 Getting Started with MITgcm 107
3.1 Where to find information . . . 107
3.2 Obtaining the code . . . 108
3.2.1 Method 1 - Checkout from CVS . . . 108
3.2.2 Method 2 - Tar file download . . . 109
3.3 Model and directory structure . . . 111
3.4 MITgcm Example Experiments . . . 112
3.4.1 Full list of model examples . . . 112
3.4.2 Directory structure of model examples . . . 113
3.5 Building MITgcm . . . 114
3.6 Running MITgcm . . . 115
3.6.1 Output files . . . 116
3.6.2 Looking at the output . . . 117
3.7 Tutorials . . . 119
3.8 Barotropic Gyre MITgcm Example . . . 119
3.8.1 Equations Solved . . . 119
3.8.2 Discrete Numerical Configuration . . . 121
3.8.3 Code Configuration . . . 122
3.9 Baroclinic Gyre MITgcm Example . . . 129
3.9.1 Overview . . . 129
3.9.2 Equations solved . . . 130
3.9.3 Discrete Numerical Configuration . . . 131
3.9.4 Code Configuration . . . 133
3.9.5 Running The Example . . . 141
3.10 Global Ocean MITgcm Exmaple . . . 143
3.10.1 Overview . . . 143
3.10.2 Discrete Numerical Configuration . . . 144
3.10.3 Experiment Configuration . . . 146
3.11 P coordinate Global Ocean MITgcm Example . . . 157
3.11.1 Overview . . . 157
3.11.2 Discrete Numerical Configuration . . . 158
3.11.3 Experiment Configuration . . . 159
3.12 Held-Suarez Atmosphere MITgcm Example . . . 176
3.12.1 Overview . . . 176
3.12.2 Discrete Numerical Configuration . . . 177
3.12.3 Experiment Configuration . . . 180
3.13 Surface Driven Convection . . . 190

3.13.1 Overview . . . 190
3.13.2 Equations solved . . . 193
3.13.3 Discrete numerical configuration . . . 193
3.13.4 Numerical stability criteria and other considerations . . . 193
3.13.5 Experiment configuration . . . 194
3.13.6 Running the example . . . 203
3.14 Gravity Plume On a Continental Slope . . . 205
3.14.1 Configuration . . . 206
3.14.2 Binary input data . . . 206
3.14.3 Code configuration . . . 209
3.14.4 Model parameters . . . 210
3.14.5 Build and run the model . . . 210
3.15 Centennial Time Scale Tracer Injection . . . 212
3.15.1 Introduction . . . 212
3.15.2 Overview . . . 212
3.15.3 Discrete Numerical Configuration . . . 213
3.15.4 Code Configuration . . . 215
3.16 Customizing MITgcm . . . 220
3.16.1 Building/compiling the code elsewhere . . . 220
3.16.2 Using `genmake2` . . . 222
3.16.3 Building with MPI . . . 225
3.16.4 Configuration and setup . . . 226
3.16.5 Computational domain, geometry and time-discretization 226
3.16.6 Equation of state . . . 228
3.16.7 Momentum equations . . . 229
3.16.8 Tracer equations . . . 231
3.16.9 Simulation controls . . . 232
3.17 Testing . . . 234
3.17.1 Using `testreport` . . . 234
3.17.2 Automated testing . . . 234

4 Software Architecture 235
4.1 Overall architectural goals . . . 235
4.2 WRAPPER . . . 237
4.2.1 Target hardware . . . 237
4.2.2 Supporting hardware neutrality . . . 237
4.2.3 WRAPPER machine model . . . 239
4.2.4 Machine model parallelism . . . 239
4.2.5 Communication mechanisms . . . 240
4.2.6 Shared memory communication . . . 241
4.2.7 Distributed memory communication . . . 244
4.2.8 Communication primitives . . . 244
4.2.9 Memory architecture . . . 246
4.2.10 Summary . . . 248
4.3 Using the WRAPPER . . . 248
4.3.1 Specifying a domain decomposition . . . 249

4.3.2 Starting the code . . . 254
4.3.3 Controlling communication . . . 259
4.4 MITgcm execution under WRAPPER . . . 265
4.4.1 Annotated call tree for MITgcm and WRAPPER . . . 265
4.4.2 Measuring and Characterizing Performance . . . 272
4.4.3 Estimating Resource Requirements . . . 272

5 Automatic Differentiation 273
5.1 Some basic algebra . . . 273
5.1.1 Forward or direct sensitivity . . . 274
5.1.2 Reverse or adjoint sensitivity . . . 274
5.1.3 Storing vs. recomputation in reverse mode . . . 278
5.2 TLM and ADM generation in general . . . 280
5.2.1 General setup . . . 281
5.2.2 Building the AD code . . . 282
5.2.3 The AD build process in detail . . . 283
5.2.4 The cost function (dependent variable) . . . 285
5.2.5 The control variables (independent variables) . . . 287
5.3 Sensitivity of Air-Sea Exchange to Tracer Injection Site . . . 294
5.3.1 Overview of the experiment . . . 294
5.3.2 Code configuration . . . 295
5.3.3 Compiling the model and its adjoint . . . 301
5.4 The gradient check package . . . 304
5.4.1 Code description . . . 304
5.4.2 Code configuration . . . 304
5.5 Adjoint dump & restart – divided adjoint (DIVA) . . . 305
5.5.1 Introduction . . . 305
5.5.2 Recipe 1: single processor . . . 307
5.5.3 Recipe 2: multi processor (MPI) . . . 308

6 Physical Parameterization and Packages 309
6.1 Using MITgcm Packages . . . 310
6.1.1 Package Inclusion/Exclusion . . . 310
6.1.2 Package Activation . . . 311
6.2 Package Coding Standards . . . 311
6.2.1 Packages are Not Libraries . . . 311
6.3 Gent/McWiliams/Redi SGS Eddy parameterization . . . 315
6.3.1 Redi scheme: Isopycnal diffusion . . . 315
6.3.2 GM parameterization . . . 316
6.3.3 Griffies Skew Flux . . . 316
6.3.4 Variable κ_{GM} . . . 317
6.3.5 Tapering and stability . . . 318
6.3.6 Tapering: Danabasoglu and McWilliams, J. Clim. 1995 . . . 320
6.3.7 Tapering: Large, Danabasoglu and Doney, JPO 1997 . . . 321
6.3.8 Package Reference . . . 321
6.4 DIC Package . . . 322

6.4.1 Introduction . . . 322
6.4.2 Key subroutines and parameters . . . 322
6.4.3 Do's and Don'ts . . . 323
6.4.4 Reference Material . . . 323
6.5 Ocean vertical mixing – the nonlocal K-profile parameterization scheme KPP . . . 324
6.6 Thermodynamic Sea Ice Package: "thsice" . . . 325
6.7 Sea Ice Package: "seaice" . . . 332
6.8 Bulk Formula Package . . . 333
6.9 Generic Advection/Diffusion . . . 338
6.9.1 Introduction . . . 338
6.9.2 Key subroutines, parameters and files . . . 338
6.10 Atmospheric Intermediate Physics: AIM . . . 339
6.10.1 Key subroutines, parameters and files . . . 339
6.11 Land package . . . 340
6.12 Coupling interface for Atmospheric Intermediate code . . . 342
6.12.1 Key subroutines, parameters and files . . . 342
6.13 Coupler for mapping between AIM and ocean . . . 343
6.13.1 Key subroutines, parameters and files . . . 343
6.14 Toolkit for building couplers . . . 344
6.14.1 Key subroutines, parameters and files . . . 344
6.15 NetCDF I/O Integration: MNC . . . 345
6.15.1 Using MNC . . . 345
6.15.2 MNC Internals . . . 347
6.16 MDSIO . . . 351
6.17 Simulation state monitoring toolkit . . . 352
6.17.1 Key subroutines, parameters and files . . . 352
6.18 exch2: Extended Cubed Sphere Topology . . . 353
6.18.1 Introduction . . . 353
6.18.2 Invoking exch2 . . . 353
6.18.3 Generating Topology Files for exch2 . . . 354
6.18.4 exch2, SIZE.h, and Multiprocessing . . . 356
6.18.5 Key Variables . . . 359
6.18.6 Key Routines . . . 362
6.19 Diagnostics–A Flexible Infrastructure . . . 364
6.19.1 Introduction . . . 364
6.19.2 Equations . . . 364
6.19.3 Key Subroutines and Parameters . . . 364
6.19.4 Usage Notes . . . 367
6.19.5 Dos and Donts . . . 404
6.19.6 Diagnostics Reference . . . 404
6.20 RW Basic binary I/O utilities . . . 405
6.20.1 Introduction . . . 405
6.21 FFT Filtering Code . . . 406
6.21.1 Key subroutines, parameters and files . . . 406
6.22 GCHEM Package . . . 407

6.22.1 Introduction . . . 407
6.22.2 Key subroutines and parameters . . . 407
6.22.3 Do's and Don'ts . . . 408
6.22.4 Reference Material . . . 409

7 Diagnostics and tools **411**
7.1 Utilities supplied with the model . . . 411
7.1.1 utils/scripts . . . 411
7.1.2 utils/matlab . . . 411
7.2 Pre-processing software . . . 412

8 Interface with ECCO **413**
8.1 The ECCO state estimation cost function DRAFT!!! . . . 413
8.1.1 Sea surface height from TOPEX/Poseidon and ERS-1/2 altimetry . . . 413
8.1.2 Hydrographic constraints . . . 416
8.2 The external forcing package `exf` . . . 420
8.2.1 Summary . . . 420
8.3 The calendar package `cal` . . . 423
8.3.1 Basic assumptions for the calendar tool . . . 423
8.3.2 Format of calendar dates . . . 423
8.3.3 Calendar dates and time intervals . . . 424
8.3.4 Using the calendar together with MITgcm . . . 424
8.3.5 The individual calendars . . . 425
8.3.6 Short routine description . . . 425
8.4 The line search optimisation algorithm . . . 428
8.4.1 General features . . . 428
8.4.2 The online vs. offline version . . . 428
8.4.3 Number of iterations vs. number of simulations . . . 428

9 Model Uses **435**

BIBLIOGRAPHY **439**

Chapter 1

Overview of MITgcm

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

1.1 Introduction

MITgcm has a number of novel aspects:

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

Key publications reporting on and charting the development of the model are [30, 39, 38, 5, 40, 6, 9, 37]:

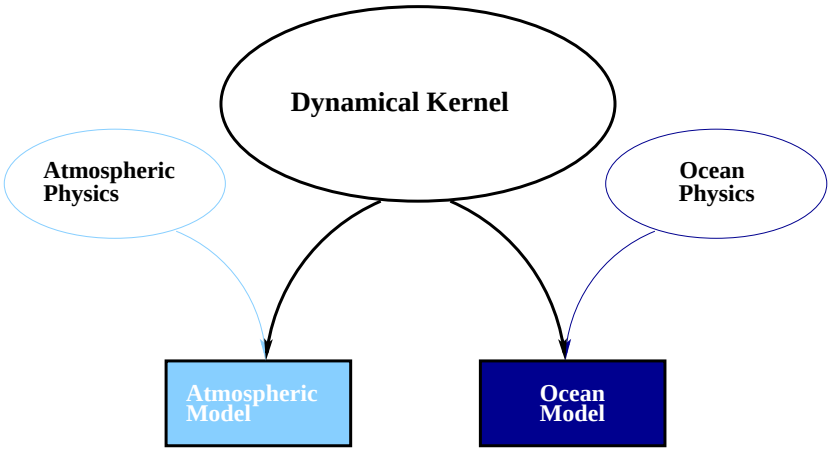


Figure 1.1: MITgcm has a single dynamical kernel that can drive forward either oceanic or atmospheric simulations.

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

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

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

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

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

Adcroft, A., Hill C. and J. Marshall: (1999)

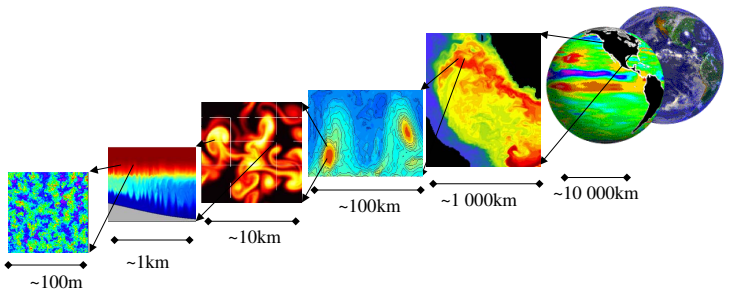


Figure 1.2: MITgcm has non-hydrostatic capabilities, allowing the model to address a wide range of phenomenon - from convection on the left, all the way through to global circulation patterns on the right.

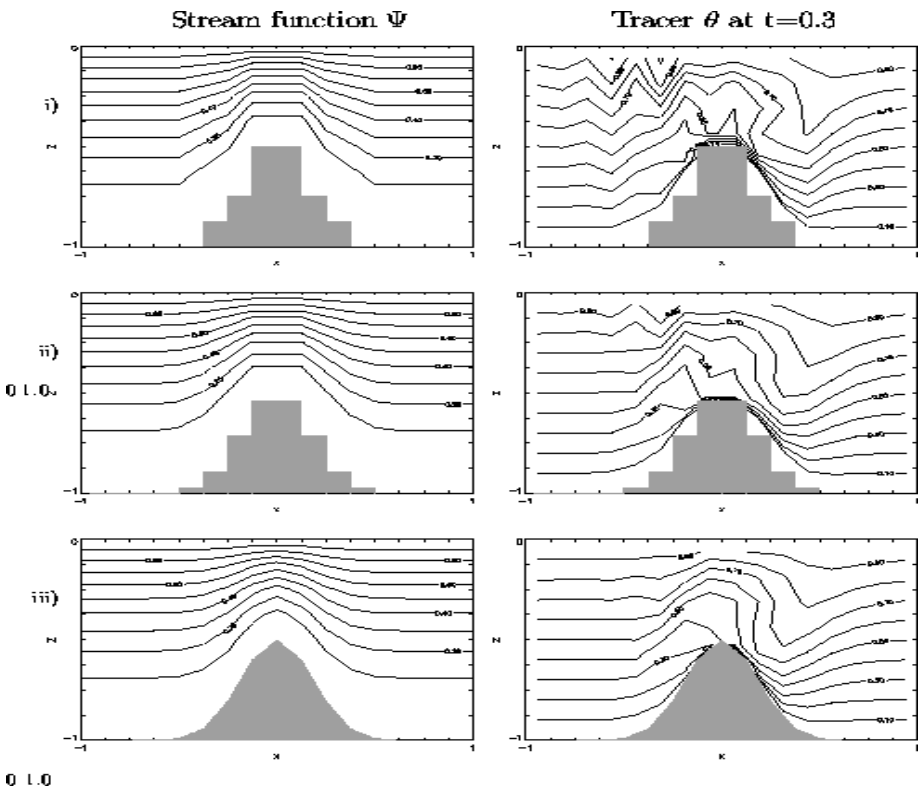


Figure 1.3: Finite volume techniques (bottom panel) are user, permitting a treatment of topography that rivals σ (terrain following) coordinates.

A new treatment of the Coriolis terms in C-grid models at both high and low resolutions,
Mon. Wea. Rev. Vol 127, pages 1928-1936

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

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

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

1.2 Illustrations of the model in action

The MITgcm has been designed and used to model a wide range of phenomena, from convection on the scale of meters in the ocean to the global pattern of atmospheric winds - see figure 1.2. To give a flavor of the kinds of problems the model has been used to study, we briefly describe some of them here. A more detailed description of the underlying formulation, numerical algorithm and implementation that lie behind these calculations is given later. Indeed many of the illustrative examples shown below can be easily reproduced: simply download the model (the minimum you need is a PC running Linux, together with a FORTRAN 77 compiler) and follow the examples described in detail in the documentation.

1.2.1 Global atmosphere: 'Held-Suarez' benchmark

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

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

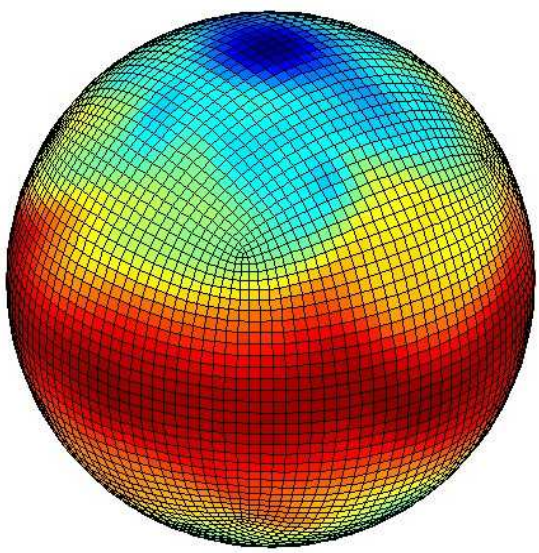


Figure 1.4: Instantaneous plot of the temerature field at 500mb obtained using the atmospheric isomorph of MITgcm

As described in Adcroft (2001), a 'cubed sphere' is used to discretize the globe permitting a uniform griding and obviated the need to Fourier filter. The 'vector-invariant' form of MITgcm supports any orthogonal curvilinear grid, of which the cubed sphere is just one of many choices.

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

1.2.2 Ocean gyres

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

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

1.2.3 Global ocean circulation

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

Figure 1.7 (bottom) shows the meridional overturning circulation of the global ocean in Sverdrups.

1.2.4 Convection and mixing over topography

Dense plumes generated by localized cooling on the continental shelf of the ocean may be influenced by rotation when the deformation radius is smaller than the width of the cooling region. Rather than gravity plumes, the mechanism for moving dense fluid down the shelf is then through geostrophic eddies. The simulation shown in the figure 1.8 (blue is cold dense fluid, red is warmer, lighter

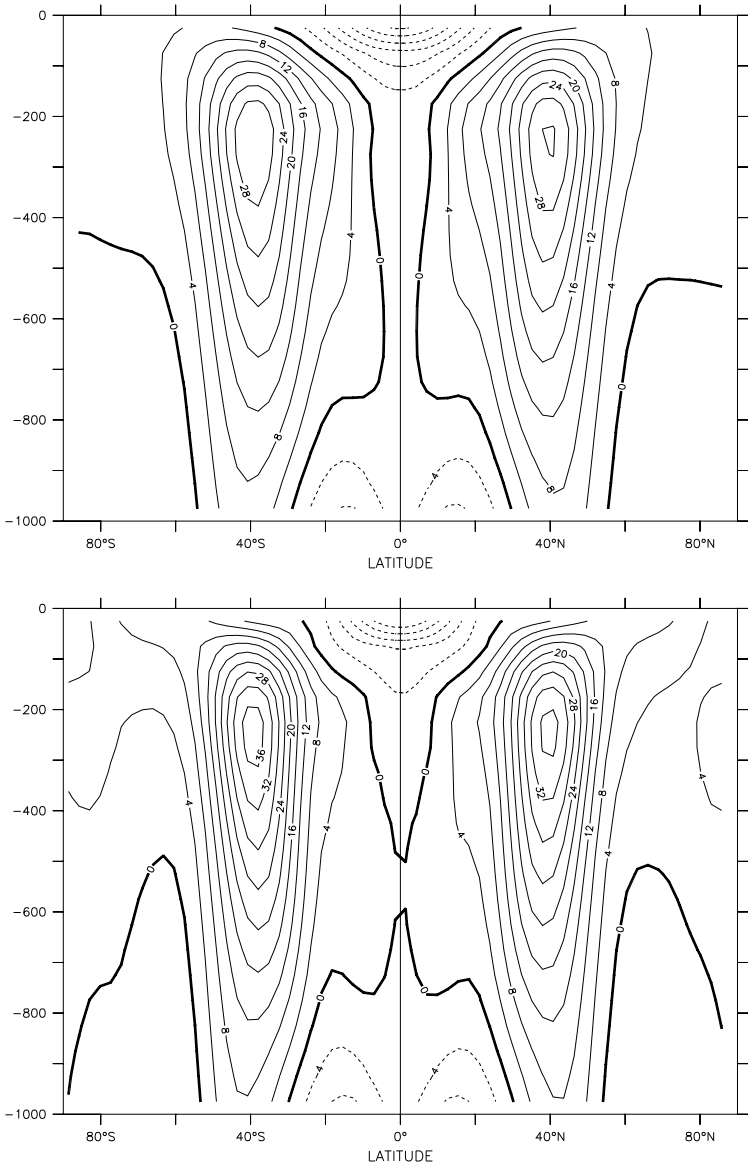


Figure 1.5: Five year mean, zonally averaged zonal flow for latitude-longitude simulation (bottom) and cube-sphere simulation(top) using Held-Suarez forcing. Note the difference in the solutions over the pole - the cubed sphere is superior.

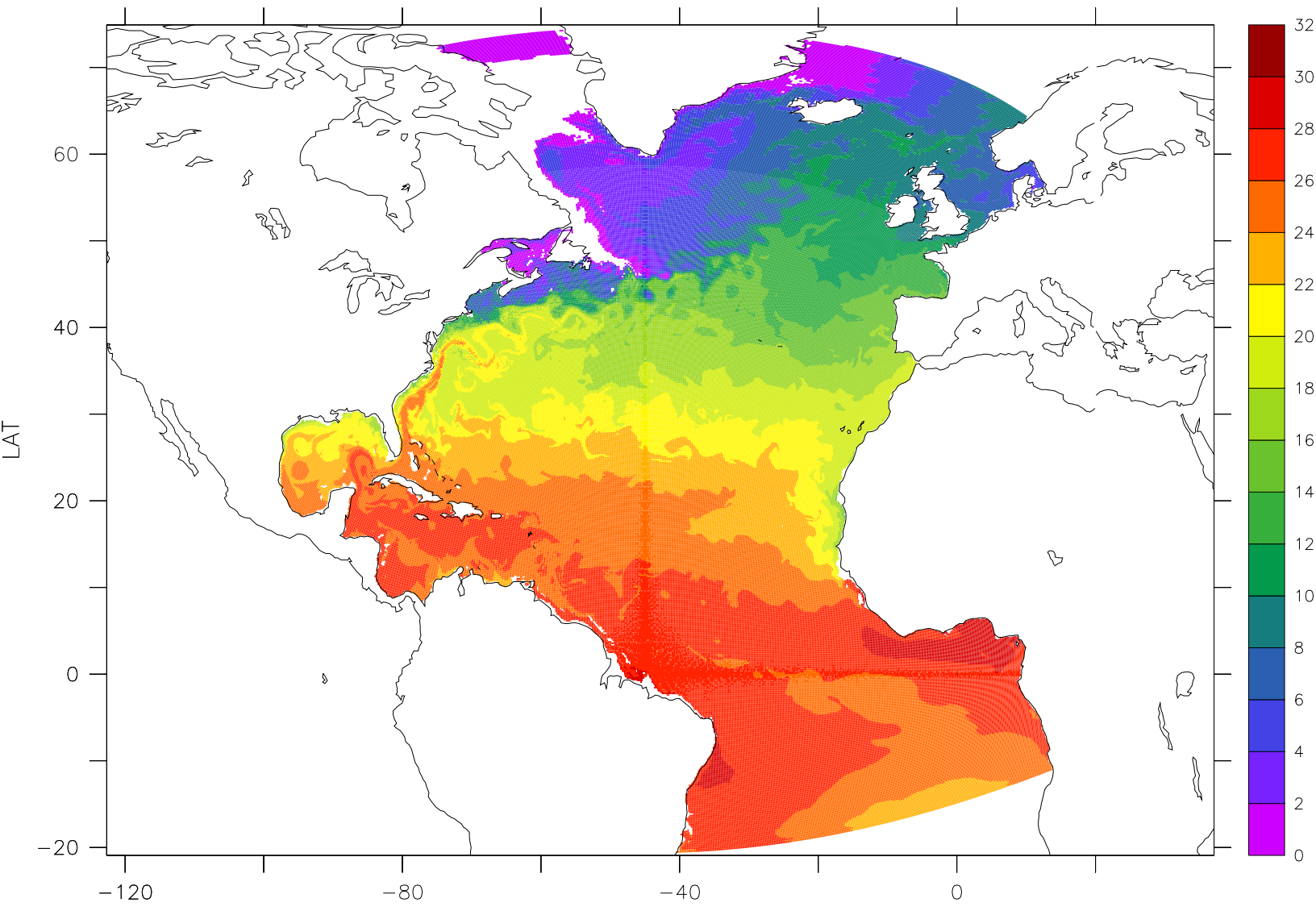


Figure 1.6: Instantaneous temperature map from a $\frac{1}{6}^\circ$ simulation of the North Atlantic. The figure shows the temperature in the second layer (37.5m deep).

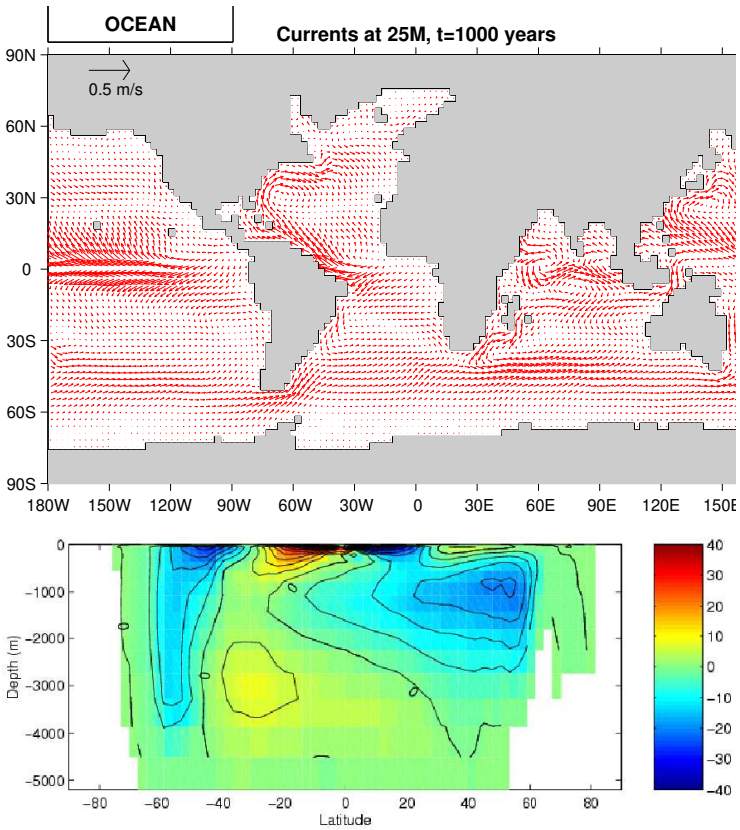


Figure 1.7: Pattern of surface ocean currents (top) and meridional overturning stream function (in Sverdrups) from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

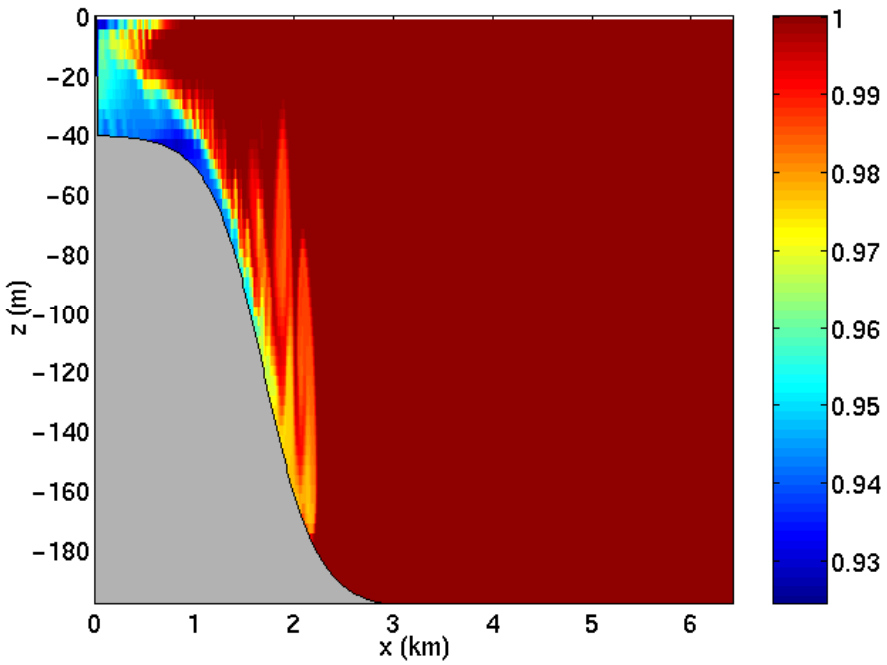


Figure 1.8: MITgcm run in a non-hydrostatic configuration to study convection over a slope.

fluid) employs the non-hydrostatic capability of MITgcm to trigger convection by surface cooling. The cold, dense water falls down the slope but is deflected along the slope by rotation. It is found that entrainment in the vertical plane is reduced when rotational control is strong, and replaced by lateral entrainment due to the baroclinic instability of the along-slope current.

1.2.5 Boundary forced internal waves

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

Fig. 1.9 shows the influence of cross-slope topographic variations on internal wave breaking - the cross-slope velocity is in color, the density contoured. The internal waves are excited by application of open boundary conditions on the left. They propagate to the sloping boundary (represented using MITgcm's finite volume spatial discretization) where they break under nonhydrostatic dynamics.

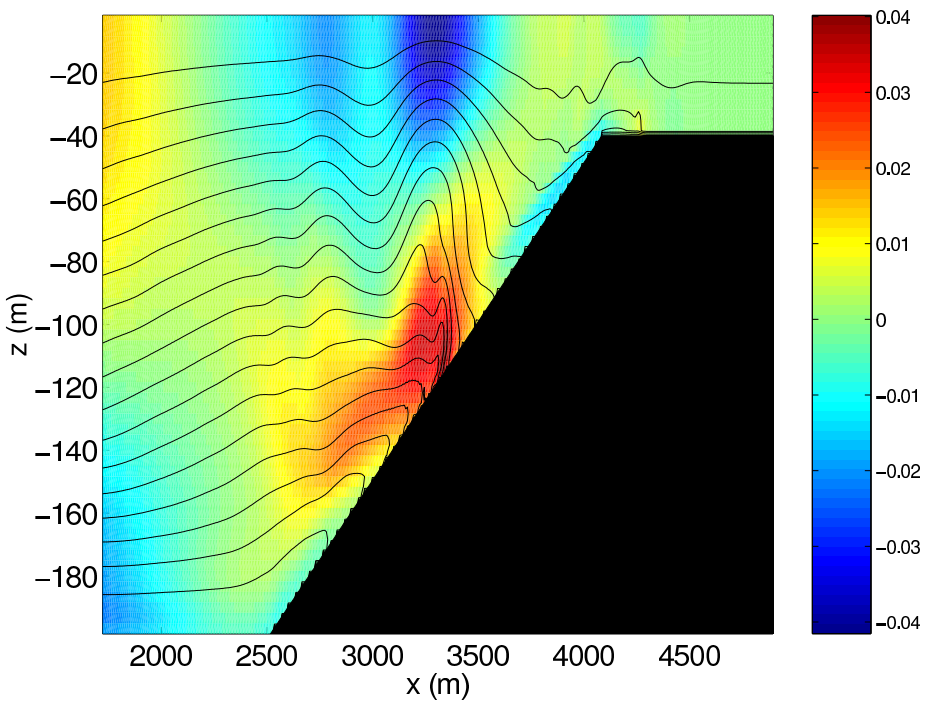


Figure 1.9: Simulation of internal waves forced at an open boundary (on the left) impacting a sloping shelf. The along slope velocity is shown colored, contour lines show density surfaces. The slope is represented with high-fidelity using lopped cells.

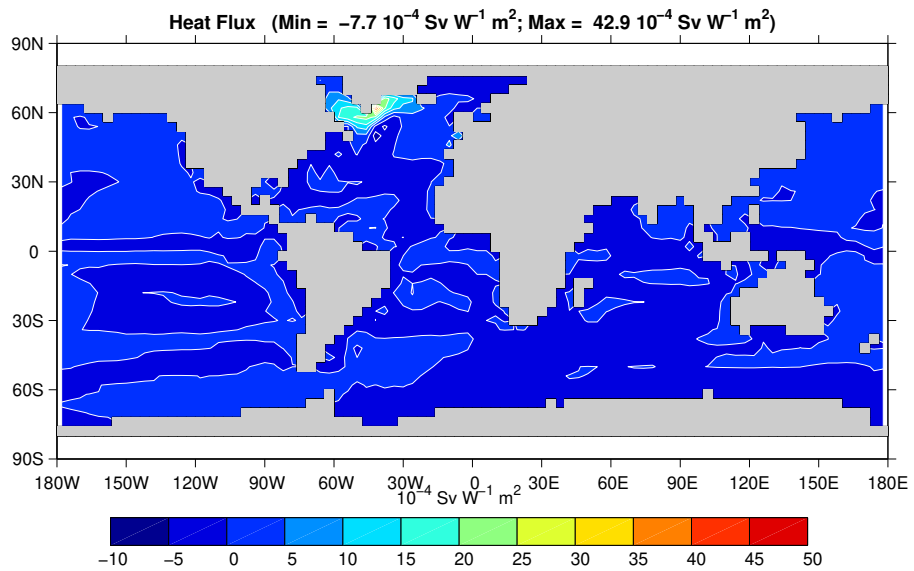


Figure 1.10: Sensitivity of meridional overturning strength to surface heat flux changes. Contours show the magnitude of the response (in Sv $\times 10^{-4}$) that a persistent $+1\mathrm{Wm}^{-2}$ heat flux anomaly at a given gid point would produce.

1.2.6 Parameter sensitivity using the adjoint of MITgcm

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

As one example of application of the MITgcm adjoint, Figure 1.10 maps the gradient $\frac{\partial J}{\partial \mathcal{H}}$where J is the magnitude of the overturning stream-function shown in figure 1.7 at 60°N and $\mathcal{H}(\lambda, \varphi)$ is the mean, local air-sea heat flux over a 100 year period. We see that J is sensitive to heat fluxes over the Labrador Sea, one of the important sources of deep water for the thermohaline circulations. This calculation also yields sensitivities to all other model parameters.

1.2.7 Global state estimation of the ocean

An important application of MITgcm is in state estimation of the global ocean circulation. An appropriately defined 'cost function', which measures the departure of the model from observations (both remotely sensed and in-situ) over an interval of time, is minimized by adjusting 'control parameters' such as air-sea fluxes, the wind field, the initial conditions etc. Figure 1.11 shows the large scale planetary circulation and a Hopf-Muller plot of Equatorial sea-surface height. Both are obtained from assimilation bringing the model in to consistency with

altimetric and in-situ observations over the period 1992-1997.

1.2.8 Ocean biogeochemical cycles

MITgcm is being used to study global biogeochemical cycles in the ocean. For example one can study the effects of interannual changes in meteorological forcing and upper ocean circulation on the fluxes of carbon dioxide and oxygen between the ocean and atmosphere. Figure 1.12 shows the annual air-sea flux of oxygen and its relation to density outcrops in the southern oceans from a single year of a global, interannually varying simulation. The simulation is run at $1^\circ \times 1^\circ$ resolution telescoping to $\frac{1}{3}^\circ \times \frac{1}{3}^\circ$ in the tropics (not shown).

1.2.9 Simulations of laboratory experiments

Figure 1.13 shows MITgcm being used to simulate a laboratory experiment inquiring into the dynamics of the Antarctic Circumpolar Current (ACC). An initially homogeneous tank of water ($1m$ in diameter) is driven from its free surface by a rotating heated disk. The combined action of mechanical and thermal forcing creates a lens of fluid which becomes baroclinically unstable. The stratification and depth of penetration of the lens is arrested by its instability in a process analogous to that which sets the stratification of the ACC.

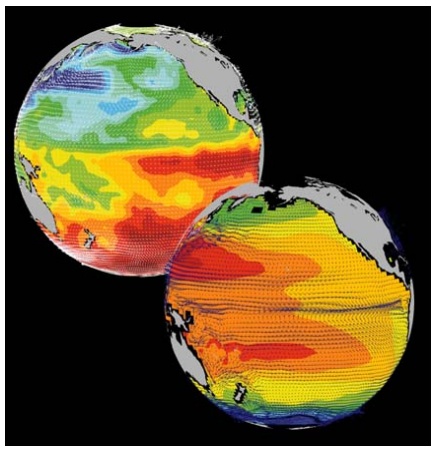
1.3 Continuous equations in 'r' coordinates

To render atmosphere and ocean models from one dynamical core we exploit 'isomorphisms' between equation sets that govern the evolution of the respective fluids - see figure 1.14. One system of hydrodynamical equations is written down and encoded. The model variables have different interpretations depending on whether the atmosphere or ocean is being studied. Thus, for example, the vertical coordinate 'r' is interpreted as pressure, p, if we are modeling the atmosphere (right hand side of figure 1.14) and height, z, if we are modeling the ocean (left hand side of figure 1.14).

The state of the fluid at any time is characterized by the distribution of velocity $\vec{\mathbf{v}}$, active tracers θ and S, a 'geopotential' ϕ and density $\rho = \rho(\theta, S, p)$ which may depend on θ, S, and p. The equations that govern the evolution of these fields, obtained by applying the laws of classical mechanics and thermodynamics to a Boussinesq, Navier-Stokes fluid are, written in terms of a generic vertical coordinate, r, so that the appropriate kinematic boundary conditions can be applied isomorphically see figure 1.15.

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

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



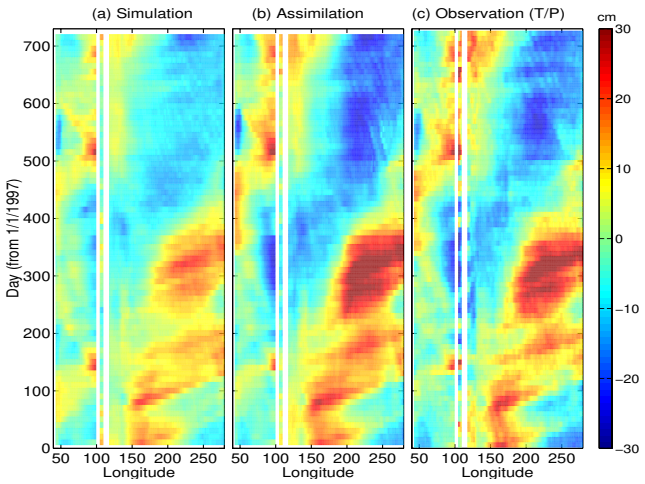


Figure 1.11: Top panel shows circulation patterns from a multi-year, global circulation simulation constrained by Topex altimeter data and WOCE cruise observations. Bottom panel shows the equatorial sea-surface height in unconstrained (left), constrained (middle) simulations and in observations. This output is from a higher resolution, shorter duration experiment with equatorially enhanced grid spacing.

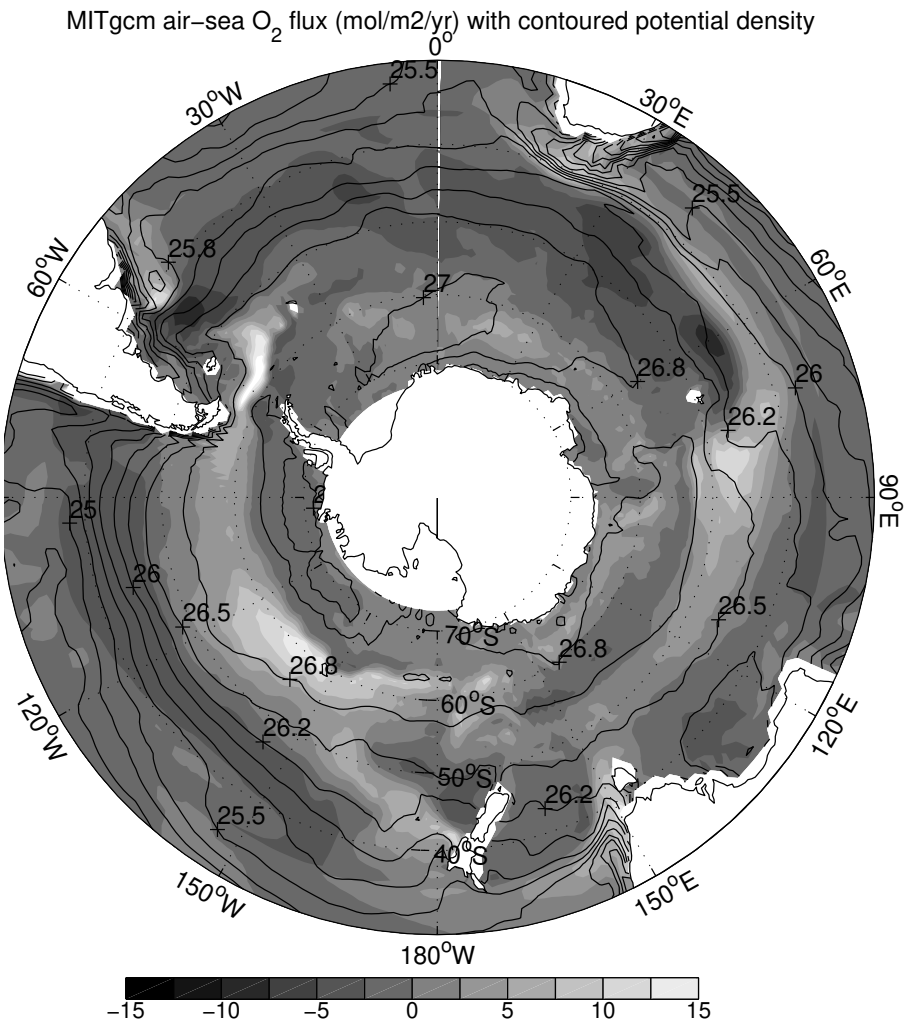


Figure 1.12: Annual air-sea flux of oxygen (shaded) plotted along with potential density outcrops of the surface of the southern ocean from a global $1^\circ \times 1^\circ$ integation with a telescoping grid (to $\frac{1}{3}^\circ$) at the equator.

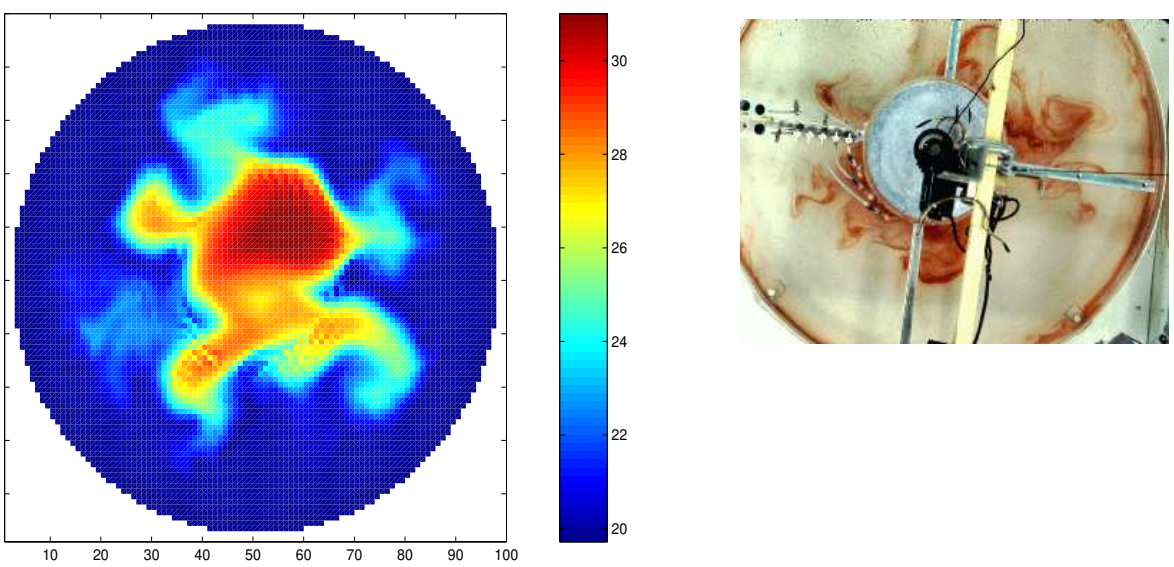


Figure 1.13: A numerical simulation (left) of a 1m diameter laboratory experiment (right) using MITgcm.

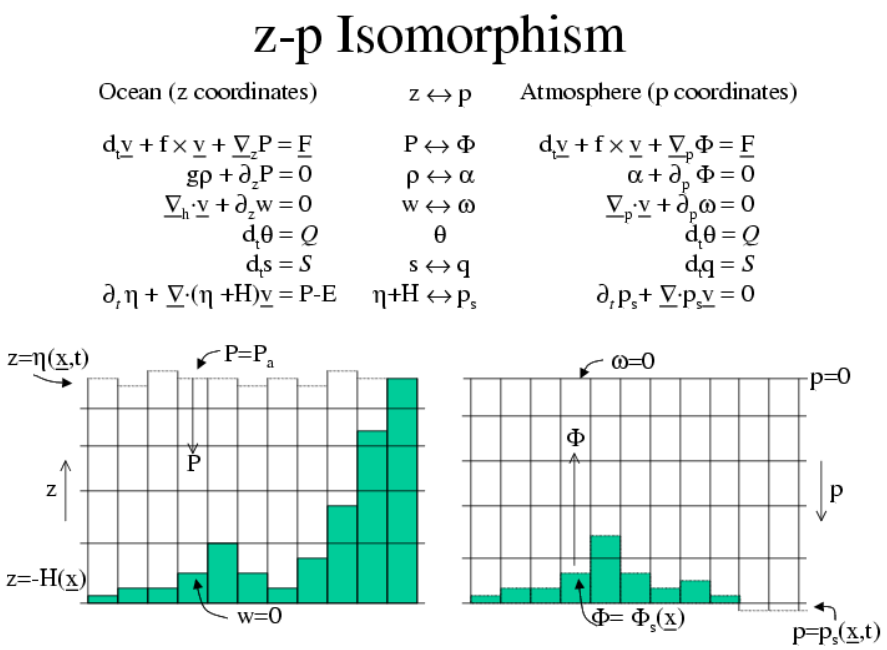


Figure 1.14: Isomorphic equation sets used for atmosphere (right) and ocean (left).

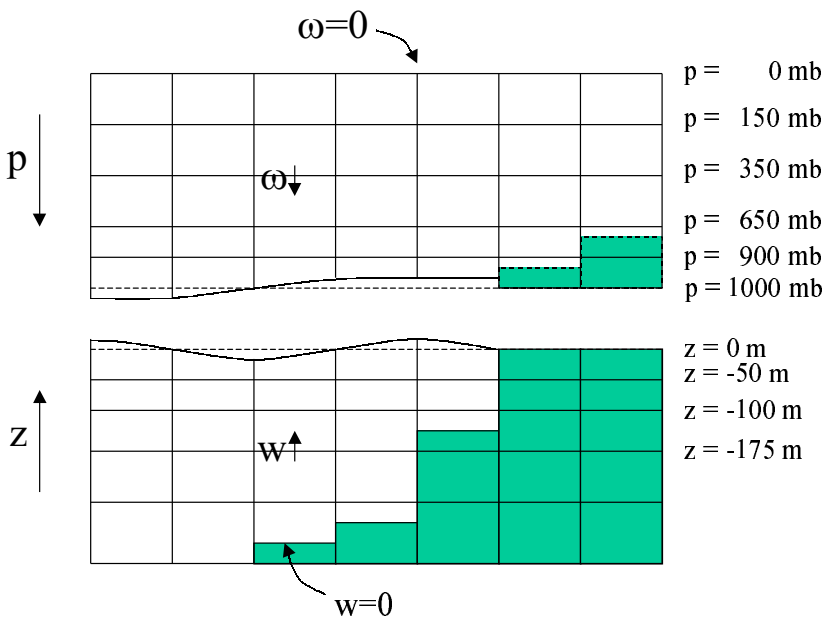


Figure 1.15: Vertical coordinates and kinematic boundary conditions for atmosphere (top) and ocean (bottom).

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

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

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

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

Here:

r is the vertical coordinate

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

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

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

t is time

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

ϕ is the 'pressure'/'geopotential'

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

b is the 'buoyancy'

θ is potential temperature

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

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

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

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

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

1.3.1 Kinematic Boundary conditions

1.3.1.1 vertical

at fixed and moving r surfaces we set (see figure 1.15):

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

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

Here

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

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

1.3.1.2 horizontal

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

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

1.3.2 Atmosphere

In the atmosphere, (see figure 1.15), we interpret:

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

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

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

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

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

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

where

$$T \text{ is absolute temperature}$$

p is the pressure

z is the height of the pressure surface

g is the acceleration due to gravity

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

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

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

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

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

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

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

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

The boundary conditions at top and bottom are given by:

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

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

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

1.3.3 Ocean

In the ocean we interpret:

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

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

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

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

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

In the above
At the bottom of the ocean: $R_{fixed}(x,y) = -H(x,y)$.
The surface of the ocean is given by: $R_{moving} = \eta$
The position of the resting free surface of the ocean is given by $R_o = Z_o = 0$.
Boundary conditions are:

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

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

where η is the elevation of the free surface.

Then equations (1.1-1.6) yield a consistent set of oceanic equations which, for convenience, are written out in z coordinates in Appendix Ocean - see eqs(1.99) to (1.104).

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

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

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

and write eq(1.1) in the form:

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

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

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

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

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

$$\left.\begin{array}{l} G_u = -\vec{\mathbf{v}}.\nabla u \\ -\left\{\underline{\frac{u\dot{r}}{r}} - \frac{uv\tan\varphi}{r}\right\} \\ -\left\{\underline{-2\Omega v \sin\varphi + 2\Omega \dot{r}\cos\varphi}\right\} \\ +\mathcal{F}_u \end{array}\right\} \left\{\begin{array}{l} \textit{advection} \\ \textit{metric} \\ \textit{Coriolis} \\ \textit{Forcing/Dissipation} \end{array}\right. \tag{1.29}$$

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

$$\left.\begin{array}{l} G_v = -\vec{\mathbf{v}}.\nabla v \\ -\left\{\underline{\frac{v\dot{r}}{r}} - \underline{\underline{\frac{u^2\tan\varphi}{r}}}\right\} \\ -\{-2\Omega u\sin\varphi\} \\ +\mathcal{F}_v \end{array}\right\} \left\{\begin{array}{l} \textit{advection} \\ \textit{metric} \\ \textit{Coriolis} \\ \textit{Forcing/Dissipation} \end{array}\right. \tag{1.30}$$

$$\left.\begin{array}{l} G_{\dot{r}} = -\underline{\underline{\vec{\mathbf{v}}.\nabla\dot{r}}} \\ +\left\{\underline{\underline{\frac{u^2+v^2}{r}}}\right\} \\ +\underline{\underline{2\Omega u\cos\varphi}} \\ \underline{\underline{\mathcal{F}_{\dot{r}}}} \end{array}\right\} \left\{\begin{array}{l} \textit{advection} \\ \textit{metric} \\ \textit{Coriolis} \\ \textit{Forcing/Dissipation} \end{array}\right. \tag{1.31}$$

In the above 'r' is the distance from the center of the earth and 'φ ' is latitude.

Grad and div operators in spherical coordinates are defined in appendix OPERATORS.

1.3.4.1 Shallow atmosphere approximation

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

1.3.4.2 Hydrostatic and quasi-hydrostatic forms

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

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

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

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

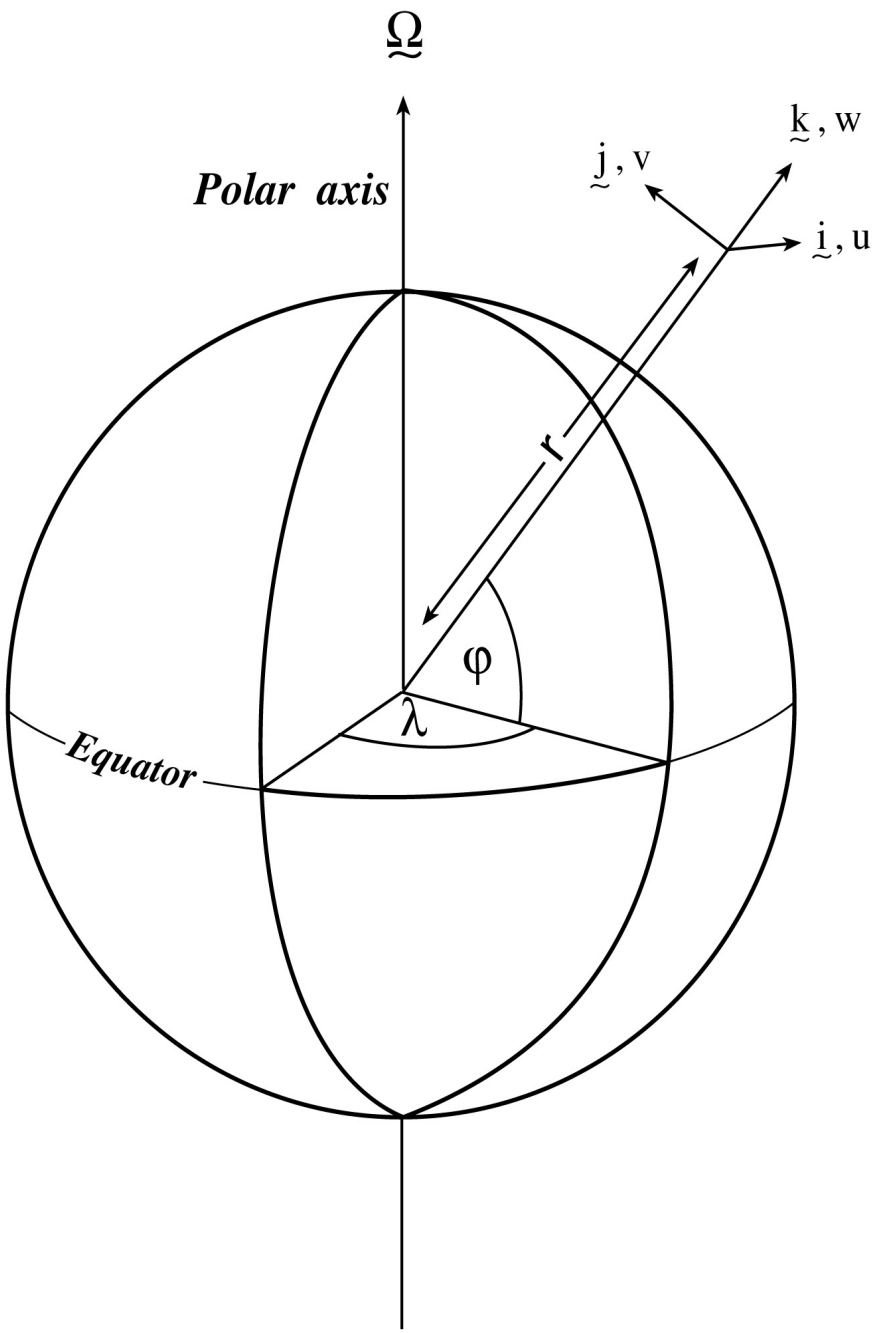


Figure 1.16: Spherical polar coordinates: longitude λ, latitude φ and r the distance from the center.

making a small correction to the hydrostatic pressure.

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

1.3.4.3 Non-hydrostatic and quasi-nonhydrostatic forms

The MIT model presently supports a full non-hydrostatic ocean isomorph, but only a quasi-non-hydrostatic atmospheric isomorph.

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

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

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

where p_{hy} is the hydrostatic pressure.

1.3.4.4 Summary of equation sets supported by model

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

Hydrostatic and quasi-hydrostatic The hydrostatic set is written out in $p-$coordinates in appendix Atmosphere - see eq(1.59).

Quasi-nonhydrostatic A quasi-nonhydrostatic form is also supported.

Ocean

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

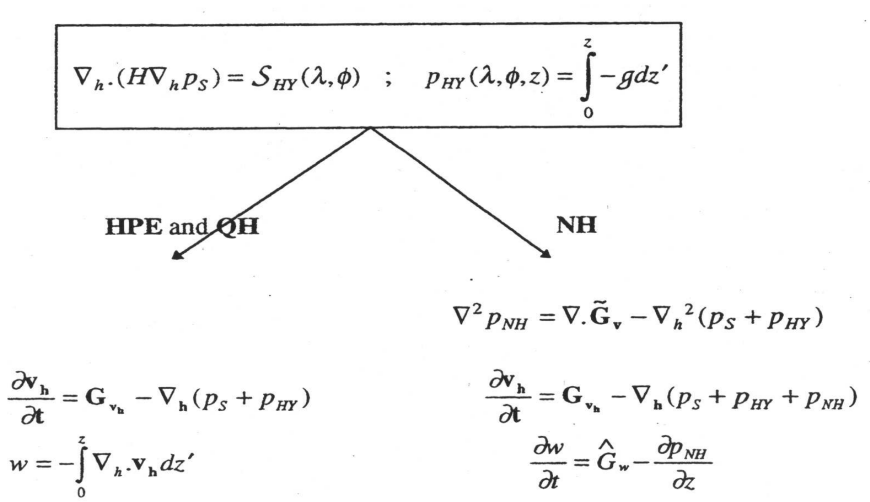


Figure 1.17: Basic solution strategy in MITgcm. **HPE** and **QH** forms diagnose the vertical velocity, in **NH** a prognostic equation for the vertical velocity is integrated.

Non-hydrostatic Non-hydrostatic forms of the incompressible Boussinesq equations in $z-$ coordinates are supported - see eqs(1.99) to (1.104).

1.3.5 Solution strategy

The method of solution employed in the **HPE**, **QH** and **NH** models is summarized in Figure 1.17. Under all dynamics, a 2-d elliptic equation is first solved to find the surface pressure and the hydrostatic pressure at any level computed from the weight of fluid above. Under **HPE** and **QH** dynamics, the horizontal momentum equations are then stepped forward and $\dot{r}$ found from continuity. Under **NH** dynamics a 3-d elliptic equation must be solved for the non-hydrostatic pressure before stepping forward the horizontal momentum equations; $\dot{r}$ is found by stepping forward the vertical momentum equation.

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

1.3.6 Finding the pressure field

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

1.3.6.1 Hydrostatic pressure

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

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

and so

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

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

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

1.3.6.2 Surface pressure

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

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

Thus:

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

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

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

where we have incorporated a source term.

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

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

where b_s is the buoyancy at the surface.

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

1.3.6.3 Non-hydrostatic pressure

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

$$\nabla_3^2 \phi_{nh} = \nabla . \vec{\mathbf{G}}_{\vec{v}} - \left(\nabla_h^2 \phi_s + \nabla^2 \phi_{hyd} \right) = \nabla . \vec{\mathbf{F}} \tag{1.37}$$

For a given rhs this 3-d elliptic equation must be inverted for ϕ_{nh} subject to appropriate choice of boundary conditions. This method is usually called *The Pressure Method* [Harlow and Welch, 1965; Williams, 1969; Potter, 1976]. In the hydrostatic primitive equations case (**HPE**), the 3-d problem does not need to be solved.

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

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

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

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

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

where

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

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

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

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

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

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

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

1.3.7 Forcing/dissipation

1.3.7.1 Forcing

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

1.3.7.2 Dissipation

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

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

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

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

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

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

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

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

1.3.8 Vector invariant form

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

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

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

1.3.9 Adjoint

Tangent linear and adjoint counterparts of the forward model are described in Chapter 5.

1.4 Appendix ATMOSPHERE

1.4.1 Hydrostatic Primitive Equations for the Atmosphere in pressure coordinates

The hydrostatic primitive equations (HPEs) in p-coordinates are:

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

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

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

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

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

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

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

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

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

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

Potential temperature is defined:

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

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

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

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

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

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

The heat equation is obtained by noting that

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

and on substituting into (1.50) gives:

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

which is in conservative form.

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

1.4.1.1 Boundary conditions

The upper and lower boundary conditions are :

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

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

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

1.4.1.2 Splitting the geo-potential

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

$$\theta = \theta_o + \theta' \tag{1.56}$$
$$\alpha = \alpha_o + \alpha' \tag{1.57}$$
$$\phi = \phi_o + \phi' \tag{1.58}$$

The reference state (indicated by subscript "0") corresponds to horizontally homogeneous atmosphere at rest $(\theta_o, \alpha_o, \phi_o)$ with surface pressure $p_o(x, y)$ that satisfies $\phi_o(p_o) = g\ Z_{topo}$, defined:

$$\begin{aligned} \theta_o(p) &= f^n(p) \\ \alpha_o(p) &= \Pi_p \theta_o \\ \phi_o(p) &= \phi_{topo} - \int_{p_0}^{p} \alpha_o dp \end{aligned}$$

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

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \nabla_p \phi' = \vec{\mathcal{F}} \tag{1.59}$$
$$\frac{\partial \phi'}{\partial p} + \alpha' = 0 \tag{1.60}$$
$$\nabla_p \cdot \vec{\mathbf{v}}_h + \frac{\partial \omega}{\partial p} = 0 \tag{1.61}$$
$$\frac{\partial \Pi}{\partial p}\theta' = \alpha' \tag{1.62}$$
$$\frac{D\theta}{Dt} = \frac{\mathcal{Q}}{\Pi} \tag{1.63}$$

1.5 Appendix OCEAN

1.5.1 Equations of motion for the ocean

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

equations for oceanic motion are:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.5.1.1 Compressible z-coordinate equations

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

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

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

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

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

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

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

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

1.5.1.2 'Anelastic' z-coordinate equations

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

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

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

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

If the time- and space-scales of the motions of interest are longer than those of acoustic modes, then $\frac{Dp'}{Dt} << (\frac{Dp_o}{Dt}, \nabla \cdot \vec{\mathbf{v}}_h)$ in the continuity equations and $\frac{\partial \rho}{\partial p}\Big|_{\theta,S}\frac{Dp'}{Dt} << \frac{\partial \rho}{\partial p}\Big|_{\theta,S}\frac{Dp_o}{Dt}$ in the EOS (1.70). Thus we set $\epsilon_s = 0$, removing

the dependency on p' in the continuity equation and EOS. Expanding $\frac{Dp_o(z)}{Dt} = -g\rho_o w$ then leads to the anelastic continuity equation:

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

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

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

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

$$\frac{1}{\rho_o} \frac{\partial \rho_o}{\partial z} = \frac{-g}{c_s^2} \tag{1.86}$$

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

$$\begin{aligned}
\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_o} \nabla_z p &= \vec{\mathcal{F}} && (1.87)\\
\epsilon_{nh} \frac{Dw}{Dt} + \frac{g\rho}{\rho_o} + \frac{1}{\rho_o} \frac{\partial p}{\partial z} &= \epsilon_{nh} \mathcal{F}_w && (1.88)\\
\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{1}{\rho_o} \frac{\partial (\rho_o w)}{\partial z} &= 0 && (1.89)\\
\rho &= \rho(\theta, S, p_o(z)) && (1.90)\\
\frac{D\theta}{Dt} &= \mathcal{Q}_\theta && (1.91)\\
\frac{DS}{Dt} &= \mathcal{Q}_s && (1.92)
\end{aligned}$$

1.5.1.3 Incompressible z-coordinate equations

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

pressible Boussinesq equations:

$$\begin{aligned}
\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_c}\nabla_z p &= \vec{\mathcal{F}} && (1.93)\\
\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho}{\rho_c} + \frac{1}{\rho_c}\frac{\partial p}{\partial z} &= \epsilon_{nh}\mathcal{F}_w && (1.94)\\
\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} &= 0 && (1.95)\\
\rho &= \rho(\theta, S) && (1.96)\\
\frac{D\theta}{Dt} &= \mathcal{Q}_\theta && (1.97)\\
\frac{DS}{Dt} &= \mathcal{Q}_s && (1.98)
\end{aligned}$$

where ρ_c is a constant reference density of water.

1.5.1.4 Compressible non-divergent equations

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

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

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

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

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

$$\begin{aligned}
\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_c}\nabla_z p' &= \vec{\mathcal{F}} && (1.99)\\
\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho'}{\rho_c} + \frac{1}{\rho_c}\frac{\partial p'}{\partial z} &= \epsilon_{nh}\mathcal{F}_w && (1.100)\\
\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} &= 0 && (1.101)\\
\rho' &= \rho(\theta, S, p_o(z)) - \rho_c && (1.102)\\
\frac{D\theta}{Dt} &= \mathcal{Q}_\theta && (1.103)\\
\frac{DS}{Dt} &= \mathcal{Q}_s && (1.104)
\end{aligned}$$

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

Though necessary, the assumptions that go into these equations are messy since we essentially assume a different EOS for the reference density and the

perturbation density. Nevertheless, it is the hydrostatic ($\epsilon_{nh} = 0$ form of these equations that are used throughout the ocean modeling community and referred to as the primitive equations (HPE).

1.6 Appendix:OPERATORS

1.6.1 Coordinate systems

1.6.1.1 Spherical coordinates

In spherical coordinates, the velocity components in the zonal, meridional and vertical direction respectively, are given by (see Fig.2) :

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

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

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

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

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

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

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

Chapter 2

Discretization and Algorithm

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

2.1 Time-stepping

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

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

1. the semi-implicit pressure method for hydrostatic equations with a rigid-lid, variables co-located in time and with Adams-Bashforth time-stepping,

2. as 1. but with an implicit linear free-surface,

3. as 1. or 2. but with variables staggered in time,

4. as 1. or 2. but with non-hydrostatic terms included,

5. as 2. or 3. but with non-linear free-surface.

In all the above configurations it is also possible to substitute the Adams-Bashforth with an alternative time-stepping scheme for terms evaluated explicitly in time. Since the over-arching algorithm is independent of the particular time-stepping scheme chosen we will describe first the over-arching algorithm, known as the pressure method, with a rigid-lid model in section 2.2. This algorithm is essentially unchanged, apart for some coefficients, when the rigid lid assumption is replaced with a linearized implicit free-surface, described in section 2.3. These two flavors of the pressure-method encompass all formulations of the model as it exists today. The integration of explicit in time terms is outlined in section 2.4 and put into the context of the overall algorithm in sections 2.6 and 2.7. Inclusion of non-hydrostatic terms requires applying the pressure method in three dimensions instead of two and this algorithm modification is described in section 2.8. Finally, the free-surface equation may be treated more exactly, including non-linear terms, and this is described in section 2.9.2.

2.2 Pressure method with rigid-lid

The horizontal momentum and continuity equations for the ocean (1.99 and 1.101), or for the atmosphere (1.45 and 1.47), can be summarized by:

$$\partial_t u + g\partial_x \eta = G_u \tag{2.1}$$
$$\partial_t v + g\partial_y \eta = G_v \tag{2.2}$$
$$\partial_x u + \partial_y v + \partial_z w = 0 \tag{2.3}$$

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

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

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

$$u^{n+1} + \Delta t g \partial_x \eta^{n+1} = u^n + \Delta t G_u^{(n+1/2)} \tag{2.5}$$
$$v^{n+1} + \Delta t g \partial_y \eta^{n+1} = v^n + \Delta t G_v^{(n+1/2)} \tag{2.6}$$
$$\partial_x H\widehat{u^{n+1}} + \partial_y H\widehat{v^{n+1}} = 0 \tag{2.7}$$

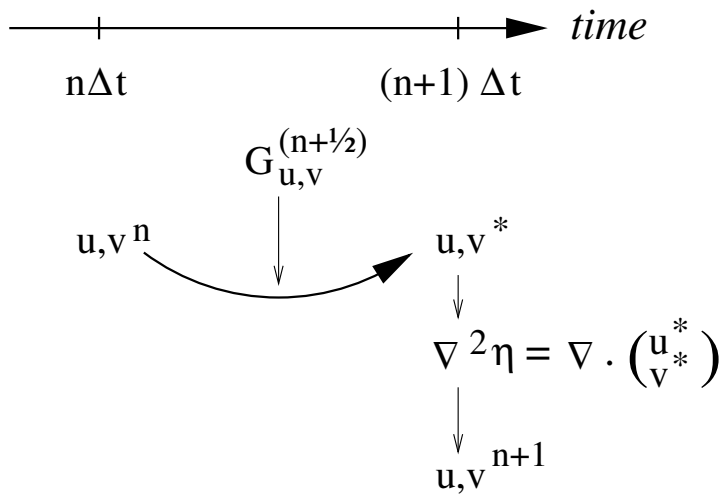


Figure 2.1: A schematic of the evolution in time of the pressure method algorithm. A prediction for the flow variables at time level $n+1$ is made based only on the explicit terms, $G^{(n+^1/_2)}$, and denoted u^*, v^*. Next, a pressure field is found such that u^{n+1}, v^{n+1} will be non-divergent. Conceptually, the $*$ quantities exist at time level $n+1$ but they are intermediate and only temporary.

FORWARD_STEP	
DYNAMICS	
TIMESTEP	u^*,v^* (2.8,2.9)
SOLVE_FOR_PRESSURE	
CALC_DIV_GHAT	$H\widehat{u^*}$,$H\widehat{v^*}$ (2.10)
CG2D	η^{n+1} (2.10)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla\eta^{n+1}$
CORRECTION_STEP	u^{n+1},v^{n+1} (2.11,2.12)

Figure 2.2: Calling tree for the pressure method algorithm (FORWARD_STEP)

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

Substituting the two momentum equations into the depth integrated continuity equation eliminates u^{n+1} and v^{n+1} yielding an elliptic equation for η^{n+1}. Equations 2.5, 2.6 and 2.7 can then be re-arranged as follows:

$$
\begin{aligned}
u^* &= u^n + \Delta t G_u^{(n+1/2)} && (2.8)\\
v^* &= v^n + \Delta t G_v^{(n+1/2)} && (2.9)\\
\partial_x \Delta t g H \partial_x \eta^{n+1} + \partial_y \Delta t g H \partial_y \eta^{n+1} &= \partial_x H\widehat{u^*} + \partial_y H\widehat{v^*} && (2.10)\\
u^{n+1} &= u^* - \Delta t g \partial_x \eta^{n+1} && (2.11)\\
v^{n+1} &= v^* - \Delta t g \partial_y \eta^{n+1} && (2.12)
\end{aligned}
$$

Equations 2.8 to 2.12, solved sequentially, represent the pressure method algorithm used in the model. The essence of the pressure method lies in the fact that any explicit prediction for the flow would lead to a divergence flow field so a pressure field must be found that keeps the flow non-divergent over each step of the integration. The particular location in time of the pressure field is somewhat ambiguous; in Fig. 2.1 we depicted as co-located with the future flow field (time level $n+1$) but it could equally have been drawn as staggered in time with the flow.

The correspondence to the code is as follows:

- the prognostic phase, equations 2.8 and 2.9, stepping forward u^n and v^n to u^* and v^* is coded in `TIMESTEP()`
- the vertical integration, $H\widehat{u^*}$ and $H\widehat{v^*}$, divergence and inversion of the elliptic operator in equation 2.10 is coded in `SOLVE_FOR_PRESSURE()`
- finally, the new flow field at time level $n+1$ given by equations 2.11 and 2.12 is calculated in `CORRECTION_STEP()`.

The calling tree for these routines is given in Fig. 2.2.

Need to discuss implicit viscosity somewhere:

$$
\begin{aligned}
\frac{1}{\Delta t}u^{n+1} - \partial_z A_v \partial_z u^{n+1} + g\partial_x \eta^{n+1} &= \frac{1}{\Delta t}u^n + G_u^{(n+1/2)} && (2.13)\\
\frac{1}{\Delta t}v^{n+1} - \partial_z A_v \partial_z v^{n+1} + g\partial_y \eta^{n+1} &= \frac{1}{\Delta t}v^n + G_v^{(n+1/2)} && (2.14)
\end{aligned}
$$

2.3 Pressure method with implicit linear free-surface

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

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

$$\partial_t \eta + \partial_x H\widehat{u} + \partial_y H\widehat{v} = P - E + R \tag{2.15}$$

which differs from the depth integrated continuity equation with rigid-lid (2.4) by the time-dependent term and fresh-water source term.

Equation 2.7 in the rigid-lid pressure method is then replaced by the time discretization of 2.15 which is:

$$\eta^{n+1} + \Delta t \partial_x H\widehat{u^{n+1}} + \Delta t \partial_y H\widehat{v^{n+1}} = \eta^n + \Delta t(P - E) \tag{2.16}$$

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

As for the rigid-lid pressure method, equations 2.5, 2.6 and 2.16 can be re-arranged as follows:

$$\begin{aligned}
u^* &= u^n + \Delta t G_u^{(n+1/2)} && (2.17)\\
v^* &= v^n + \Delta t G_v^{(n+1/2)} && (2.18)\\
\eta^* &= \epsilon_{fs}\left(\eta^n + \Delta t(P-E)\right) - \Delta t \partial_x H\widehat{u^*} + \partial_y H\widehat{v^*} && (2.19)\\
\partial_x g H \partial_x \eta^{n+1} &+ \partial_y g H \partial_y \eta^{n+1} - \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} && (2.20)\\
u^{n+1} &= u^* - \Delta t g \partial_x \eta^{n+1} && (2.21)\\
v^{n+1} &= v^* - \Delta t g \partial_y \eta^{n+1} && (2.22)
\end{aligned}$$

Equations 2.17 to 2.22, solved sequentially, represent the pressure method algorithm with a backward implicit, linearized free surface. The method is still formerly a pressure method because in the limit of large Δt the rigid-lid method is recovered. However, the implicit treatment of the free-surface allows the flow to be divergent and for the surface pressure/elevation to respond on a finite time-scale (as opposed to instantly). To recover the rigid-lid formulation, we introduced a switch-like parameter, ϵ_{fs}, which selects between the free-surface and rigid-lid; $\epsilon_{fs} = 1$ allows the free-surface to evolve; $\epsilon_{fs} = 0$ imposes the rigid-lid. The evolution in time and location of variables is exactly as it was for the rigid-lid model so that Fig. 2.1 is still applicable. Similarly, the calling sequence, given in Fig. 2.2, is as for the pressure-method.

FORWARD_STEP		
THERMODYNAMICS		
CALC_GT		
	GAD_CALC_RHS	$G_\theta^n = G_\theta(u, \theta^n)$
either	EXTERNAL_FORCING	$G_\theta^n = G_\theta^n + \mathcal{Q}$
	ADAMS_BASHFORTH2	$G_\theta^{(n+1/2)}$ (2.24)
or	EXTERNAL_FORCING	$G_\theta^{(n+1/2)} = G_\theta^{(n+1/2)} + \mathcal{Q}$
TIMESTEP_TRACER		τ^* (2.23)
IMPLDIFF		$\tau^{(n+1)}$ (2.27)

Figure 2.3: Calling tree for the Adams-Bashforth time-stepping of temperature with implicit diffusion.

2.4 Explicit time-stepping: Adams-Bashforth

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

In the previous sections, we summarized an explicit scheme as:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.23}$$

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

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

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

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

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

AJA needs to find his notes on this...
...and for this too.

2.5 Implicit time-stepping: backward method

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

$$\tau^{n+1} - \Delta t \partial_r \kappa_v \partial_r \tau^{n+1} = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.25}$$

where $G_\tau^{(n+1/2)}$ is the remaining explicit terms extrapolated using the Adams-Bashforth method as described above. Equation 2.25 can be split split into:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.26}$$
$$\tau^{n+1} = \mathcal{L}_\tau^{-1}(\tau^*) \tag{2.27}$$

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

$$\mathcal{L} = [1 + \Delta t \partial_r \kappa_v \partial_r] \tag{2.28}$$

Equation 2.26 looks exactly as 2.23 while 2.27 involves an operator or matrix inversion. By re-arranging 2.25 in this way we have cast the method as an explicit prediction step and an implicit step allowing the latter to be inserted into the over all algorithm with minimal interference.

Fig. 2.3 shows the calling sequence for stepping forward a tracer variable such as temperature.

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

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

The Adams-Bashforth extrapolation of explicit tendencies fits neatly into the pressure method algorithm when all state variables are co-located in time. Fig. 2.4 illustrates the location of variables in time and the evolution of the

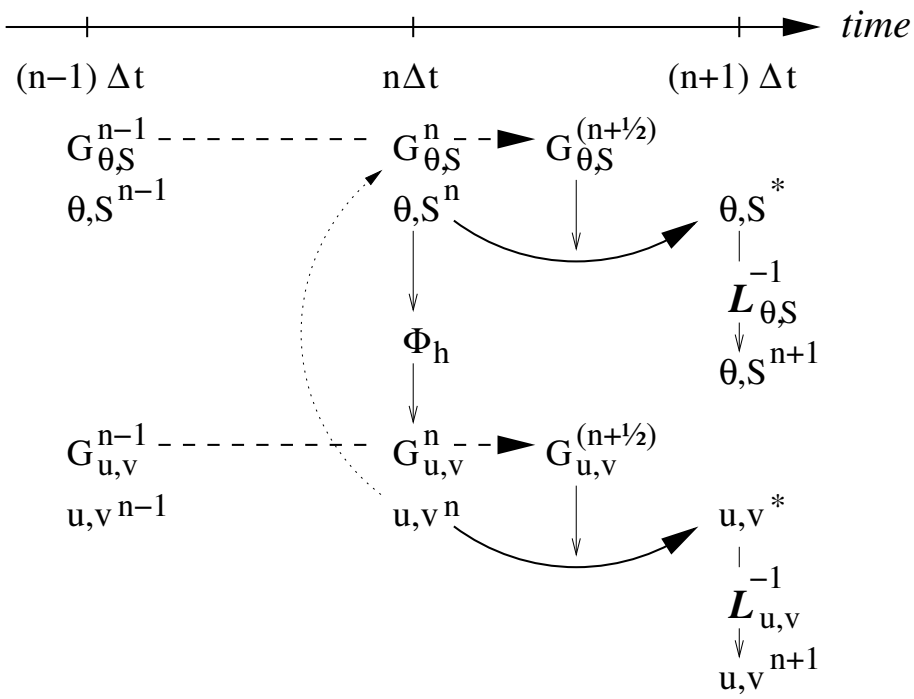


Figure 2.4: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm. All prognostic variables are co-located in time. Explicit tendencies are evaluated at time level n as a function of the state at that time level (dotted arrow). The explicit tendency from the previous time level, $n-1$, is used to extrapolate tendencies to $n+1/2$ (dashed arrow). This extrapolated tendency allows variables to be stably integrated forward-in-time to render an estimate ($*$-variables) at the $n+1$ time level (solid arc-arrow). The operator $\mathcal{L}$ formed from implicit-in-time terms is solved to yield the state variables at time level $n+1$.

FORWARD_STEP	
EXTERNAL_FIELDS_LOAD	
DO_ATMOSPHERIC_PHYS	
DO_OCEANIC_PHYS	
THERMODYNAMICS	
CALC_GT	
GAD_CALC_RHS	$G_\theta^n = G_\theta(u, \theta^n)$ (2.29)
EXTERNAL_FORCING	$G_\theta^n = G_\theta^n + \mathcal{Q}$
ADAMS_BASHFORTH2	$G_\theta^{(n+1/2)}$ (2.30)
TIMESTEP_TRACER	θ^* (2.31)
IMPLDIFF	$\theta^{(n+1)}$ (2.32)
DYNAMICS	
CALC_PHI_HYD	ϕ_{hyd}^n (2.33)
MOM_FLUXFORM or MOM_VECINV	$G_{\vec{\mathbf{v}}}^n$ (2.34)
TIMESTEP	$\vec{\mathbf{v}}^*$ (2.35, 2.36)
IMPLDIFF	$\vec{\mathbf{v}}^{**}$ (2.37)
UPDATE_R_STAR or UPDATE_SURF_DR	(NonLin-FS only)
SOLVE_FOR_PRESSURE	
CALC_DIV_GHAT	η^* (2.38)
CG2D	η^{n+1} (2.39)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla\eta^{n+1}$
CORRECTION_STEP	u^{n+1},v^{n+1} (2.40)
TRACERS_CORRECTION_STEP	
CYCLE_TRACER	θ^{n+1}
FILTER	Shapiro Filter, Zonal Filter (FFT)
CONVECTIVE_ADJUSTMENT	

Figure 2.5: Calling tree for the overall synchronous algorithm using Adams-Bashforth time-stepping. The place where the model geometry (**hFac** factors) is updated is added here but is only relevant for the non-linear free-surface algorithm. For completeness, the external forcing, ocean and atmospheric physics have been added, although they are mainly optional

algorithm with time. The algorithm can be represented by the sequential solution of the follow equations:

$$
\begin{aligned}
G^n_{\theta,S} &= G_{\theta,S}(u^n, \theta^n, S^n) && (2.29)\\
G^{(n+1/2)}_{\theta,S} &= (3/2+\epsilon_{AB})G^n_{\theta,S} - (1/2+\epsilon_{AB})G^{n-1}_{\theta,S} && (2.30)\\
(\theta^*, S^*) &= (\theta^n, S^n) + \Delta t G^{(n+1/2)}_{\theta,S} && (2.31)\\
(\theta^{n+1}, S^{n+1}) &= \mathcal{L}^{-1}_{\theta,S}(\theta^*, S^*) && (2.32)\\
\phi^n_{hyd} &= \int b(\theta^n, S^n) dr && (2.33)\\
\vec{\mathbf{G}}^n_{\vec{\mathbf{v}}} &= \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^n, \phi^n_{hyd}) && (2.34)\\
\vec{\mathbf{G}}^{(n+1/2)}_{\vec{\mathbf{v}}} &= (3/2+\epsilon_{AB})\vec{\mathbf{G}}^n_{\vec{\mathbf{v}}} - (1/2+\epsilon_{AB})\vec{\mathbf{G}}^{n-1}_{\vec{\mathbf{v}}} && (2.35)\\
\vec{\mathbf{v}}^* &= \vec{\mathbf{v}}^n + \Delta t \vec{\mathbf{G}}^{(n+1/2)}_{\vec{\mathbf{v}}} && (2.36)\\
\vec{\mathbf{v}}^{**} &= \mathcal{L}^{-1}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^*) && (2.37)\\
\eta^* &= \epsilon_{fs}\left(\eta^n + \Delta t(P-E)\right) - \Delta t \nabla \cdot H\widehat{\vec{\mathbf{v}}^{**}} && (2.38)\\
\nabla \cdot gH\nabla\eta^{n+1} &- \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} && (2.39)\\
\vec{\mathbf{v}}^{n+1} &= \vec{\mathbf{v}}^* - \Delta t g \nabla \eta^{n+1} && (2.40)
\end{aligned}
$$

Fig. 2.4 illustrates the location of variables in time and evolution of the algorithm with time. The Adams-Bashforth extrapolation of the tracer tendencies is illustrated by the dashed arrow, the prediction at $n+1$ is indicated by the solid arc. Inversion of the implicit terms, $\mathcal{L}^{-1}_{\theta,S}$, then yields the new tracer fields at $n+1$. All these operations are carried out in subroutine *THERMODYNAMICS* an subsidiaries, which correspond to equations 2.29 to 2.32. Similarly illustrated is the Adams-Bashforth extrapolation of accelerations, stepping forward and solving of implicit viscosity and surface pressure gradient terms, corresponding to equations 2.34 to 2.40. These operations are carried out in subroutines *DYNAMCIS*, *SOLVE_FOR_PRESSURE* and *MOMENTUM_CORRECTION_STEP*. This, then, represents an entire algorithm for stepping forward the model one time-step. The corresponding calling tree is given in 2.5.

2.7 Staggered baroclinic time-stepping

For well stratified problems, internal gravity waves may be the limiting process for determining a stable time-step. In the circumstance, it is more efficient to stagger in time the thermodynamic variables with the flow variables. Fig. 2.6 illustrates the staggering and algorithm. The key difference between this and Fig. 2.4 is that the thermodynamic variables are solved after the dynamics, using the recently updated flow field. This essentially allows the gravity wave terms to leap-frog in time giving second order accuracy and more stability.

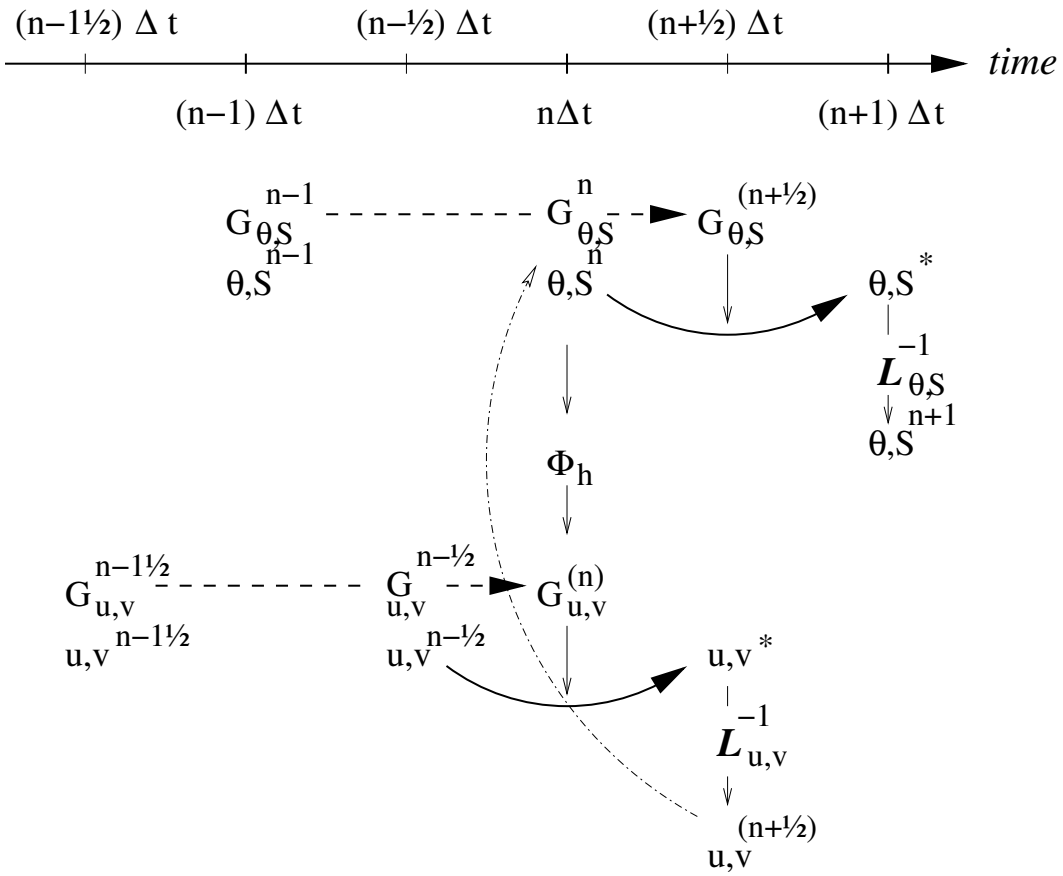


Figure 2.6: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm but with staggering in time of thermodynamic variables with the flow. Explicit momentum tendencies are evaluated at time level $n - 1/2$ as a function of the flow field at that time level $n - 1/2$. The explicit tendency from the previous time level, $n - 3/2$, is used to extrapolate tendencies to n (dashed arrow). The hydrostatic pressure/geo-potential ϕ_{hyd} is evaluated directly at time level n (vertical arrows) and used with the extrapolated tendencies to step forward the flow variables from $n-1/2$ to $n+1/2$ (solid arc-arrow). The implicit-in-time operator $\mathcal{L}_{\mathbf{u},\mathbf{v}}$ (vertical arrows) is then applied to the previous estimation of the the flow field ($*$-variables) and yields to the two velocity components u, v at time level $n+1/2$. These are then used to calculate the advection term (dashed arc-arrow) of the thermo-dynamics tendencies at time step n. The extrapolated thermodynamics tendency, from time level $n-1$ and n to $n+1/2$, allows thermodynamic variables to be stably integrated forward-in-time (solid arc-arrow) up to time level $n+1$.

The essential change in the staggered algorithm is that the thermodynamics solver is delayed from half a time step, allowing the use of the most recent velocities to compute the advection terms. Once the thermodynamics fields are updated, the hydrostatic pressure is computed to step forwrad the dynamics. Note that the pressure gradient must also be taken out of the Adams-Bashforth extrapolation. Also, retaining the integer time-levels, n and $n+1$, does not give a user the sense of where variables are located in time. Instead, we re-write the entire algorithm, 2.29 to 2.40, annotating the position in time of variables appropriately:

$$\phi^n_{hyd} = \int b(\theta^n, S^n)dr \tag{2.41}$$

$$\vec{\mathbf{G}}^{n-1/2}_{\vec{\mathbf{v}}} = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^{n-1/2}) \tag{2.42}$$

$$\vec{\mathbf{G}}^{(n)}_{\vec{\mathbf{v}}} = (3/2+\epsilon_{AB})\vec{\mathbf{G}}^{n-1/2}_{\vec{\mathbf{v}}} - (1/2+\epsilon_{AB})\vec{\mathbf{G}}^{n-3/2}_{\vec{\mathbf{v}}} \tag{2.43}$$

$$\vec{\mathbf{v}}^{*} = \vec{\mathbf{v}}^{n-1/2} + \Delta t\left(\vec{\mathbf{G}}^{(n)}_{\vec{\mathbf{v}}} - \nabla\phi^n_{hyd}\right) \tag{2.44}$$

$$\vec{\mathbf{v}}^{**} = \mathcal{L}^{-1}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^{*}) \tag{2.45}$$

$$\eta^{*} = \epsilon_{fs}\left(\eta^{n-1/2} + \Delta t(P-E)^n\right) - \Delta t\nabla\cdot H\widehat{\vec{\mathbf{v}}^{**}} \tag{2.46}$$

$$\nabla\cdot gH\nabla\eta^{n+1/2} - \frac{\epsilon_{fs}\eta^{n+1/2}}{\Delta t^2} = -\frac{\eta^{*}}{\Delta t^2} \tag{2.47}$$

$$\vec{\mathbf{v}}^{n+1/2} = \vec{\mathbf{v}}^{*} - \Delta t g\nabla\eta^{n+1/2} \tag{2.48}$$

$$G^n_{\theta,S} = G_{\theta,S}(u^{n+1/2}, \theta^n, S^n) \tag{2.49}$$

$$G^{(n+1/2)}_{\theta,S} = (3/2+\epsilon_{AB})G^n_{\theta,S} - (1/2+\epsilon_{AB})G^{n-1}_{\theta,S} \tag{2.50}$$

$$(\theta^{*}, S^{*}) = (\theta^n, S^n) + \Delta t G^{(n+1/2)}_{\theta,S} \tag{2.51}$$

$$(\theta^{n+1}, S^{n+1}) = \mathcal{L}^{-1}_{\theta,S}(\theta^{*}, S^{*}) \tag{2.52}$$

$$\tag{2.53}$$

The corresponding calling tree is given in 2.7. The staggered algorithm is activated with the run-time flag **staggerTimeStep**=*.TRUE.* in parameter file *data*, namelist *PARM01*.

The only difficulty with this approach is apparent in equation 2.49 and illustrated by the dotted arrow connecting $u, v^{n+1/2}$ with G^n_θ. The flow used to advect tracers around is not naturally located in time. This could be avoided by applying the Adams-Bashforth extrapolation to the tracer field itself and advecting that around but this approach is not yet available. We're not aware of any detrimental effect of this feature. The difficulty lies mainly in interpretation of what time-level variables and terms correspond to.

2.8 Non-hydrostatic formulation

The non-hydrostatic formulation re-introduces the full vertical momentum equation and requires the solution of a 3-D elliptic equations for non-hydrostatic

FORWARD_STEP	
EXTERNAL_FIELDS_LOAD	
DO_ATMOSPHERIC_PHYS	
DO_OCEANIC_PHYS	
DYNAMICS	
CALC_PHI_HYD	ϕ^n_{hyd} (2.41)
MOM_FLUXFORM or MOM_VECINV	$G^{n-1/2}_{\vec{\mathbf{v}}}$ (2.42)
TIMESTEP	$\vec{\mathbf{v}}^*$ (2.43, 2.44)
IMPLDIFF	$\vec{\mathbf{v}}^{**}$ (2.45)
UPDATE_R_STAR or UPDATE_SURF_DR	(NonLin-FS only)
SOLVE_FOR_PRESSURE	
CALC_DIV_GHAT	η^* (2.46)
CG2D	$\eta^{n+1/2}$ (2.47)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla\eta^{n+1/2}$
CORRECTION_STEP	$u^{n+1/2}$,$v^{n+1/2}$ (2.48)
THERMODYNAMICS	
CALC_GT	
GAD_CALC_RHS	$G^n_\theta = G_\theta(u,\theta^n)$ (2.49)
EXTERNAL_FORCING	$G^n_\theta = G^n_\theta + \mathcal{Q}$
ADAMS_BASHFORTH2	$G^{(n+1/2)}_\theta$ (2.50)
TIMESTEP_TRACER	θ^* (2.51)
IMPLDIFF	$\theta^{(n+1)}$ (2.52)
TRACERS_CORRECTION_STEP	
CYCLE_TRACER	θ^{n+1}
FILTER	Shapiro Filter, Zonal Filter (FFT)
CONVECTIVE_ADJUSTMENT	

Figure 2.7: Calling tree for the overall staggered algorithm using Adams-Bashforth time-stepping. The place where the model geometry (**hFac** factors) is updated is added here but is only relevant for the non-linear free-surface algorithm.

pressure perturbation. We still intergrate vertically for the hydrostatic pressure and solve a 2-D elliptic equation for the surface pressure/elevation for this reduces the amount of work needed to solve for the non-hydrostatic pressure.

The momentum equations are discretized in time as follows:

$$\begin{aligned}
\frac{1}{\Delta t}u^{n+1} + g\partial_x\eta^{n+1} + \partial_x\phi_{nh}^{n+1} &= \frac{1}{\Delta t}u^n + G_u^{(n+1/2)} && (2.54)\\
\frac{1}{\Delta t}v^{n+1} + g\partial_y\eta^{n+1} + \partial_y\phi_{nh}^{n+1} &= \frac{1}{\Delta t}v^n + G_v^{(n+1/2)} && (2.55)\\
\frac{1}{\Delta t}w^{n+1} + \partial_r\phi_{nh}^{n+1} &= \frac{1}{\Delta t}w^n + G_w^{(n+1/2)} && (2.56)\\
& && (2.57)
\end{aligned}$$

which must satisfy the discrete-in-time depth integrated continuity, equation 2.16 and the local continuity equation

$$\partial_x u^{n+1} + \partial_y v^{n+1} + \partial_r w^{n+1} = 0 \tag{2.58}$$

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

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

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

$$u^{n+1} = u^{**} - \Delta t\partial_x\phi_{nh}^{n+1} \qquad u^{**} = u^* - \Delta t g\partial_x\eta^{n+1} \tag{2.59}$$
$$v^{n+1} = v^{**} - \Delta t\partial_y\phi_{nh}^{n+1} \qquad v^{**} = v^* - \Delta t g\partial_y\eta^{n+1} \tag{2.60}$$

Substituting into the depth integrated continuity (equation 2.16) gives

$$\partial_x H\partial_x\left(g\eta^{n+1} + \widehat{\phi}_{nh}^{n+1}\right) + \partial_y H\partial_y\left(g\eta^{n+1} + \widehat{\phi}_{nh}^{n+1}\right) - \frac{\epsilon_{fs}\eta^*}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \tag{2.61}$$

which is approximated by equation 2.20 on the basis that i) ϕ_{nh}^{n+1} is not yet known and ii) $\nabla\widehat{\phi}_{nh} << g\nabla\eta$. If 2.20 is solved accurately then the implication is that $\widehat{\phi}_{nh} \approx 0$ so that thet non-hydrostatic pressure field does not drive barotropic motion.

The flow must satisfy non-divergence (equation 2.58) locally, as well as depth integrated, and this constraint is used to form a 3-D elliptic equations for ϕ_{nh}^{n+1}:

$$\partial_{xx}\phi_{nh}^{n+1} + \partial_{yy}\phi_{nh}^{n+1} + \partial_{rr}\phi_{nh}^{n+1} = \partial_x u^{**} + \partial_y v^{**} + \partial_r w^* \tag{2.62}$$

The entire algorithm can be summarized as the sequential solution of the

following equations:

$$
\begin{aligned}
u^* &= u^n + \Delta t G_u^{(n+1/2)} && (2.63)\\
v^* &= v^n + \Delta t G_v^{(n+1/2)} && (2.64)\\
w^* &= w^n + \Delta t G_w^{(n+1/2)} && (2.65)\\
\eta^* &= \epsilon_{fs}\left(\eta^n + \Delta t(P-E)\right) - \Delta t \partial_x H\widehat{u^*} + \partial_y H\widehat{v^*} && (2.66)\\
\partial_x g H \partial_x \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} &- \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} && (2.67)\\
u^{**} &= u^* - \Delta t g \partial_x \eta^{n+1} && (2.68)\\
v^{**} &= v^* - \Delta t g \partial_y \eta^{n+1} && (2.69)\\
\partial_{xx}\phi_{nh}^{n+1} + \partial_{yy}\phi_{nh}^{n+1} + \partial_{rr}\phi_{nh}^{n+1} &= \partial_x u^{**} + \partial_y v^{**} + \partial_r w^* && (2.70)\\
u^{n+1} &= u^{**} - \Delta t \partial_x \phi_{nh}^{n+1} && (2.71)\\
v^{n+1} &= v^{**} - \Delta t \partial_y \phi_{nh}^{n+1} && (2.72)\\
\partial_r w^{n+1} &= -\partial_x u^{n+1} - \partial_y v^{n+1} && (2.73)
\end{aligned}
$$

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

2.9 Variants on the Free Surface

We now describe the various formulations of the free-surface that include non-linear forms, implicit in time using Crank-Nicholson, explicit and [one day] split-explicit. First, we'll reiterate the underlying algorithm but this time using the notation consistent with the more general vertical coordinate r. The elliptic equation for free-surface coordinate (units of r), corresponding to 2.16, and assuming no non-hydrostatic effects ($\epsilon_{nh} = 0$) is:

$$\epsilon_{fs}\eta^{n+1} - \nabla_h \cdot \Delta t^2 (R_o - R_{fixed}) \nabla_h b_s \eta^{n+1} = \eta^* \tag{2.74}$$

where

$$\eta^* = \epsilon_{fs}\,\eta^n - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o} \vec{\mathbf{v}}^* dr \;+\; \epsilon_{fw}\Delta_t (P-E)^n \tag{2.75}$$

S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)
u^*: **GuNm1** (*DYNVARS.h*)
v^*: **GvNm1** (*DYNVARS.h*)
η^*: **cg2d_b** (*SOLVE_FOR_PRESSURE.h)*
η^{n+1}*:* **etaN** *(*DYNVARS.h)

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

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

This is known as the correction step. However, when the model is non-hydrostatic ($\epsilon_{nh} = 1$) we need an additional step and an additional equation for

ϕ'_{nh}. This is obtained by substituting 2.54, 2.55 and 2.56 into continuity:

$$\left[\nabla_h^2 + \partial_{rr}\right] {\phi'_{nh}}^{n+1} = \frac{1}{\Delta t}\left(\nabla_h \cdot \vec{\mathbf{v}}^{**} + \partial_r \dot{r}^*\right) \tag{2.76}$$

where

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

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

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

$$\vec{\mathbf{v}}^{n+1} = \vec{\mathbf{v}}^{**} - \epsilon_{nh} \Delta t \nabla_h {\phi'_{nh}}^{n+1} \tag{2.77}$$

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

S/R CORRECTION_STEP (*correction_step.F*)
η^{n+1}: **etaN** (*DYNVARS.h)*
ϕ_{nh}^{n+1}*:* **phi_nh** *(*DYNVARS.h)
u^*: **GuNm1** (*DYNVARS.h*)
v^*: **GvNm1** (*DYNVARS.h*)
u^{n+1}: **uVel** (*DYNVARS.h*)
v^{n+1}: **vVel** (*DYNVARS.h*)

Regarding the implementation of the surface pressure solver, all computation are done within the routine *SOLVE_FOR_PRESSURE* and its dependent calls. The standard method to solve the 2D elliptic problem (2.74) uses the conjugate gradient method (routine *CG2D*); the solver matrix and conjugate gradient operator are only function of the discretized domain and are therefore evaluated separately, before the time iteration loop, within *INI_CG2D*. The computation of the RHS η^* is partly done in *CALC_DIV_GHAT* and in *SOLVE_FOR_PRESSURE*.

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

2.9.1 Crank-Nickelson barotropic time stepping

The full implicit time stepping described previously is unconditionally stable but damps the fast gravity waves, resulting in a loss of potential energy. The modification presented now allows one to combine an implicit part (β, γ) and an explicit part $(1-\beta, 1-\gamma)$ for the surface pressure gradient (β) and for the barotropic flow divergence (γ).
For instance, $\beta = \gamma = 1$ is the previous fully implicit scheme; $\beta = \gamma = 1/2$ is the non damping (energy conserving), unconditionally stable, Crank-Nickelson scheme; $(\beta, \gamma) = (1, 0)$ or $= (0, 1)$ corresponds to the forward - backward scheme

that conserves energy but is only stable for small time steps.
In the code, β, γ are defined as parameters, respectively **implicSurfPress**, **implicDiv2DFlow**. They are read from the main parameter file "*data*" and are set by default to 1,1.

Equations 2.17 – 2.22 are modified as follows:

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

$$\epsilon_{fs}\frac{\eta^{n+1}-\eta^n}{\Delta t} + \nabla_h \cdot \int_{R_{fixed}}^{R_o} [\gamma\vec{\mathbf{v}}^{n+1} + (1-\gamma)\vec{\mathbf{v}}^n]dr = \epsilon_{fw}(P-E) \qquad (2.78)$$

where:

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

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

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

and then to compute (*CORRECTION_STEP*):

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

Notes:

1. The RHS term of equation 2.78 corresponds the contribution of fresh water flux (P-E) to the free-surface variations ($\epsilon_{fw} = 1$, **useRealFreshWater**=*TRUE* in parameter file *data*). In order to remain consistent with the tracer equation, specially in the non-linear free-surface formulation, this term is also affected by the Crank-Nickelson time stepping. The RHS reads: $\epsilon_{fw}(\gamma(P-E)^{n+1/2} + (1-\gamma)(P-E)^{n-1/2})$

2. The non-hydrostatic part of the code has not yet been updated, and therefore cannot be used with $(\beta, \gamma) \neq (1, 1)$.

3. The stability criteria with Crank-Nickelson time stepping for the pure linear gravity wave problem in cartesian coordinates is:

   - $\beta + \gamma < 1$: unstable
   - $\beta \geq 1/2$ and $\gamma \geq 1/2$: stable
   - $\beta + \gamma \geq 1$: stable if

$$c_{max}^2(\beta - 1/2)(\gamma - 1/2) + 1 \geq 0$$

$$\text{with} \quad c_{max} = 2\Delta t\sqrt{gH}\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}}$$

2.9.2 Non-linear free-surface

Recently, new options have been added to the model that concern the free surface formulation.

2.9.2.1 pressure/geo-potential and free surface

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

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

we get:

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

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

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

$$\text{and re write :} \qquad \phi' = \int_{R_o}^{r_{surf}} b \; dr \quad + \quad \int_{r}^{R_o} (b - b_o) dr \tag{2.79}$$

$$\text{or :} \qquad \phi' = \int_{R_o}^{r_{surf}} b_o dr \quad + \quad \int_{r}^{r_{surf}} (b - b_o) dr \tag{2.80}$$

In section 1.3.6, following eq.2.79, the pressure/geo-potential ϕ' has been separated into surface (ϕ_s), and hydrostatic anomaly (ϕ'_{hyd}). In this section, the split between ϕ_s and ϕ'_{hyd} is made according to equation 2.80. This slightly different definition reflects the actual implementation in the code and is valid for both linear and non-linear free-surface formulation, in both r-coordinate and r*-coordinate.

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

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

Note that in this case, the two definitions of ϕ_s and ϕ'_{hyd} from equation 2.79 and 2.80 converge toward the same (approximated) expressions: $\phi_s = \int_{R_o}^{r_{surf}} b_o dr$ and $\phi'_{hyd} = \int_{r}^{R_o} b' dr$.
On the contrary, the unapproximated formulation ("non-linear free-surface", see

the next section) retains the full expression: $\phi'_{hyd} = \int_r^{r_{surf}} (b - b_o) dr$. This is obtained by selecting **nonlinFreeSurf**=4 in parameter file *data*.

Regarding the surface potential:

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

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

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

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

The total thickness of the fluid column is $r_{surf} - R_{fixed} = \eta + R_o - R_{fixed}$. In most applications, the free surface displacements are small compared to the total thickness $\eta \ll H_o = R_o - R_{fixed}$. In the previous sections and in older version of the model, the linearized free-surface approximation was made, assuming $r_{surf} - R_{fixed} \simeq H_o$ when computing horizontal transports, either in the continuity equation or in tracer and momentum advection terms. This approximation is dropped when using the non-linear free-surface formulation and the total thickness, including the time varying part η, is considered when computing horizontal transports. Implications for the barotropic part are presented hereafter. In section 2.9.2.3 consequences for tracer conservation is briefly discussed (more details can be found in [8]) ; the general time-stepping is presented in section 2.9.2.4 with some limitations regarding the vertical resolution in section 2.9.2.5.

In the non-linear formulation, the continuous form of the model equations remains unchanged, except for the 2D continuity equation (2.16) which is now integrated from $R_{fixed}(x, y)$ up to $r_{surf} = R_o + \eta$:

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

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

If the column thickness is evaluated at time step n, and with implicit treatment of the surface potential gradient, equations (2.74 and 2.75) becomes:

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

where

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

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

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

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

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

Finally, another option is a "linearized" formulation where the total column thickness appears only in the integral term of the RHS (2.75) but not directly in the equation (2.74).

Those different options (see Table 2.1) have been tested and show little differences. However, we recommend the use of the most precise method (the 1rst one) since the computation cost involved in the solver matrix update is negligible.

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

Table 2.1: Non-linear free-surface flags

2.9.2.3 Tracer conservation with non-linear free-surface

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

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

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

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

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

Using the implicit (non-linear) free surface described above (section 2.3) we have:

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

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

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

The use of a 3 time-levels time-stepping scheme such as the Adams-Bashforth make the conservation sightly tricky. The current implementation with the Adams-Bashforth time-stepping provides an exact local conservation and prevents any drift in the global tracer content ([8]). Compared to the linear free-surface method, an additional step is required: the variation of the water column thickness (from h^n to h^{n+1}) is not incorporated directly into the tracer equation. Instead, the model uses the G_θ terms (first step) as in the linear free surface formulation (with the "*surface correction*" turned "on", see tracer section):

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

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

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

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

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

The grid cell thickness was hold constant with the linear free-surface ; with the non-linear free-surface, it is now varying in time, at least at the surface level. This implies some modifications of the general algorithm described earlier in sections 2.6 and 2.7.

A simplified version of the staggered in time, non-linear free-surface algorithm is detailed hereafter, and can be compared to the equivalent linear free-surface case (eq. 2.42 to 2.52) and can also be easily transposed to the synchronous time-stepping case. Among the simplifications, salinity equation, implicit operator and detailed elliptic equation are omitted. Surface forcing is explicitly written as fluxes of temperature, fresh water and momentum, $Q^{n+1/2}$, $P^{n+1/2}$, $F_{\mathbf{v}}^{n}$ respectively. h^n and dh^n are the column and grid box thickness in r-coordinate.

$$\phi_{hyd}^{n} = \int b(\theta^n, S^n, r)dr \tag{2.81}$$

$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1/2} = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(dh^{n-1}, \vec{\mathbf{v}}^{n-1/2}) \;;\; \vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n)} = \frac{3}{2}\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1/2} - \frac{1}{2}\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-3/2} \tag{2.82}$$

$$\vec{\mathbf{v}}^{*} = \vec{\mathbf{v}}^{n-1/2} + \Delta t \frac{dh^{n-1}}{dh^{n}} \left(\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n)} + F_{\vec{\mathbf{v}}}^{n}/dh^{n-1} \right) - \Delta t \nabla \phi_{hyd}^{n} \tag{2.83}$$

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

$$\begin{aligned} \eta^{n+1/2} &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{\mathbf{v}}^{n+1/2} dh^n \\ &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \left(\vec{\mathbf{v}}^{*} - g\Delta t \nabla \eta^{n+1/2} \right) dh^n \end{aligned} \tag{2.84}$$

$$\vec{\mathbf{v}}^{n+1/2} = \vec{\mathbf{v}}^{*} - g\Delta t \nabla \eta^{n+1/2} \tag{2.85}$$

$$h^{n+1} = h^n + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{\mathbf{v}}^{n+1/2} dh^n \tag{2.86}$$

$$G_{\theta}^{n} = G_{\theta}(dh^n, u^{n+1/2}, \theta^n) \;;\; G_{\theta}^{(n+1/2)} = \frac{3}{2}G_{\theta}^{n} - \frac{1}{2}G_{\theta}^{n-1} \tag{2.87}$$

$$\theta^{n+1} = \theta^n + \Delta t \frac{dh^n}{dh^{n+1}} \left(G_{\theta}^{(n+1/2)} + (P^{n+1/2}(\theta_{\mathrm{rain}} - \theta^n) + Q^{n+1/2})/dh^n \right) \tag{2.88}$$

Two steps have been added to linear free-surface algorithm (eq. 2.42 to 2.52): Firstly, the model "geometry" (here the **hFacC,W,S**) is updated just before entering *SOLVE_FOR_PRESSURE*, using the current dh^n field. Secondly, the vertically integrated continuity equation (eq.2.86) has been added (**exactConserv**=*TRUE*, in parameter file *data*, namelist *PARM01*) just before computing the vertical velocity, in subroutine *INTEGR_CONTINUITY*. This ensures that tracer and continuity equation discretization a Although this equation might appear redundant with eq.2.84, the integrated column thickness h^{n+1} can be different from $\eta^{n+1/2} + H$:

- when Crank-Nickelson time-stepping is used (see section 2.9.1).
- when filters are applied to the flow field, after (2.85) and alter the divergence of the flow.
- when the solver does not iterate until convergence ; for example, because a too large residual target was set (**cg2dTargetResidual**, parameter file *data*, namelist *PARM02*).

In this staggered time-stepping algorithm, the momentum tendencies are computed using dh^{n-1} geometry factors. (eq.2.82) and then rescaled in subroutine *TIMESTEP*, (eq.2.83), similarly to tracer tendencies (see section 2.9.2.3). The tracers are stepped forward later, using the recently updated flow field $\mathbf{v}^{n+1/2}$ and the corresponding model geometry dh^n to compute the tendencies (eq.2.87); Then the tendencies are rescaled by dh^n/dh^{n+1} to derive the new tracers values $(\theta, S)^{n+1}$ (eq.2.88, in subroutine *CALC_GT, CALC_GS*).

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

Regarding the restart procedure, two 2.D fields h^{n-1} and $(h^n - h^{n-1})/\Delta t$ in addition to the standard state variables and tendencies ($\eta^{n-1/2}$, $\mathbf{v}^{n-1/2}$, θ^n, S^n, $\mathbf{G}_{\mathbf{v}}^{n-3/2}$, $G_{\theta,S}^{n-1}$) are stored in a "*pickup*" file. The model restarts reading this "*pickup*" file, then update the model geometry according to h^{n-1}, and compute h^n and the vertical velocity before starting the main calling sequence (eq.2.81 to 2.88, *S/R FORWARD_STEP*).

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

2.9.2.5 Non-linear free-surface and vertical resolution

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

A better alternative to the vanishing level problem has been found and implemented recently, and rely on a different vertical coordinate r^* : The time variation ot the total column thickness becomes part of the r* coordinate motion, as in a σ_z, σ_p model, but the fixed part related to topography is treated as in a height or pressure coordinate model. A complete description is given in [4].

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

2.10 Spatial discretization of the dynamical equations

Spatial discretization is carried out using the finite volume method. This amounts to a grid-point method (namely second-order centered finite difference)

in the fluid interior but allows boundaries to intersect a regular grid allowing a more accurate representation of the position of the boundary. We treat the horizontal and vertical directions as separable and differently.

2.10.1 Notation

The notations we use to describe the discrete formulation of the model are summarized hereafter:
general notation:
$\Delta x, \Delta y, \Delta r$ grid spacing in X,Y,R directions.
A_o : Area of the face orthogonal to "o" direction (o=u,v,w ...).
$\mathcal{V}_u, \mathcal{V}_v, \mathcal{V}_v, \mathcal{V}_\theta$: Volume of the grid box surrounding u, v, w, θ point;
i, j, k : current index relative to X,Y,R directions;
basic operator:
δ_i : $\delta_i \Phi = \Phi_{i+1/2} - \Phi_{i-1/2}$
$\overline{\Phi}^i$: $\overline{\Phi}^i = (\Phi_{i+1/2} + \Phi_{i-1/2})/2$
δ_x : $\delta_x \Phi = \frac{1}{\Delta x} \delta_i \Phi$

$\overline{\nabla}$ = gradient operator : $\overline{\nabla}\Phi = \{\delta_x \Phi, \delta_y \Phi\}$
$\overline{\nabla}\cdot$ = divergence operator : $\overline{\nabla} \cdot \vec{\mathrm{f}} = \frac{1}{\mathcal{A}} \{\delta_i \Delta y \mathrm{f}_x + \delta_j \Delta x \mathrm{f}_y\}$
$\overline{\nabla}^2$ = Laplacian operator : $\overline{\nabla}^2 \Phi = \overline{\nabla} \cdot \overline{\nabla} \Phi$

2.10.2 The finite volume method: finite volumes versus finite difference

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

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

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

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

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

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

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

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

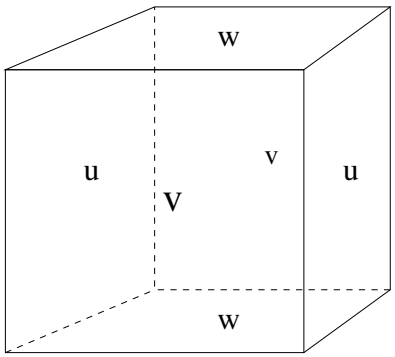


Figure 2.8: Three dimensional staggering of velocity components. This facilitates the natural discretization of the continuity and tracer equations.

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

Choosing simple centered second-order interpolation and differencing recovers the same ODE's resulting from finite differencing for the interior of a fluid. Differences arise at boundaries where a boundary is not positioned on a regular or smoothly varying grid. This method is used to represent the topography using lopped cell, see [5]. Subtle difference also appear in more than one dimension away from boundaries. This happens because the each direction is discretized independently in the finite difference method while the integrating over finite volume implicitly treats all directions simultaneously. Illustration of this is given in [2].

2.10.3 C grid staggering of variables

The basic algorithm employed for stepping forward the momentum equations is based on retaining non-divergence of the flow at all times. This is most naturally done if the components of flow are staggered in space in the form of an Arakawa C grid [7].

Fig. 2.8 shows the components of flow (u,v,w) staggered in space such that the zonal component falls on the interface between continuity cells in the zonal direction. Similarly for the meridional and vertical directions. The continuity cell is synonymous with tracer cells (they are one and the same).

2.10.4 Grid initialization and data

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

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

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

S/R INI_GRID (*model/src/ini_grid.F*)
S/R INI_MASKS_ETC (*model/src/ini_masks_etc.F*)
grid data: (*model/inc/GRID.h*)

2.10.5 Horizontal grid

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

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

Caution!

In the following, we refer to the orientation of quantities on the computational grid using geographic terminology such as points of the compass. This is purely for convenience but should note be confused with the actual geographic orientation of model quantities.

A_c: **rAc**
Δx_g: **DXg**
Δy_g: **DYg**

Fig. 2.9a shows the tracer cell (synonymous with the continuity cell). The length of the southern edge, Δx_g, western edge, Δy_g and surface area, A_c, presented in the vertical are stored in arrays **DXg**, **DYg** and **rAc**. The "g" suffix indicates that the lengths are along the defining grid boundaries. The "c" suffix associates the quantity with the cell centers. The quantities are staggered in space and the indexing is such that **DXg(i,j)** is positioned to the south of **rAc(i,j)** and **DYg(i,j)** positioned to the west.

A_ζ: **rAz**
Δx_c: **DXc**
Δy_c: **DYc**

Fig. 2.9b shows the vorticity cell. The length of the southern edge, Δx_c, western edge, Δy_c and surface area, A_ζ, presented in the vertical are stored in arrays **DXg**, **DYg** and **rAz**. The "z" suffix indicates that the lengths are measured between the cell centers and the "ζ" suffix associates points with the

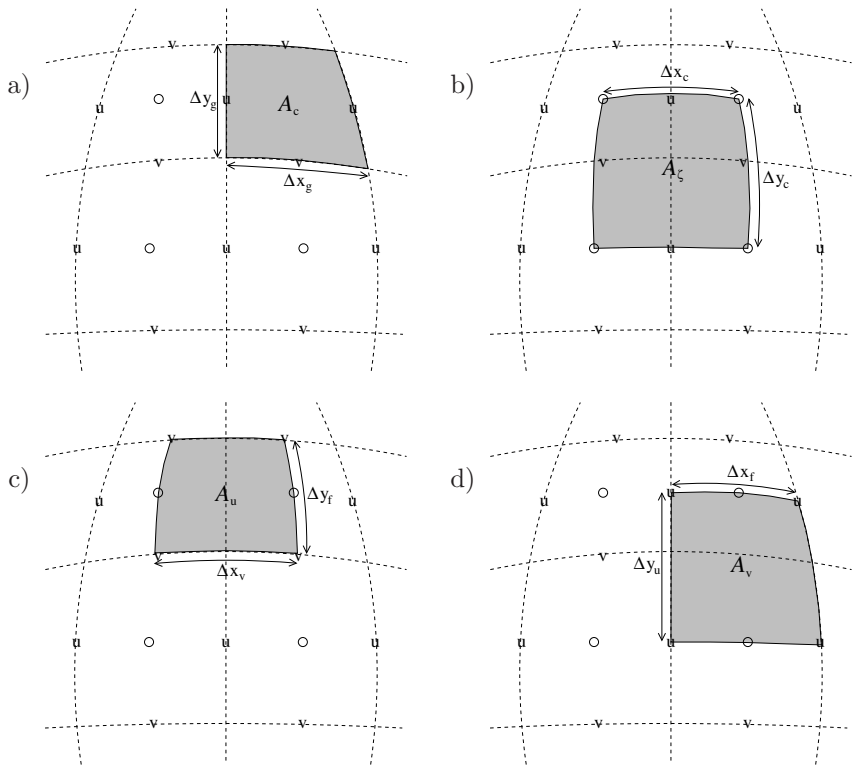


Figure 2.9: Staggering of horizontal grid descriptors (lengths and areas). The grid lines indicate the tracer cell boundaries and are the reference grid for all panels. a) The area of a tracer cell, A_c, is bordered by the lengths Δx_g and Δy_g. b) The area of a vorticity cell, A_ζ, is bordered by the lengths Δx_c and Δy_c. c) The area of a u cell, A_c, is bordered by the lengths Δx_v and Δy_f. d) The area of a v cell, A_c, is bordered by the lengths Δx_f and Δy_u.

vorticity points. The quantities are staggered in space and the indexing is such that **DXc(i,j)** is positioned to the north of **rAc(i,j)** and **DYc(i,j)** positioned to the east.

Fig. 2.9c shows the "u" or western (w) cell. The length of the southern edge, Δx_v, eastern edge, Δy_f and surface area, A_w, presented in the vertical are stored in arrays **DXv**, **DYf** and **rAw**. The "v" suffix indicates that the length is measured between the v-points, the "f" suffix indicates that the length is measured between the (tracer) cell faces and the "w" suffix associates points with the u-points (w stands for west). The quantities are staggered in space and the indexing is such that **DXv(i,j)** is positioned to the south of **rAw(i,j)** and **DYf(i,j)** positioned to the east.

A_w: **rAw**
Δx_v: **DXv**
Δy_f: **DYf**

Fig. 2.9d shows the "v" or southern (s) cell. The length of the northern edge, Δx_f, western edge, Δy_u and surface area, A_s, presented in the vertical are stored in arrays **DXf**, **DYu** and **rAs**. The "u" suffix indicates that the length is measured between the u-points, the "f" suffix indicates that the length is measured between the (tracer) cell faces and the "s" suffix associates points with the v-points (s stands for south). The quantities are staggered in space and the indexing is such that **DXf(i,j)** is positioned to the north of **rAs(i,j)** and **DYu(i,j)** positioned to the west.

A_s: **rAs**
Δx_f: **DXf**
Δy_u: **DYu**

S/R INI_CARTESIAN_GRID (*model/src/ini_cartesian_grid.F*)
S/R INI_SPHERICAL_POLAR_GRID (*model/src/ini_spherical_polar_grid.F*)
S/R INI_CURVILINEAR_GRID (*model/src/ini_curvilinear_grid.F*)
A_c, A_ζ, A_w, A_s: **rAc**, **rAz**, **rAw**, **rAs** (*GRID.h*)
Δx_g, Δy_g: **DXg**, **DYg** (*GRID.h*)
Δx_c, Δy_c: **DXc**, **DYc** (*GRID.h*)
Δx_f, Δy_f: **DXf**, **DYf** (*GRID.h*)
Δx_v, Δy_u: **DXv**, **DYu** (*GRID.h*)

2.10.5.1 Reciprocals of horizontal grid descriptors

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

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

S/R INI_MASKS_ETC (*model/src/ini_masks_etc.F*)
A_c^{-1}: **RECIP_Ac** (*GRID.h*)
A_ζ^{-1}: **RECIP_Az** (*GRID.h*)
A_w^{-1}: **RECIP_Aw** (*GRID.h*)
A_s^{-1}: **RECIP_As** (*GRID.h*)
Δx_g^{-1}, Δy_g^{-1}: **RECIP_DXg**, **RECIP_DYg** (*GRID.h*)
Δx_c^{-1}, Δy_c^{-1}: **RECIP_DXc**, **RECIP_DYc** (*GRID.h*)
Δx_f^{-1}, Δy_f^{-1}: **RECIP_DXf**, **RECIP_DYf** (*GRID.h*)
Δx_v^{-1}, Δy_u^{-1}: **RECIP_DXv**, **RECIP_DYu** (*GRID.h*)

2.10.5.2 Cartesian coordinates

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

2.10.5.3 Spherical-polar coordinates

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

2.10.5.4 Curvilinear coordinates

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

2.10.6 Vertical grid

As for the horizontal grid, we use the suffixes "c" and "f" to indicates faces and centers. Fig. 2.10a shows the default vertical grid used by the model. Δr_f is the difference in r (vertical coordinate) between the faces (i.e. $\Delta r_f \equiv -\delta_k r$ where the minus sign appears due to the convention that the surface layer has index $k = 1$.).

Δr_f: **DRf**
Δr_c: **DRc**

The vertical grid is calculated in subroutine *INI_VERTICAL_GRID* and specified via the vector **DELR** in namelist *PARM04*. The units of "r" are either meters or Pascals depending on the isomorphism being used which in turn is dependent only on the choice of equation of state.

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

Caution!

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

The above grid (Fig. 2.10a) is known as the cell centered approach because the tracer points are at cell centers; the cell centers are mid-way between the cell interfaces. This discretization is selected when the thickness of the levels are provided (**delR**, parameter file *data*, namelist *PARM04*) An alternative, the

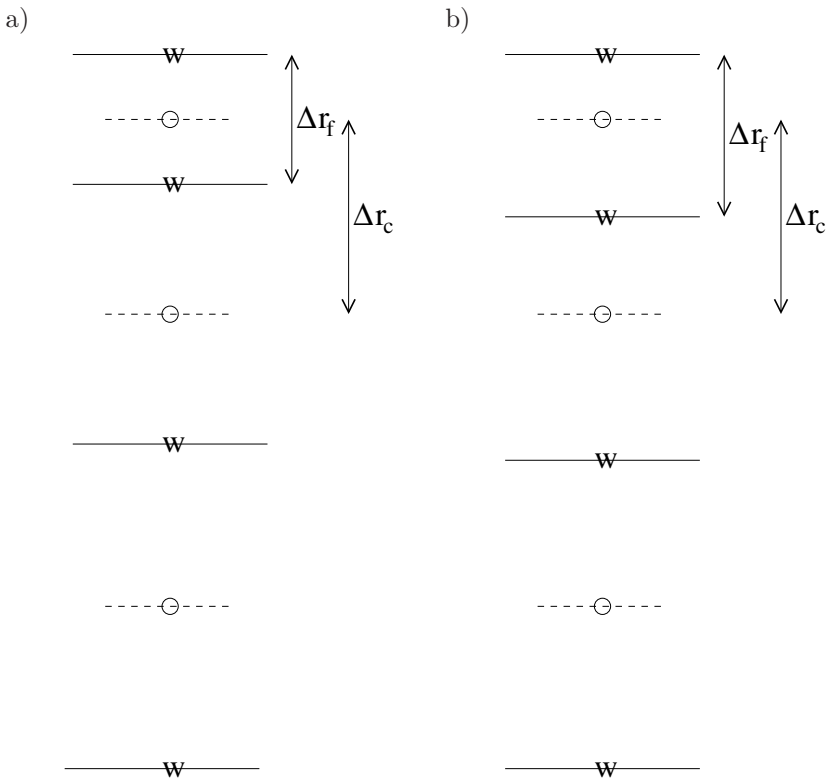


Figure 2.10: Two versions of the vertical grid. a) The cell centered approach where the interface depths are specified and the tracer points centered in between the interfaces. b) The interface centered approach where tracer levels are specified and the w-interfaces are centered in between.

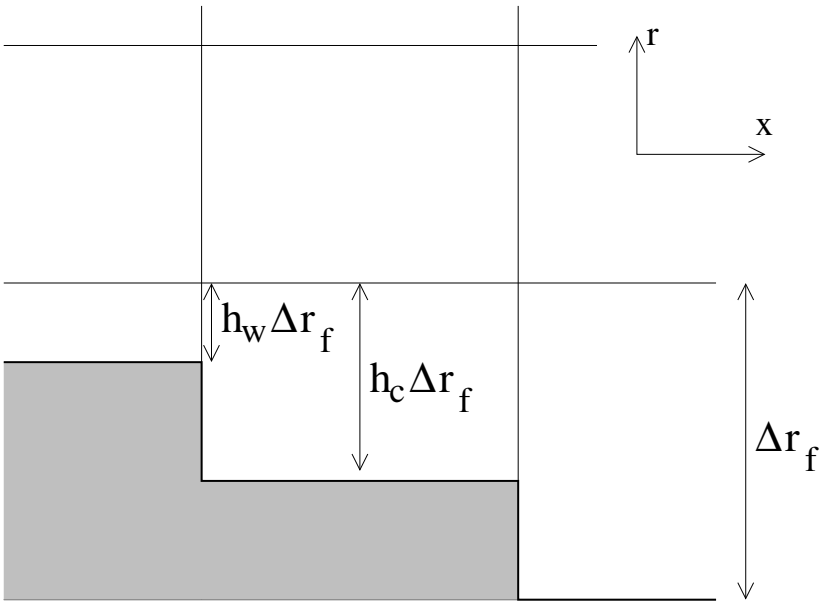


Figure 2.11: A schematic of the x-r plane showing the location of the non-dimensional fractions h_c and h_w. The physical thickness of a tracer cell is given by $h_c(i,j,k)\Delta r_f(k)$ and the physical thickness of the open side is given by $h_w(i,j,k)\Delta r_f(k)$.

vertex or interface centered approach, is shown in Fig. 2.10b. Here, the interior interfaces are positioned mid-way between the tracer nodes (no longer cell centers). This approach is formally more accurate for evaluation of hydrostatic pressure and vertical advection but historically the cell centered approach has been used. An alternative form of subroutine *INI_VERTICAL_GRID* is used to select the interface centered approach This form requires to specify $Nr+1$ vertical distances **delRc** (parameter file *data*, namelist *PARM04*, e.g. *verification/ideal_2D_oce/input/data*) corresponding to surface to center, $Nr-1$ center to center, and center to bottom distances.

S/R INI_VERTICAL_GRID (*model/src/ini_vertical_grid.F*)
Δr_f: **DRf** (*GRID.h*)
Δr_c: **DRc** (*GRID.h*)
Δr_f^{-1}: **RECIP_DRf** (*GRID.h*)
Δr_c^{-1}: **RECIP_DRc** (*GRID.h*)

2.10.7 Topography: partially filled cells

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

h_c: **hFacC**
h_w: **hFacW**
h_s: **hFacS**

Fig. 2.11 shows a schematic of the x-r plane indicating how the thickness of a level is determined at tracer and u points. The physical thickness of a tracer cell is given by $h_c(i,j,k)\Delta r_f(k)$ and the physical thickness of the open side is given by $h_w(i,j,k)\Delta r_f(k)$. Three 3-D descriptors h_c, h_w and h_s are used to describe the geometry: **hFacC**, **hFacW** and **hFacS** respectively. These are calculated in subroutine *INI_MASKS_ETC* along with there reciprocals **RECIP_hFacC**, **RECIP_hFacW** and **RECIP_hFacS**.

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

S/R INI_MASKS_ETC (*model/src/ini_masks_etc.F*)
h_c: **hFacC** (*GRID.h*)
h_w: **hFacW** (*GRID.h*)
h_s: **hFacS** (*GRID.h*)
h_c^{-1}: **RECIP_hFacC** (*GRID.h*)
h_w^{-1}: **RECIP_hFacW** (*GRID.h*)
h_s^{-1}: **RECIP_hFacS** (*GRID.h*)

2.11 Continuity and horizontal pressure gradient terms

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

$$\partial_t u + \frac{1}{\Delta x_c}\delta_i \left.\frac{\partial \Phi}{\partial r}\right|_s \eta + \frac{\epsilon_{nh}}{\Delta x_c}\delta_i \Phi'_{nh} = G_u - \frac{1}{\Delta x_c}\delta_i \Phi'_h \quad (2.89)$$

$$\partial_t v + \frac{1}{\Delta y_c}\delta_j \left.\frac{\partial \Phi}{\partial r}\right|_s \eta + \frac{\epsilon_{nh}}{\Delta y_c}\delta_j \Phi'_{nh} = G_v - \frac{1}{\Delta y_c}\delta_j \Phi'_h \quad (2.90)$$

$$\epsilon_{nh}\left(\partial_t w + \frac{1}{\Delta r_c}\delta_k \Phi'_{nh}\right) = \epsilon_{nh} G_w + \overline{b}^k - \frac{1}{\Delta r_c}\delta_k \Phi'_h \quad (2.91)$$

$$\delta_i \Delta y_g \Delta r_f h_w u + \delta_j \Delta x_g \Delta r_f h_s v + \delta_k \mathcal{A}_c w = \mathcal{A}_c \delta_k (P-E)_{r=0} \quad (2.92)$$

where the continuity equation has been most naturally discretized by staggering the three components of velocity as shown in Fig. 2.8. The grid lengths Δx_c and Δy_c are the lengths between tracer points (cell centers). The grid lengths Δx_g, Δy_g are the grid lengths between cell corners. Δr_f and Δr_c are the distance (in units of r) between level interfaces (w-level) and level centers (tracer level). The surface area presented in the vertical is denoted $\mathcal{A}_c$. The factors h_w and h_s are non-dimensional fractions (between 0 and 1) that represent the fraction cell depth that is "open" for fluid flow.

h_w: **hFacW**
h_s: **hFacS**

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

$$\mathcal{A}_c \partial_t \eta + \delta_i \sum_k \Delta y_g \Delta r_f h_w u + \delta_j \sum_k \Delta x_g \Delta r_f h_s v = \mathcal{A}_c (P-E)_{r=0} \quad (2.93)$$

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

2.12 Hydrostatic balance

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

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

$$\epsilon_{nh}\partial_t w + g\overline{\rho'}^k + \frac{1}{\Delta z}\delta_k \Phi'_h = \ldots \quad (2.94)$$

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

$$\overline{\theta'}^k + \frac{1}{\Delta \Pi}\delta_k \Phi'_h = 0 \quad (2.95)$$

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

The difference in approach between ocean and atmosphere occurs because of the direct use of the ideal gas equation in forming the potential energy conversion term $\alpha\omega$. The form of these conversion terms is discussed at length in [13].

Because of the different representation of hydrostatic balance between ocean and atmosphere there is no elegant way to represent both systems using an arbitrary coordinate.

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

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

2.13 Flux-form momentum equations

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

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

G_u: **Gu**
G_v: **Gv**
G_w: **Gw**

$$G_u = G_u^{adv} + G_u^{cor} + G_u^{h-diss} + G_u^{v-diss} + G_u^{metric} + G_u^{nh-metric} \quad (2.96)$$
$$G_v = G_v^{adv} + G_v^{cor} + G_v^{h-diss} + G_v^{v-diss} + G_v^{metric} + G_v^{nh-metric} \quad (2.97)$$
$$G_w = G_w^{adv} + G_w^{cor} + G_w^{h-diss} + G_w^{v-diss} + G_w^{metric} + G_w^{nh-metric} \quad (2.98)$$

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

These terms are calculated in routines called from subroutine *CALC_MOM_RHS* a collected into the global arrays **Gu**, **Gv**, and **Gw**.

S/R CALC_MOM_RHS (*pkg/mom_fluxform/calc_mom_rhs.F*)
G_u: **Gu** (*DYNVARS.h*)
G_v: **Gv** (*DYNVARS.h*)
G_w: **Gw** (*DYNVARS.h*)

2.13.1 Advection of momentum

The advective operator is second order accurate in space:

$$\mathcal{A}_w \Delta r_f h_w G_u^{adv} = \delta_i \overline{U}^i \overline{u}^i + \delta_j \overline{V}^i \overline{u}^j + \delta_k \overline{W}^i \overline{u}^k \quad (2.99)$$
$$\mathcal{A}_s \Delta r_f h_s G_v^{adv} = \delta_i \overline{U}^j \overline{v}^i + \delta_j \overline{V}^j \overline{v}^j + \delta_k \overline{W}^j \overline{v}^k \quad (2.100)$$
$$\mathcal{A}_c \Delta r_c G_w^{adv} = \delta_i \overline{U}^k \overline{w}^i + \delta_j \overline{V}^k \overline{w}^j + \delta_k \overline{W}^k \overline{w}^k \quad (2.101)$$

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

U: **uTrans**
V: **vTrans**
W: **rTrans**

$$U = \Delta y_g \Delta r_f h_w u \quad (2.102)$$
$$V = \Delta x_g \Delta r_f h_s v \quad (2.103)$$
$$W = \mathcal{A}_c w \quad (2.104)$$

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

S/R MOM_U_ADV_UU (*mom_u_adv_uu.F*)
S/R MOM_U_ADV_VU (*mom_u_adv_vu.F*)
S/R MOM_U_ADV_WU (*mom_u_adv_wu.F*)
S/R MOM_U_ADV_UV (*mom_u_adv_uv.F*)
S/R MOM_U_ADV_VV (*mom_u_adv_vv.F*)
S/R MOM_U_ADV_WV (*mom_u_adv_wv.F*)
uu, *uv*, *vu*, *vv*: **aF** (local to *calc_mom_rhs.F*)

2.13.2 Coriolis terms

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

$$\mathcal{A}_w \Delta r_f h_w G_u^{Cor} = \overline{f \mathcal{A}_c \Delta r_f h_c \overline{v}^j}^i - \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \overline{w}^k}^i \quad (2.105)$$
$$\mathcal{A}_s \Delta r_f h_s G_v^{Cor} = -\overline{f \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^j \quad (2.106)$$
$$\mathcal{A}_c \Delta r_c G_w^{Cor} = \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^k \quad (2.107)$$

where the Coriolis parameters f and f' are defined:

$$f = 2\Omega \sin\varphi \quad (2.108)$$
$$f' = 2\Omega \cos\varphi \quad (2.109)$$

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

$$f = f_o + \beta y \quad (2.110)$$
$$f' = 0 \quad (2.111)$$

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

$$G_u^{Cor} = f_u \overline{v}^{ji} - \epsilon_{nh} f'_u \overline{w}^{ik} \quad (2.112)$$
$$G_v^{Cor} = -f_v \overline{u}^{ij} \quad (2.113)$$
$$G_w^{Cor} = \epsilon_{nh} f'_w \overline{u}^{ik} \quad (2.114)$$

where the subscripts on f and f' indicate evaluation of the Coriolis parameters at the appropriate points in space. The above discretization does *not* conserve anything, especially energy and for historical reasons is the default for the code. A flag controls this discretization: set run-time logical **useEnergyConservingCoriolis** to *true* which otherwise defaults to *false*.

Need to change the defa code to match this

S/R MOM_CDSCHEME (*mom_cdscheme.F*)
S/R MOM_U_CORIOLIS (*mom_u_coriolis.F*)
S/R MOM_V_CORIOLIS (*mom_v_coriolis.F*)
G_u^{Cor}, G_v^{Cor}: **cF** (local to *calc_mom_rhs.F*)

2.13.3 Curvature metric terms

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

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = \overline{\frac{\overline{u}^i}{a} \tan\varphi \mathcal{A}_c \Delta r_f h_c \overline{v}^j}^i \quad (2.115)$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = -\overline{\frac{\overline{u}^i}{a} \tan\varphi \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^j \quad (2.116)$$

$$G_w^{metric} = 0 \quad (2.117)$$

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

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

$$G_u^{metric} = \frac{u\overline{v}^{ij}}{a}\tan\varphi \quad (2.118)$$

$$G_v^{metric} = \frac{\overline{u}^{ij}\overline{u}^{ij}}{a}\tan\varphi \quad (2.119)$$

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

S/R MOM_U_METRIC_SPHERE (*mom_u_metric_sphere.F*)
S/R MOM_V_METRIC_SPHERE (*mom_v_metric_sphere.F*)
G_u^{metric}, G_v^{metric}: **mT** (local to *calc_mom_rhs.F*)

2.13.4 Non-hydrostatic metric terms

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

momentum:

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = -\overline{\frac{\overline{u}^i \overline{w}^k}{a} \mathcal{A}_c \Delta r_f h_c}^i \quad (2.120)$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = -\overline{\frac{\overline{v}^j \overline{w}^k}{a} \mathcal{A}_c \Delta r_f h_c}^j \quad (2.121)$$

$$\mathcal{A}_c \Delta r_c G_w^{metric} = \overline{\frac{\overline{u}^{i\,2} + \overline{v}^{j\,2}}{a} \mathcal{A}_c \Delta r_f h_c}^k \quad (2.122)$$

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

$$G_u^{metric} = -\frac{u}{a} \overline{w}^{ik} \quad (2.123)$$

$$G_v^{metric} = -\frac{v}{a} \overline{w}^{jk} \quad (2.124)$$

$$G_w^{metric} = \frac{1}{a} (\overline{u}^{ik\,2} + \overline{v}^{jk\,2}) \quad (2.125)$$

S/R MOM_U_METRIC_NH (*mom_u_metric_nh.F*)
S/R MOM_V_METRIC_NH (*mom_v_metric_nh.F*)
G_u^{metric}, G_v^{metric}: **mT** (local to *calc_mom_rhs.F*)

2.13.5 Lateral dissipation

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

$$\mathcal{A}_w \Delta r_f h_w G_u^{h-diss} = \delta_i \Delta y_f \Delta r_f h_c \tau_{11} + \delta_j \Delta x_v \Delta r_f h_\zeta \tau_{12} \quad (2.126)$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{h-diss} = \delta_i \Delta y_u \Delta r_f h_\zeta \tau_{21} + \delta_j \Delta x_f \Delta r_f h_c \tau_{22} \quad (2.127)$$

Check signs of stress definitions

The lateral viscous stresses are discretized:

$$\tau_{11} = A_h c_{11\Delta}(\varphi) \frac{1}{\Delta x_f} \delta_i u - A_4 c_{11\Delta^2}(\varphi) \frac{1}{\Delta x_f} \delta_i \nabla^2 u \quad (2.128)$$

$$\tau_{12} = A_h c_{12\Delta}(\varphi) \frac{1}{\Delta y_u} \delta_j u - A_4 c_{12\Delta^2}(\varphi) \frac{1}{\Delta y_u} \delta_j \nabla^2 u \quad (2.129)$$

$$\tau_{21} = A_h c_{21\Delta}(\varphi) \frac{1}{\Delta x_v} \delta_i v - A_4 c_{21\Delta^2}(\varphi) \frac{1}{\Delta x_v} \delta_i \nabla^2 v \quad (2.130)$$

$$\tau_{22} = A_h c_{22\Delta}(\varphi) \frac{1}{\Delta y_f} \delta_j v - A_4 c_{22\Delta^2}(\varphi) \frac{1}{\Delta y_f} \delta_j \nabla^2 v \quad (2.131)$$

where the non-dimensional factors $c_{lm\Delta^n}(\varphi), \{l, m, n\} \in \{1, 2\}$ define the "cosine" scaling with latitude which can be applied in various ad-hoc ways. For

instance, $c_{11\Delta} = c_{21\Delta} = (\cos\varphi)^{3/2}$, $c_{12\Delta} = c_{22\Delta} = 0$ would represent the an-isotropic cosine scaling typically used on the "lat-lon" grid for Laplacian viscosity.

Need to tidy up metho controlling this in code

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

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

S/R MOM_U_XVISCFLUX (*mom_u_xviscflux.F*)
S/R MOM_U_YVISCFLUX (*mom_u_yviscflux.F*)
S/R MOM_V_XVISCFLUX (*mom_v_xviscflux.F*)
S/R MOM_V_YVISCFLUX (*mom_v_yviscflux.F*)
τ_{11}, τ_{12}, τ_{22}, τ_{22}: **vF**, **v4F** (local to *calc_mom_rhs.F*)

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

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

The no-slip condition defines the normal gradient of a tangential flow such that the flow is zero on the boundary. Rather than modify the stresses by using complicated functions of the masks and "ghost" points (see [3]) we add the boundary stresses as an additional source term in cells next to solid boundaries. This has the advantage of being able to cope with "thin walls" and also makes the interior stress calculation (code) independent of the boundary conditions. The "body" force takes the form:

$$G_u^{side-drag} = \frac{4}{\Delta z_f}(1-h_\zeta)\overline{\frac{\Delta x_v}{\Delta y_u}}^{j}\left(A_h c_{12\Delta}(\varphi)u - A_4 c_{12\Delta^2}(\varphi)\nabla^2 u\right) \quad (2.132)$$

$$G_v^{side-drag} = \frac{4}{\Delta z_f}(1-h_\zeta)\overline{\frac{\Delta y_u}{\Delta x_v}}^{i}\left(A_h c_{21\Delta}(\varphi)v - A_4 c_{21\Delta^2}(\varphi)\nabla^2 v\right) \quad (2.133)$$

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

S/R MOM_U_SIDEDRAG (*mom_u_sidedrag.F*)
S/R MOM_V_SIDEDRAG (*mom_v_sidedrag.F*)
$G_u^{side-drag}$, $G_v^{side-drag}$: **vF** (local to *calc_mom_rhs.F*)

2.13.6 Vertical dissipation

Vertical viscosity terms are discretized with only partial adherence to the variable grid lengths introduced by the finite volume formulation. This reduces the formal accuracy of these terms to just first order but only next to boundaries;

exactly where other terms appear such as linear and quadratic bottom drag.

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w}\delta_k \tau_{13} \tag{2.134}$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s}\delta_k \tau_{23} \tag{2.135}$$

$$G_w^{v-diss} = \epsilon_{nh}\frac{1}{\Delta r_f h_d}\delta_k \tau_{33} \tag{2.136}$$

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

$$\tau_{13} = A_v \frac{1}{\Delta r_c}\delta_k u \tag{2.137}$$

$$\tau_{23} = A_v \frac{1}{\Delta r_c}\delta_k v \tag{2.138}$$

$$\tau_{33} = A_v \frac{1}{\Delta r_f}\delta_k w \tag{2.139}$$

It should be noted that in the non-hydrostatic form, the stress tensor is even less consistent than for the hydrostatic (see [52]). It is well known how to do this properly (see [24]) and is on the list of to-do's.

S/R MOM_U_RVISCLFUX (*mom_u_rviscflux.F*)
S/R MOM_V_RVISCLFUX (*mom_v_rviscflux.F*)
τ_{13}: **urf** (local to *calc_mom_rhs.F*)
τ_{23}: **vrf** (local to *calc_mom_rhs.F*)

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

$$\tau_{13}^{bottom-drag} = \left(2A_v\frac{1}{\Delta r_c} + r_b + C_d\sqrt{\overline{2KE}^{i}}\right)u \tag{2.140}$$

$$\tau_{23}^{bottom-drag} = \left(2A_v\frac{1}{\Delta r_c} + r_b + C_d\sqrt{\overline{2KE}^{j}}\right)v \tag{2.141}$$

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

S/R MOM_U_BOTTOMDRAG (*mom_u_bottomdrag.F*)
S/R MOM_V_BOTTOMDRAG (*mom_v_bottomdrag.F*)
$\tau_{13}^{bottom-drag}$, $\tau_{23}^{bottom-drag}$: **vf** (local to *calc_mom_rhs.F*)

2.13.7 Derivation of discrete energy conservation

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

$$KE = \frac{1}{2}\left(\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh}\overline{w^2}^k\right) \tag{2.142}$$

2.14 Vector invariant momentum equations

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

The non-hydrostatic vector invariant equations read:

$$\partial_t \vec{v} + (2\vec{\Omega} + \vec{\zeta}) \wedge \vec{v} - b\hat{r} + \vec{\nabla} B = \vec{\nabla} \cdot \vec{\tau} \tag{2.143}$$

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

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

$$G_u = G_u^{fv} + G_u^{\zeta_3 v} + G_u^{\zeta_2 w} + G_u^{\partial_x B} + G_u^{\partial_z \tau^x} + G_u^{h-dissip} + G_u^{v-dissip} \tag{2.144}$$

$$G_v = G_v^{fu} + G_v^{\zeta_3 u} + G_v^{\zeta_1 w} + G_v^{\partial_y B} + G_v^{\partial_z \tau^y} + G_v^{h-dissip} + G_v^{v-dissip} \tag{2.145}$$

$$G_w = G_w^{fu} + G_w^{\zeta_1 v} + G_w^{\zeta_2 u} + G_w^{\partial_z B} + G_w^{h-dissip} + G_w^{v-dissip} \tag{2.146}$$

S/R CALC_MOM_RHS (*pkg/mom_vecinv/calc_mom_rhs.F*)
G_u: **Gu** (*DYNVARS.h*)
G_v: **Gv** (*DYNVARS.h*)
G_w: **Gw** (*DYNVARS.h*)

2.14.1 Relative vorticity

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

Relative vorticity is defined:

$$\zeta_3 = \frac{\Gamma}{A_\zeta} = \frac{1}{\mathcal{A}_\zeta}(\delta_i \Delta y_c v - \delta_j \Delta x_c u) \tag{2.147}$$

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

S/R MOM_VI_CALC_RELVORT3 (*mom_vi_calc_relvort3.F*)
ζ_3: **vort3** (local to *calc_mom_rhs.F*)

2.14.2 Kinetic energy

The kinetic energy, denoted KE, is defined:

$$KE = \frac{1}{2}(\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh}\overline{w^2}^k) \tag{2.148}$$

S/R MOM_VI_CALC_KE (*mom_vi_calc_ke.F*)
KE: **KE** (local to *calc_mom_rhs.F*)

2.14.3 Coriolis terms

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

$$G_u^{fv} = \frac{1}{\Delta x_c}\overline{\frac{f}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \tag{2.149}$$

$$G_v^{fu} = -\frac{1}{\Delta y_c}\overline{\frac{f}{h_\zeta}}^i \overline{\overline{\Delta y_g h_w u}^i}^j \tag{2.150}$$

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

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

$$G_u^{\zeta_3 v} = \frac{1}{\Delta x_c}\overline{\frac{\zeta_3}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \tag{2.151}$$

$$G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c}\overline{\frac{\zeta_3}{h_\zeta}}^i \overline{\overline{\Delta y_g h_w u}^i}^j \tag{2.152}$$

ζ_3: **vort3**

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

$$G_u^{fv} + G_u^{\zeta_3 v} = \frac{1}{\Delta x_c}\overline{\frac{f+\zeta_3}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \tag{2.153}$$

$$G_v^{fu} + G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c}\overline{\frac{f+\zeta_3}{h_\zeta}}^i \overline{\overline{\Delta y_g h_w u}^i}^j \tag{2.154}$$

Run-time control needs added for these options

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

S/R MOM_VI_CORIOLIS (*mom_vi_coriolis.F*)
S/R MOM_VI_U_CORIOLIS (*mom_vi_u_coriolis.F*)
S/R MOM_VI_V_CORIOLIS (*mom_vi_v_coriolis.F*)
G_u^{fv}, $G_u^{\zeta_3 v}$: **uCf** (local to *calc_mom_rhs.F*)
G_v^{fu}, $G_v^{\zeta_3 u}$: **vCf** (local to *calc_mom_rhs.F*)

2.14.4 Shear terms

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

N-H terms have not been tried!

$$G_u^{\zeta_2 w} = \frac{1}{\mathcal{A}_w \Delta r_f h_w} \overline{\overline{\mathcal{A}_c w}^i (\delta_k u - \epsilon_{nh} \delta_j w)}^k \tag{2.155}$$

$$G_v^{\zeta_1 w} = \frac{1}{\mathcal{A}_s \Delta r_f h_s} \overline{\overline{\mathcal{A}_c w}^i (\delta_k u - \epsilon_{nh} \delta_j w)}^k \tag{2.156}$$

S/R MOM_VI_U_VERTSHEAR (*mom_vi_u_vertshear.F*)
S/R MOM_VI_V_VERTSHEAR (*mom_vi_v_vertshear.F*)
$G_u^{\zeta_2 w}$: **uCf** (local to *calc_mom_rhs.F*)
$G_v^{\zeta_1 w}$: **vCf** (local to *calc_mom_rhs.F*)

2.14.5 Gradient of Bernoulli function

$$G_u^{\partial_x B} = \frac{1}{\Delta x_c} \delta_i (\phi' + KE) \tag{2.157}$$

$$G_v^{\partial_y B} = \frac{1}{\Delta x_y} \delta_j (\phi' + KE) \tag{2.158}$$

S/R MOM_VI_U_GRAD_KE (*mom_vi_u_grad_ke.F*)
S/R MOM_VI_V_GRAD_KE (*mom_vi_v_grad_ke.F*)
$G_u^{\partial_x KE}$: **uCf** (local to *calc_mom_rhs.F*)
$G_v^{\partial_y KE}$: **vCf** (local to *calc_mom_rhs.F*)

2.14.6 Horizontal dissipation

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

$$D = \frac{1}{\mathcal{A}_c h_c} (\delta_i \Delta y_g h_w u + \delta_j \Delta x_g h_s v) \tag{2.159}$$

S/R MOM_VI_CALC_HDIV (*mom_vi_calc_hdiv.F*)
D: **hDiv** (local to *calc_mom_rhs.F*)

2.14.7 Horizontal dissipation

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

$$G_u^{h-dissip} = \frac{1}{\Delta x_c}\delta_i(A_D D - A_{D4} D^*) - \frac{1}{\Delta y_u h_w}\delta_j h_\zeta (A_\zeta \zeta - A_{\zeta 4}\zeta^*) \quad (2.160)$$

$$G_v^{h-dissip} = \frac{1}{\Delta x_v h_s}\delta_i h_\zeta (A_\zeta \zeta - A_\zeta \zeta^*) + \frac{1}{\Delta y_c}\delta_j (A_D D - A_{D4} D^*) \quad (2.161)$$

where

$$D^* = \frac{1}{\mathcal{A}_c h_c}(\delta_i \Delta y_g h_w \nabla^2 u + \delta_j \Delta x_g h_s \nabla^2 v) \quad (2.162)$$

$$\zeta^* = \frac{1}{\mathcal{A}_\zeta}(\delta_i \Delta y_c \nabla^2 v - \delta_j \Delta x_c \nabla^2 u) \quad (2.163)$$

S/R MOM_VI_HDISSIP (*mom_vi_hdissip.F*)
$G_u^{h-dissip}$: **uDiss** (local to *calc_mom_rhs.F*)
$G_v^{h-dissip}$: **vDiss** (local to *calc_mom_rhs.F*)

2.14.8 Vertical dissipation

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

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w}\delta_k \tau_{13} \quad (2.164)$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s}\delta_k \tau_{23} \quad (2.165)$$

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

$$\tau_{13} = A_v \frac{1}{\Delta r_c}\delta_k u \quad (2.166)$$

$$\tau_{23} = A_v \frac{1}{\Delta r_c}\delta_k v \quad (2.167)$$

S/R MOM_U_RVISCLFUX (*mom_u_rviscflux.F*)
S/R MOM_V_RVISCLFUX (*mom_v_rviscflux.F*)
τ_{13}: **urf** (local to *calc_mom_rhs.F*)
τ_{23}: **vrf** (local to *calc_mom_rhs.F*)

2.15 Tracer equations

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

2.15.1 Time-stepping of tracers: ABII

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

$$\partial_t \tau + G^\tau_{adv} = G^\tau_{diff} + G^\tau_{forc} \tag{2.168}$$

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

$$G^\tau_{adv} = \partial_x u\tau + \partial_y v\tau + \partial_r w\tau - \tau\nabla\cdot\mathbf{v} \tag{2.169}$$

$$G^\tau_{diff} = \nabla\cdot\mathbf{K}\nabla\tau \tag{2.170}$$

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

The term, $\tau\nabla\cdot\mathbf{v}$, is required to retain local conservation in conjunction with the linear implicit free-surface. It only affects the surface layer since the flow is non-divergent everywhere else. This term is therefore referred to as the surface correction term. Global conservation is not possible using the flux-form (as here) and a linearized free-surface ([24, 8]).

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

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

S/R GAD_CALC_RHS (*pkg/generic_advdiff/gad_calc_rhs.F*)
τ: **tracer** (argument)
$G^{(n)}$: **gTracer** (argument)
F_r: **fVerT** (argument)

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

B_eps

$$G^{(n+1/2)} = (\frac{3}{2} + \epsilon)G^{(n)} - (\frac{1}{2} + \epsilon)G^{(n-1)} \tag{2.171}$$

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

S/R ADAMS_BASHFORTH2 (*model/src/adams_bashforth2.F*)
$G^{(n+1/2)}$: **gTracer** (argument on exit)
$G^{(n)}$: **gTracer** (argument on entry)
$G^{(n-1)}$: **gTrNm1** (argument)
ϵ: **ABeps** (PARAMS.h)

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

$$\tau^{(n+1)} = \tau^{(n)} + \Delta t G^{(n+1/2)} \tag{2.172}$$

Δt: **deltaTtracer**

S/R TIMESTEP_TRACER (*model/src/timestep_tracer.F*)
$\tau^{(n+1)}$: **gTracer** (argument on exit)
$\tau^{(n)}$: **tracer** (argument on entry)
$G^{(n+1/2)}$: **gTracer** (argument)
Δt: **deltaTtracer** (PARAMS.h)

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

2.16 Linear advection schemes

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

2.16.1 Centered second order advection-diffusion

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

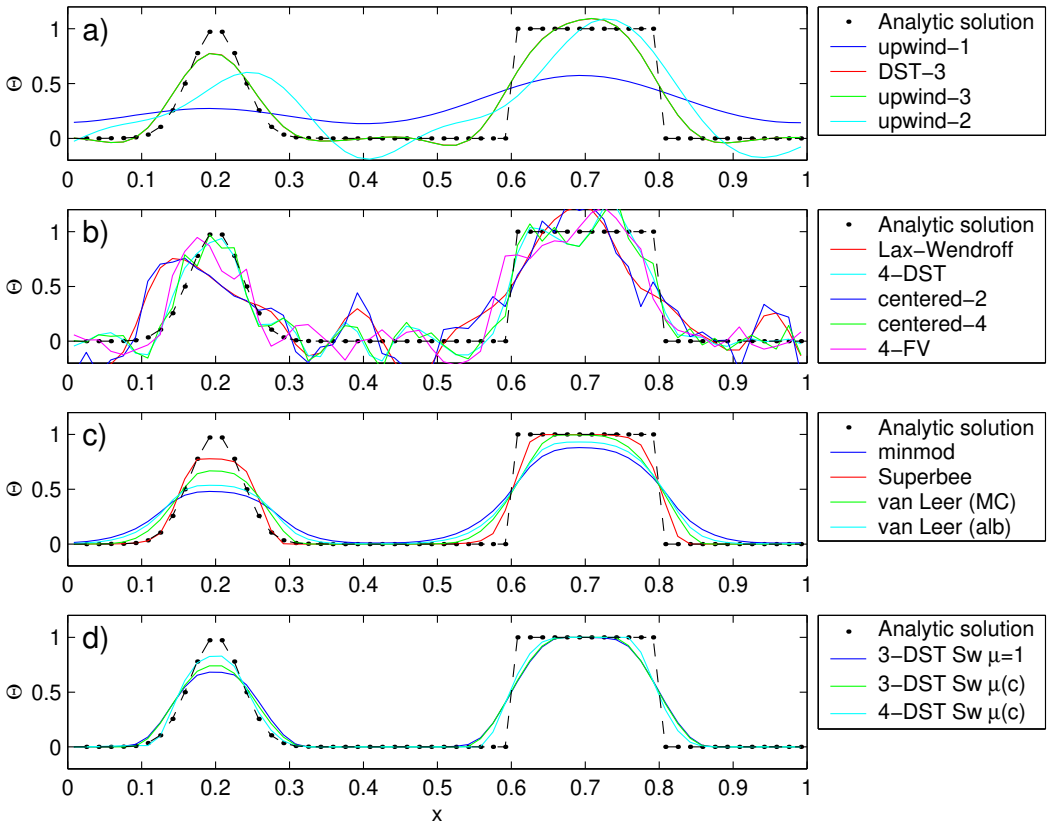


Figure 2.12: Comparison of 1-D advection schemes. Courant number is 0.05 with 60 points and solutions are shown for T=1 (one complete period). a) Shows the upwind biased schemes; first order upwind, DST3, third order upwind and second order upwind. b) Shows the centered schemes; Lax-Wendroff, DST4, centered second order, centered fourth order and finite volume fourth order. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1$, $\mu = c/(1-c)$ and a fourth order DST method with Sweby limiter, $\mu = c/(1-c)$.

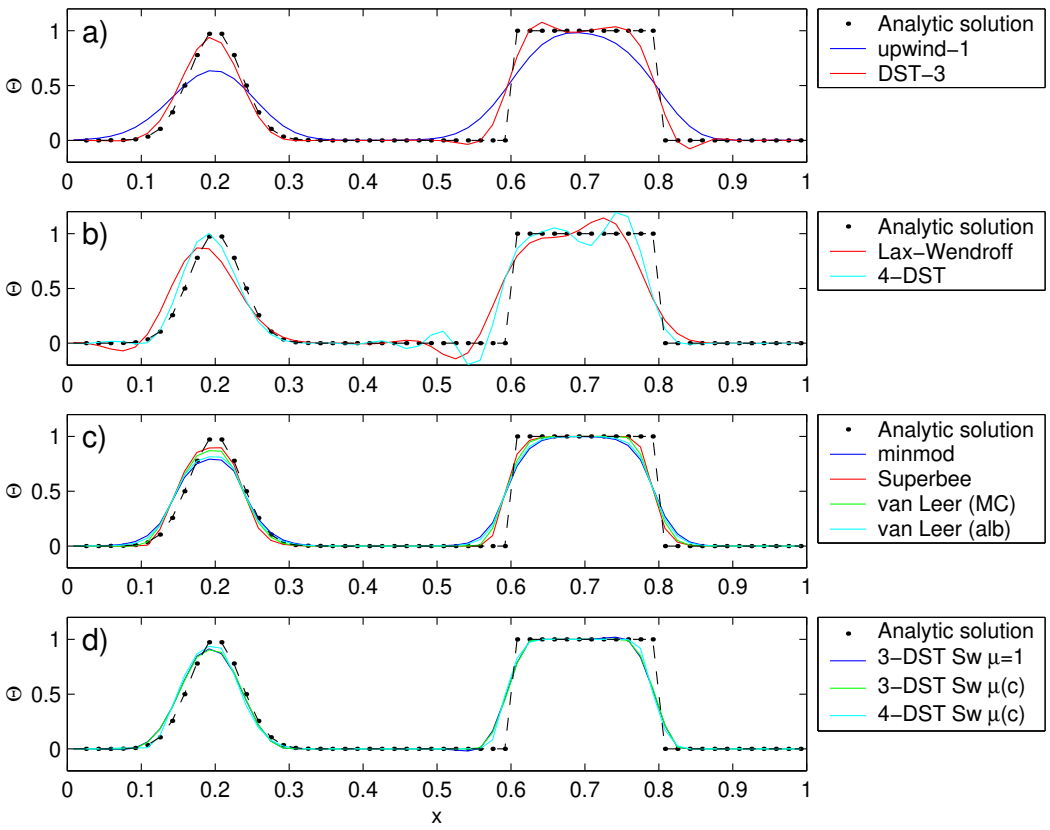


Figure 2.13: Comparison of 1-D advection schemes. Courant number is 0.89 with 60 points and solutions are shown for T=1 (one complete period). a) Shows the upwind biased schemes; first order upwind and DST3. Third order upwind and second order upwind are unstable at this Courant number. b) Shows the centered schemes; Lax-Wendroff, DST4. Centered second order, centered fourth order and finite volume fourth order and unstable at this Courant number. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1$, $\mu = c/(1-c)$ and a fourth order DST method with Sweby limiter, $\mu = c/(1-c)$.

The advection operator is discretized:

$$\mathcal{A}_c \Delta r_f h_c G^{\tau}_{adv} = \delta_i F_x + \delta_j F_y + \delta_k F_r \tag{2.173}$$

where the area integrated fluxes are given by:

$$F_x = U\overline{\tau}^i \tag{2.174}$$
$$F_y = V\overline{\tau}^j \tag{2.175}$$
$$F_r = W\overline{\tau}^k \tag{2.176}$$

U: **uTrans**
V: **vTrans**
W: **rTrans**

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

$$U = \Delta y_g \Delta r_f h_w u \tag{2.177}$$
$$V = \Delta x_g \Delta r_f h_s v \tag{2.178}$$
$$W = \mathcal{A}_c w \tag{2.179}$$

For non-divergent flow, this discretization can be shown to conserve the tracer both locally and globally and to globally conserve tracer variance, τ^2. The proof is given in [1, 5].

S/R GAD_C2_ADV_X (*gad_c2_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C2_ADV_Y (*gad_c2_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C2_ADV_R (*gad_c2_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.16.2 Third order upwind bias advection

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

The third order upwind fluxes are discretized:

$$F_x = \overline{U\tau - \frac{1}{6}\delta_{ii}\tau}^{i} + \frac{1}{2}|U|\delta_i \frac{1}{6}\delta_{ii}\tau \tag{2.180}$$
$$F_y = \overline{V\tau - \frac{1}{6}\delta_{ii}\tau}^{j} + \frac{1}{2}|V|\delta_j \frac{1}{6}\delta_{jj}\tau \tag{2.181}$$
$$F_r = \overline{W\tau - \frac{1}{6}\delta_{ii}\tau}^{k} + \frac{1}{2}|W|\delta_k \frac{1}{6}\delta_{kk}\tau \tag{2.182}$$

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

S/R GAD_U3_ADV_X (*gad_u3_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_U3_ADV_Y (*gad_u3_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_U3_ADV_R (*gad_u3_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.16.3 Centered fourth order advection

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

The centered fourth order fluxes are discretized:

$$F_x = \overline{U\tau - \frac{1}{6}\delta_{ii}\tau}^{i} \tag{2.183}$$

$$F_y = \overline{V\tau - \frac{1}{6}\delta_{ii}\tau}^{j} \tag{2.184}$$

$$F_r = \overline{W\tau - \frac{1}{6}\delta_{ii}\tau}^{k} \tag{2.185}$$

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

S/R GAD_C4_ADV_X (*gad_c4_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C4_ADV_Y (*gad_c4_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C4_ADV_R (*gad_c4_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.16.4 First order upwind advection

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

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

$$F_x = U\overline{\tau}^i - \frac{1}{2}|U|\delta_i\tau \tag{2.186}$$

$$F_y = V\overline{\tau}^j - \frac{1}{2}|V|\delta_j\tau \tag{2.187}$$

$$F_r = W\overline{\tau}^k - \frac{1}{2}|W|\delta_k\tau \tag{2.188}$$

If for some reason, the above method is required, then the second order flux limiter scheme described later reduces to the above scheme if the limiter is set to zero.

2.17 Non-linear advection schemes

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

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

2.17.1 Second order flux limiters

The second order flux limiter method can be cast in several ways but is generally expressed in terms of other flux approximations. For example, in terms of a first

order upwind flux and second order Lax-Wendroff flux, the limited flux is given as:

$$F = F_1 + \psi(r) F_{LW} \tag{2.189}$$

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

$$F_1 = u\overline{\tau}^i - \frac{1}{2}|u|\delta_i \tau \tag{2.190}$$

is the upwind flux,

$$F_{LW} = F_1 + \frac{|u|}{2}(1-c)\delta_i \tau \tag{2.191}$$

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

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

$$r = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}} \quad \forall \quad u > 0 \tag{2.192}$$

$$r = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}} \quad \forall \quad u < 0 \tag{2.193}$$

as it's argument. There are many choices of limiter function but we only provide the Superbee limiter [45]:

$$\psi(r) = \max[0, \min[1, 2r], \min[2, r]] \tag{2.194}$$

S/R GAD_FLUXLIMIT_ADV_X (*gad_fluxlimit_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_FLUXLIMIT_ADV_Y (*gad_fluxlimit_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_FLUXLIMIT_ADV_R (*gad_fluxlimit_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.17.2 Third order direct space time

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

$$F = u\left(\tau_{i-1} + d_0(\tau_i - \tau_{i-1}) + d_1(\tau_{i-1} - \tau_{i-2})\right) \quad \forall \quad u > 0 \tag{2.195}$$

$$F = u\left(\tau_i - d_0(\tau_i - \tau_{i-1}) - d_1(\tau_{i+1} - \tau_i)\right) \quad \forall \quad u < 0 \tag{2.196}$$

where

$$d_1 = \frac{1}{6}(2-|c|)(1-|c|) \tag{2.197}$$

$$d_2 = \frac{1}{6}(1-|c|)(1+|c|) \tag{2.198}$$

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

The DST3 method described above must be used in a forward-in-time manner and is stable for $0 \leq |c| \leq 1$. Although the scheme appears to be forward-in-time, it is in fact third order in time and the accuracy increases with the Courant number! For low Courant number, DST3 produces very similar results (indistinguishable in Fig. 2.12) to the linear third order method but for large Courant number, where the linear upwind third order method is unstable, the scheme is extremely accurate (Fig. 2.13) with only minor overshoots.

S/R GAD_DST3_ADV_X (*gad_dst3_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3_ADV_Y (*gad_dst3_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3_ADV_R (*gad_dst3_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.17.3 Third order direct space time with flux limiting

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

$$F = \frac{1}{2}(u+|u|)\left(\tau_{i-1} + \psi(r^+)(\tau_i - \tau_{i-1})\right) + \frac{1}{2}(u-|u|)\left(\tau_{i-1} + \psi(r^-)(\tau_i - \tau_{i-1})\right) \tag{2.199}$$

where

$$r^+ = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}} \tag{2.200}$$

$$r^- = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}} \tag{2.201}$$

and the limiter is the Sweby limiter:

$$\psi(r) = \max[0, \min[\min(1, d_0 + d_1 r], \frac{1-c}{c} r]] \tag{2.202}$$

S/R GAD_DST3FL_ADV_X (*gad_dst3_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3FL_ADV_Y (*gad_dst3_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3FL_ADV_R (*gad_dst3_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.17.4 Multi-dimensional advection

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

$$\tau^{n+1/3} = \tau^n - \Delta t \left(\frac{1}{\Delta x} \delta_i F^x(\tau^n) + \tau^n \frac{1}{\Delta x} \delta_i u \right) \tag{2.203}$$

$$\tau^{n+2/3} = \tau^n - \Delta t \left(\frac{1}{\Delta y} \delta_j F^y(\tau^{n+1/3}) + \tau^n \frac{1}{\Delta y} \delta_i v \right) \tag{2.204}$$

$$\tau^{n+3/3} = \tau^n - \Delta t \left(\frac{1}{\Delta r} \delta_k F^x(\tau^{n+2/3}) + \tau^n \frac{1}{\Delta r} \delta_i w \right) \tag{2.205}$$

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

$$G_{adv}^{n+1/2} = \frac{1}{\Delta t} (\tau^{n+3/3} - \tau^n) \tag{2.206}$$

So that the over all time-stepping looks likes:

$$\tau^{n+1} = \tau^n + \Delta t \left(G_{adv}^{n+1/2} + G_{diff}(\tau^n) + G_{forcing}^n \right) \tag{2.207}$$

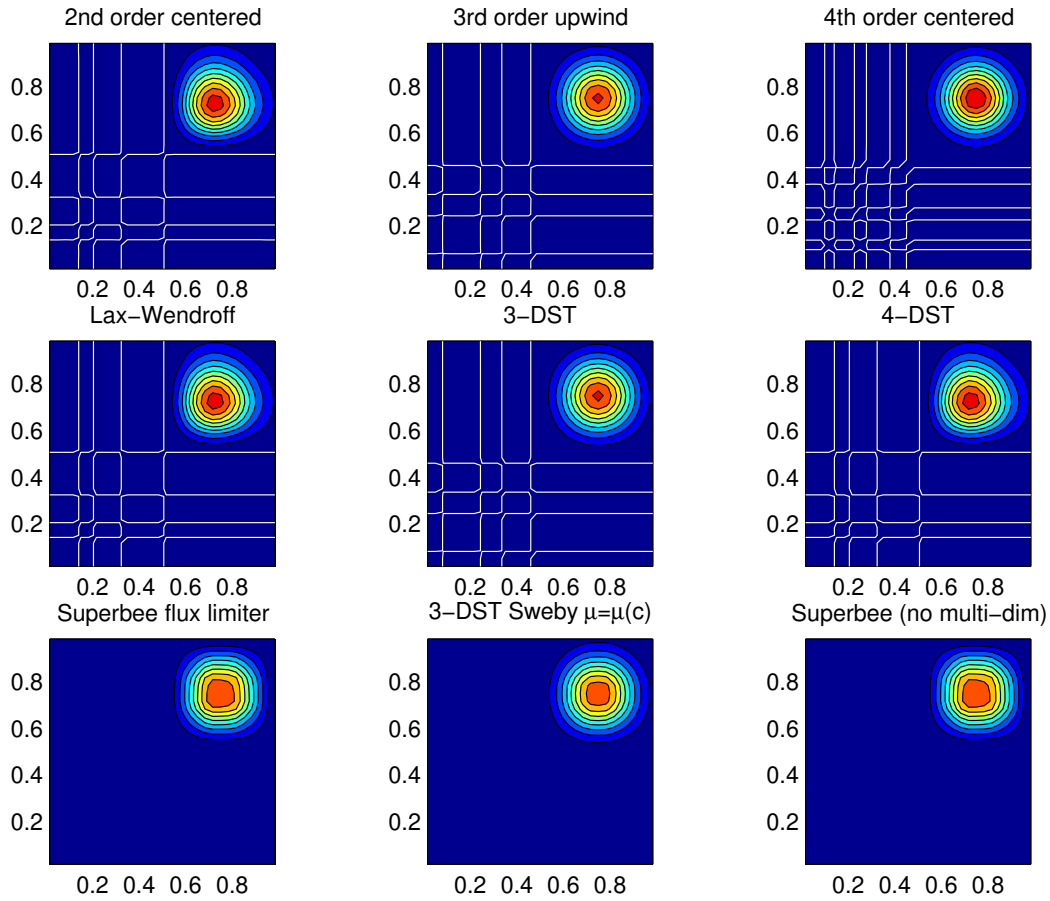


Figure 2.14: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.01 with 30×30 points and solutions are shown for T=1/2. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

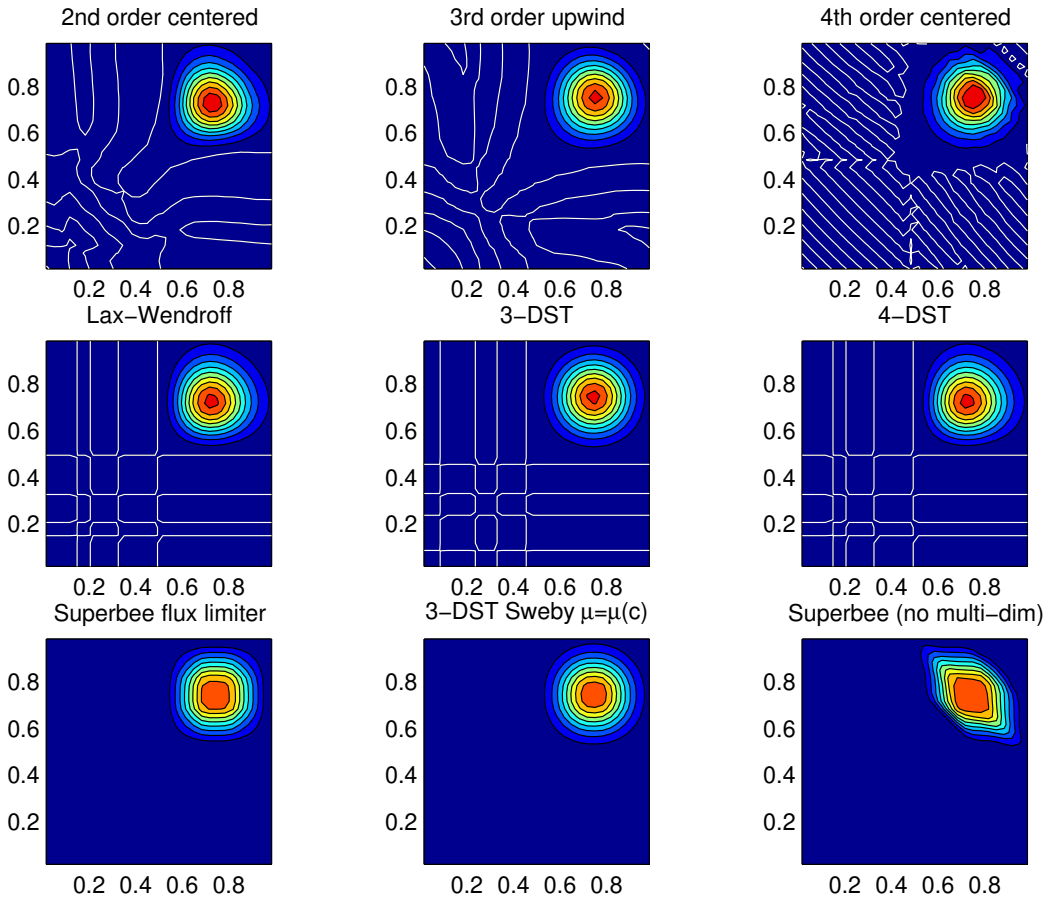


Figure 2.15: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.27 with 30×30 points and solutions are shown for T=1/2. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

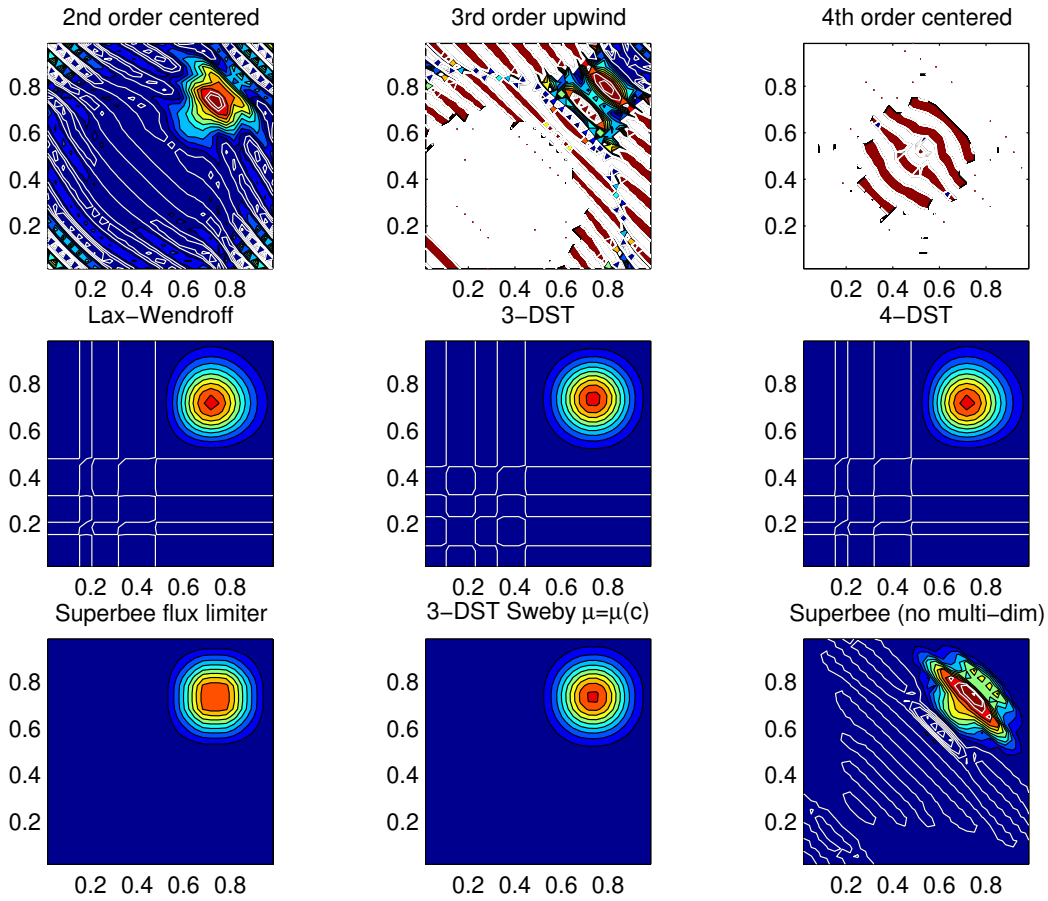


Figure 2.16: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.47 with 30×30 points and solutions are shown for T=1/2. White lines indicate zero crossings and initial maximum values (ie. the presence of false extrema). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

S/R GAD_ADVECTION (*gad_advection.F*)
τ: **Tracer** (argument)
$G_{adv}^{n+1/2}$: **Gtracer** (argument)
F_x, F_y, F_r: **af** (local)
U: **uTrans** (local)
V: **vTrans** (local)
W: **rTrans** (local)

2.18 Comparison of advection schemes

Advection Scheme	code	use A.B.	use Multi-dimension	Stencil (1 dim)	comments
centered 2^{nd}order	2	Yes	No	3 pts	linear
3^{rd}order upwind	3	Yes	No	5 pts	linear/τ
centered 4^{th}order	4	Yes	No	5 pts	linear
3^{rd}order DST	30	No	Yes	5 pts	linear/τ, non-linear/v
2^{nd}order Flux Limiters	77	No	Yes	5 pts	non-linear
3^{nd}order DST Flux limiter	33	No	Yes	5 pts	non-linear

Table 2.2: Summary of the different advection schemes available in MITgcm. "A.B." stands for Adams-Bashforth and "DST" for direct space time. The code corresponds to the number used to select the corresponding advection scheme in the parameter file (e.g., **tempAdvScheme**=3 in file *data* selects the 3^{rd} order upwind advection scheme for temperature).

Figs. 2.14, 2.15 and 2.16 show solutions to a simple diagonal advection problem using a selection of schemes for low, moderate and high Courant numbers, respectively. The top row shows the linear schemes, integrated with the Adams-Bashforth method. Theses schemes are clearly unstable for the high Courant number and weakly unstable for the moderate Courant number. The presence of false extrema is very apparent for all Courant numbers. The middle row shows solutions obtained with the unlimited but multi-dimensional schemes. These solutions also exhibit false extrema though the pattern now shows symmetry due to the multi-dimensional scheme. Also, the schemes are stable at high Courant number where the linear schemes weren't. The bottom row (left and middle) shows the limited schemes and most obvious is the absence of false extrema. The accuracy and stability of the unlimited non-linear schemes is retained at high Courant number but at low Courant number the tendency is to loose amplitude in sharp peaks due to diffusion. The one dimensional tests shown in Figs. 2.12 and 2.13 showed this phenomenon.

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

is determined by the maximum Courant number applied of each dimension while the stability of the method of lines is determined by the sum. Hence, in the high Courant number plot, the scheme is unstable.

With many advection schemes implemented in the code two questions arise: "Which scheme is best?" and "Why don't you just offer the best advection scheme?". Unfortunately, no one advection scheme is "the best" for all particular applications and for new applications it is often a matter of trial to determine which is most suitable. Here are some guidelines but these are not the rule;

- If you have a coarsely resolved model, using a positive or upwind biased scheme will introduce significant diffusion to the solution and using a centered higher order scheme will introduce more noise. In this case, simplest may be best.

- If you have a high resolution model, using a higher order scheme will give a more accurate solution but scale-selective diffusion might need to be employed. The flux limited methods offer similar accuracy in this regime.

- If your solution has shocks or propagating fronts then a flux limited scheme is almost essential.

- If your time-step is limited by advection, the multi-dimensional non-linear schemes have the most stability (up to Courant number 1).

- If you need to know how much diffusion/dissipation has occurred you will have a lot of trouble figuring it out with a non-linear method.

- The presence of false extrema is non-physical and this alone is the strongest argument for using a positive scheme.

2.19 Shapiro Filter

The Shapiro filter (Shapiro 1970, 1975) is a high order horizontal filter that efficiently remove small scale grid noise without affecting the physical structures of a field. It is applied at the end of the time step on both velocity and tracer fields.

Three different space operators are considered here (S1,S2 and S4). They differs essentially by the sequence of derivative in both X and Y directions. Consequently they show different damping response function specially in the diagonal directions X+Y and X-Y.

Space derivatives can be computed in the real space, taken into account the grid spacing. Alternatively, a pure computational filter can be defined, using pure numerical differences and ignoring grid spacing. This later form is stable whatever the grid is, and therefore specially useful for highly anisotropic grid such as spherical coordinate grid. A damping time-scale parameter τ_{shap} defines the strength of the filter damping.

The 3 computational filter operators are :

$$\text{S1c}: \qquad [1 - 1/2\frac{\Delta t}{\tau_{shap}}\{(\frac{1}{4}\delta_{ii})^n + (\frac{1}{4}\delta_{jj})^n\}]$$

$$\text{S2c}: \qquad [1 - \frac{\Delta t}{\tau_{shap}}\{\frac{1}{8}(\delta_{ii} + \delta_{jj})\}^n]$$

$$\text{S4c}: \qquad [1 - \frac{\Delta t}{\tau_{shap}}(\frac{1}{4}\delta_{ii})^n][1 - \frac{\Delta t}{\tau_{shap}}(\frac{1}{4}\delta_{jj})^n]$$

In addition, the S2 operator can easily be extended to a physical space filter:

$$\text{S2g}: \qquad [1 - \frac{\Delta t}{\tau_{shap}}\{\frac{L_{shap}^2}{8}\overline{\nabla}^2\}^n]$$

with the Laplacian operator $\overline{\nabla}^2$ and a length scale parameter L_{shap}. The stability of this S2g filter requires $L_{shap} < \text{Min}^{(Global)}(\Delta x, \Delta y)$.

Add Response functions and figures

Chapter 3

Getting started with MITgcm

In this section, we describe how to use the model. In the first section, we provide enough information to help you get started with the model. We believe the best way to familiarize yourself with the model is to run the case study examples provided with the base version. Information on how to obtain, compile, and run the code is found there as well as a brief description of the model structure directory and the case study examples. The latter and the code structure are described more fully in chapters 2 and 4, respectively. Here, in this section, we provide information on how to customize the code when you are ready to try implementing the configuration you have in mind.

3.1 Where to find information

A web site is maintained for release 2 ("Pelican") of MITgcm:

`http://mitgcm.org/pelican`

Here you will find an on-line version of this document, a "browsable" copy of the code and a searchable database of the model and site, as well as links for downloading the model and documentation, to data-sources, and other related sites.

There is also a web-archived support mailing list for the model that you can email at `MITgcm-support@mitgcm.org` or browse at:

`http://mitgcm.org/mailman/listinfo/mitgcm-support/`
`http://mitgcm.org/pipermail/mitgcm-support/`

Essentially all of the MITgcm web pages can be searched using a popular web crawler such as Google or through our own search facility:

`http://mitgcm.org/htdig/`

3.2 Obtaining the code

MITgcm can be downloaded from our system by following the instructions below. As a courtesy we ask that you send e-mail to us at MITgcm-support@mitgcm.org to enable us to keep track of who's using the model and in what application. You can download the model two ways:

1. Using CVS software. CVS is a freely available source code management tool. To use CVS you need to have the software installed. Many systems come with CVS pre-installed, otherwise good places to look for the software for a particular platform are cvshome.org and wincvs.org .
2. Using a tar file. This method is simple and does not require any special software. However, this method does not provide easy support for maintenance updates.

3.2.1 Method 1 - Checkout from CVS

If CVS is available on your system, we strongly encourage you to use it. CVS provides an efficient and elegant way of organizing your code and keeping track of your changes. If CVS is not available on your machine, you can also download a tar file.

Before you can use CVS, the following environment variable(s) should be set within your shell. For a csh or tcsh shell, put the following

```
% setenv CVSROOT :pserver:cvsanon@mitgcm.org:/u/gcmpack
```

in your .cshrc or .tcshrc file. For bash or sh shells, put:

```
% export CVSROOT=':pserver:cvsanon@mitgcm.org:/u/gcmpack'
```

in your `.profile` or `.bashrc` file.

To get MITgcm through CVS, first register with the MITgcm CVS server using command:

```
% cvs login ( CVS password: cvsanon )
```

You only need to do a "cvs login" once.

To obtain the latest sources type:

```
% cvs co MITgcm
```

or to get a specific release type:

```
% cvs co -P -r checkpoint52i_post  MITgcm
```

The MITgcm web site contains further directions concerning the source code and CVS. It also contains a web interface to our CVS archive so that one may easily view the state of files, revisions, and other development milestones:

```
http://mitgcm.org/source_code.html
```

As a convenience, the MITgcm CVS server contains aliases which are named subsets of the codebase. These aliases can be especially helpful when used over slow internet connections or on machines with restricted storage space. Table 3.1 contains a list of CVS aliases

Alias Name	Information (directories) Contained
`MITgcm_code`	Only the source code – none of the verification examples.
`MITgcm_verif_basic`	Source code plus a small set of the verification examples (`global_ocean.90x40x15`, `aim.5l_cs`, `hs94.128x64x5`, `front_relax`, and `plume_on_slope`).
`MITgcm_verif_atmos`	Source code plus all of the atmospheric examples.
`MITgcm_verif_ocean`	Source code plus all of the oceanic examples.
`MITgcm_verif_all`	Source code plus all of the verification examples.

Table 3.1: MITgcm CVS Modules

The checkout process creates a directory called *MITgcm*. If the directory *MITgcm* exists this command updates your code based on the repository. Each directory in the source tree contains a directory *CVS*. This information is required by CVS to keep track of your file versions with respect to the repository. Don't edit the files in *CVS*! You can also use CVS to download code updates. More extensive information on using CVS for maintaining MITgcm code can be found here . It is important to note that the CVS aliases in Table 3.1 cannot be used in conjunction with the CVS `-d DIRNAME` option. However, the `MITgcm` directories they create can be changed to a different name following the check-out:

```
% cvs co MITgcm_verif_basic
% mv MITgcm MITgcm_verif_basic
```

3.2.2 Method 2 - Tar file download

If you do not have CVS on your system, you can download the model as a tar file from the web site at:

`http://mitgcm.org/download/`

The tar file still contains CVS information which we urge you not to delete; even if you do not use CVS yourself the information can help us if you should need to send us your copy of the code. If a recent tar file does not exist, then please contact the developers through the MITgcm-support@mitgcm.org mailing list.

3.2.2.1 Upgrading from an earlier version

If you already have an earlier version of the code you can "upgrade" your copy instead of downloading the entire repository again. First, "cd" (change directory) to the top of your working copy:

```
% cd MITgcm
```

and then issue the cvs update command such as:

```
% cvs -q update -r checkpoint52i_post -d -P
```

This will update the "tag" to "checkpoint52i_post", add any new directories (-d) and remove any empty directories (-P). The -q option means be quiet which will reduce the number of messages you'll see in the terminal. If you have modified the code prior to upgrading, CVS will try to merge your changes with the upgrades. If there is a conflict between your modifications and the upgrade, it will report that file with a "C" in front, e.g.:

```
C model/src/ini_parms.F
```

If the list of conflicts scrolled off the screen, you can re-issue the cvs update command and it will report the conflicts. Conflicts are indicated in the code by the delimites "<<<<<<<", "=======" and ">>>>>>>". For example,

```
<<<<<<< ini_parms.F
     & bottomDragLinear,myOwnBottomDragCoefficient,
=======
     & bottomDragLinear,bottomDragQuadratic,
>>>>>>> 1.18
```

means that you added "myOwnBottomDragCoefficient" to a namelist at the same time and place that we added "bottomDragQuadratic". You need to resolve this conflict and in this case the line should be changed to:

```
     & bottomDragLinear,bottomDragQuadratic,myOwnBottomDragCoefficient,
```

and the lines with the delimiters (<<<<<<<,=======,>>>>>>>) be deleted. Unless you are making modifications which exactly parallel developments we make, these types of conflicts should be rare.

Upgrading to the current pre-release version We don't make a "release" for every little patch and bug fix in order to keep the frequency of upgrades to a minimum. However, if you have run into a problem for which "we have already fixed in the latest code" and we haven't made a "tag" or "release" since that patch then you'll need to get the latest code:

```
% cvs -q update -A -d -P
```

Unlike, the "check-out" and "update" procedures above, there is no "tag" or release name. The -A tells CVS to upgrade to the very latest version. As a rule, we don't recommend this since you might upgrade while we are in the processes of checking in the code so that you may only have part of a patch. Using this method of updating also means we can't tell what version of the code you are working with. So please be sure you understand what you're doing.

3.3 Model and directory structure

The "numerical" model is contained within a execution environment support wrapper. This wrapper is designed to provide a general framework for grid-point models. MITgcmUV is a specific numerical model that uses the framework. Under this structure the model is split into execution environment support code and conventional numerical model code. The execution environment support code is held under the *eesupp* directory. The grid point model code is held under the *model* directory. Code execution actually starts in the *eesupp* routines and not in the *model* routines. For this reason the top-level *MAIN.F* is in the *eesupp/src* directory. In general, end-users should not need to worry about this level. The top-level routine for the numerical part of the code is in *model/src/THE_MODEL_MAIN.F*. Here is a brief description of the directory structure of the model under the root tree (a detailed description is given in section 3: Code structure).

- *bin*: this directory is initially empty. It is the default directory in which to compile the code.
- *diags*: contains the code relative to time-averaged diagnostics. It is subdivided into two subdirectories *inc* and *src* that contain include files (*.*h* files) and Fortran subroutines (*.*F* files), respectively.
- *doc*: contains brief documentation notes.
- *eesupp*: contains the execution environment source code. Also subdivided into two subdirectories *inc* and *src*.
- *exe*: this directory is initially empty. It is the default directory in which to execute the code.
- *model*: this directory contains the main source code. Also subdivided into two subdirectories *inc* and *src*.
- *pkg*: contains the source code for the packages. Each package corresponds to a subdirectory. For example, *gmredi* contains the code related to the Gent-McWilliams/Redi scheme, *aim* the code relative to the atmospheric intermediate physics. The packages are described in detail in section 3.
- *tools*: this directory contains various useful tools. For example, *genmake2* is a script written in csh (C-shell) that should be used to generate your makefile. The directory *adjoint* contains the makefile specific to the Tangent linear and Adjoint Compiler (TAMC) that generates the adjoint code. The latter is described in details in part V.
- *utils*: this directory contains various utilities. The subdirectory *knudsen2* contains code and a makefile that compute coefficients of the polynomial approximation to the knudsen formula for an ocean nonlinear equation of state. The *matlab* subdirectory contains matlab scripts for

reading model output directly into matlab. *scripts* contains C-shell post-processing scripts for joining processor-based and tiled-based model output.

- *verification*: this directory contains the model examples. See section 3.4.

3.4 Example experiments

The MITgcm distribution comes with more than a dozen pre-configured numerical experiments. Some of these example experiments are tests of individual parts of the model code, but many are fully fledged numerical simulations. A few of the examples are used for tutorial documentation in sections 3.8 - 3.10. The other examples follow the same general structure as the tutorial examples. However, they only include brief instructions in a text file called *README*. The examples are located in subdirectories under the directory *verification*. Each example is briefly described below.

3.4.1 Full list of model examples

1. *exp0* - single layer, ocean double gyre (barotropic with free-surface). This experiment is described in detail in section 3.8.
2. *exp1* - Four layer, ocean double gyre. This experiment is described in detail in section **??**.
3. *exp2* - 4x4 degree global ocean simulation with steady climatological forcing. This experiment is described in detail in section 3.10.
4. *exp4* - Flow over a Gaussian bump in open-water or channel with open boundaries.
5. *exp5* - Inhomogenously forced ocean convection in a doubly periodic box.
6. *front_relax* - Relaxation of an ocean thermal front (test for Gent/McWilliams scheme). 2D (Y-Z).
7. *internal wave* - Ocean internal wave forced by open boundary conditions.
8. *natl_box* - Eastern subtropical North Atlantic with KPP scheme; 1 month integration
9. *hs94.1x64x5* - Zonal averaged atmosphere using Held and Suarez '94 forcing.
10. *hs94.128x64x5* - 3D atmosphere dynamics using Held and Suarez '94 forcing.
11. *hs94.cs-32x32x5* - 3D atmosphere dynamics using Held and Suarez '94 forcing on the cubed sphere.

12. *aim.5l_zon-ave* - Intermediate Atmospheric physics. Global Zonal Mean configuration, 1x64x5 resolution.

13. *aim.5l_XZ_Equatorial_Slice* - Intermediate Atmospheric physics, equatorial Slice configuration. 2D (X-Z).

14. *aim.5l_Equatorial_Channel* - Intermediate Atmospheric physics. 3D Equatorial Channel configuration.

15. *aim.5l_LatLon* - Intermediate Atmospheric physics. Global configuration, on latitude longitude grid with 128x64x5 grid points (2.8°degree resolution).

16. *adjustment.128x64x1* Barotropic adjustment problem on latitude longitude grid with 128x64 grid points (2.8°degree resolution).

17. *adjustment.cs-32x32x1* Barotropic adjustment problem on cube sphere grid with 32x32 points per face (roughly 2.8°degree resolution).

18. *advect_cs* Two-dimensional passive advection test on cube sphere grid.

19. *advect_xy* Two-dimensional (horizontal plane) passive advection test on Cartesian grid.

20. *advect_yz* Two-dimensional (vertical plane) passive advection test on Cartesian grid.

21. *carbon* Simple passive tracer experiment. Includes derivative calculation. Described in detail in section **??**.

22. *flt_example* Example of using float package.

23. *global_ocean.90x40x15* Global circulation with GM, flux boundary conditions and poles.

24. *global_ocean_pressure* Global circulation in pressure coordinate (non-Boussinesq ocean model). Described in detail in section 3.11.

25. *solid-body.cs-32x32x1* Solid body rotation test for cube sphere grid.

3.4.2 Directory structure of model examples

Each example directory has the following subdirectories:

- *code*: contains the code particular to the example. At a minimum, this directory includes the following files:

  - *code/CPP_EEOPTIONS.h*: declares CPP keys relative to the "execution environment" part of the code. The default version is located in *eesupp/inc*.

  - *code/CPP_OPTIONS.h*: declares CPP keys relative to the "numerical model" part of the code. The default version is located in *model/inc*.
  - *code/SIZE.h*: declares size of underlying computational grid. The default version is located in *model/inc*.

  In addition, other include files and subroutines might be present in *code* depending on the particular experiment. See Section 2 for more details.

- *input*: contains the input data files required to run the example. At a minimum, the *input* directory contains the following files:

  - *input/data*: this file, written as a namelist, specifies the main parameters for the experiment.
  - *input/data.pkg*: contains parameters relative to the packages used in the experiment.
  - *input/eedata*: this file contains "execution environment" data. At present, this consists of a specification of the number of threads to use in X and Y under multithreaded execution.

  In addition, you will also find in this directory the forcing and topography files as well as the files describing the initial state of the experiment. This varies from experiment to experiment. See section 2 for more details.

- *results*: this directory contains the output file *output.txt* produced by the simulation example. This file is useful for comparison with your own output when you run the experiment.

Once you have chosen the example you want to run, you are ready to compile the code.

3.5 Building the code

To compile the code, we use the *make* program. This uses a file (*Makefile*) that allows us to pre-process source files, specify compiler and optimization options and also figures out any file dependencies. We supply a script (*genmake2*), described in section 3.16.2, that automatically creates the *Makefile* for you. You then need to build the dependencies and compile the code.

As an example, let's assume that you want to build and run experiment *verification/exp2*. The are multiple ways and places to actually do this but here let's build the code in *verification/exp2/input*:

```
% cd verification/exp2/input
```

First, build the *Makefile*:

```
% ../../../tools/genmake2 -mods=../code
```

The command line option tells *genmake* to override model source code with any files in the directory *../code/*.

On many systems, the *genmake2* program will be able to automatically recognize the hardware, find compilers and other tools within the user's path ("echo $PATH"), and then choose an appropriate set of options from the files ("optfiles") contained in the *tools/build_options* directory. Under some circumstances, a user may have to create a new "optfile" in order to specify the exact combination of compiler, compiler flags, libraries, and other options necessary to build a particular configuration of MITgcm. In such cases, it is generally helpful to read the existing "optfiles" and mimic their syntax.

Through the MITgcm-support list, the MITgcm developers are willing to provide help writing or modifing "optfiles". And we encourage users to post new "optfiles" (particularly ones for new machines or architectures) to the MITgcm-support@mitgcm.org list.

To specify an optfile to *genmake2*, the syntax is:

```
% ../../../tools/genmake2 -mods=../code -of /path/to/optfile
```

Once a *Makefile* has been generated, we create the dependencies:

```
% make depend
```

This modifies the *Makefile* by attaching a [long] list of files upon which other files depend. The purpose of this is to reduce re-compilation if and when you start to modify the code. The `make depend` command also creates links from the model source to this directory. It is important to note that the `make depend` stage will occasionally produce warnings or errors since the dependency parsing tool is unable to find all of the necessary header files (*eg.* `netcdf.inc`). In these circumstances, it is usually OK to ignore the warnings/errors and proceed to the next step.

Next compile the code:

```
% make
```

The `make` command creates an executable called *mitgcmuv*. Additional make "targets" are defined within the makefile to aid in the production of adjoint and other versions of MITgcm.

Now you are ready to run the model. General instructions for doing so are given in section 3.6. Here, we can run the model with:

```
./mitgcmuv > output.txt
```

where we are re-directing the stream of text output to the file *output.txt*.

3.6 Running the model in prognostic mode

If compilation finished succesfuully (section 3.5) then an executable called `mitgcmuv` will now exist in the local directory.

To run the model as a single process (*ie.* not in parallel) simply type:

```
% ./mitgcmuv
```

The "./" is a safe-guard to make sure you use the local executable in case you have others that exist in your path (surely odd if you do!). The above command will spew out many lines of text output to your screen. This output contains details such as parameter values as well as diagnostics such as mean Kinetic energy, largest CFL number, etc. It is worth keeping this text output with the binary output so we normally re-direct the *stdout* stream as follows:

```
% ./mitgcmuv > output.txt
```

In the event that the model encounters an error and stops, it is very helpful to include the last few line of this `output.txt` file along with the (`stderr`) error message within any bug reports.

For the example experiments in *verification*, an example of the output is kept in *results/output.txt* for comparison. You can compare your *output.txt* with the corresponding one for that experiment to check that the set-up works.

3.6.1 Output files

The model produces various output files. Depending upon the I/O package selected (either `mdsio` or `mnc` or both as determined by both the compile-time settings and the run-time flags in `data.pkg`), the following output may appear.

3.6.1.1 MDSIO output files

The "traditional" output files are generated by the `mdsio` package. At a minimum, the instantaneous "state" of the model is written out, which is made of the following files:

- *U.00000nIter* - zonal component of velocity field (m/s and > 0 eastward).
- *V.00000nIter* - meridional component of velocity field (m/s and > 0 northward).
- *W.00000nIter* - vertical component of velocity field (ocean: m/s and > 0 upward, atmosphere: Pa/s and > 0 towards increasing pressure i.e. downward).
- *T.00000nIter* - potential temperature (ocean: ^{0}C, atmosphere: ^{0}K).
- *S.00000nIter* - ocean: salinity (psu), atmosphere: water vapor (g/kg).
- *Eta.00000nIter* - ocean: surface elevation (m), atmosphere: surface pressure anomaly (Pa).

The chain *00000nIter* consists of ten figures that specify the iteration number at which the output is written out. For example, *U.0000000300* is the zonal velocity at iteration 300.

In addition, a "pickup" or "checkpoint" file called:

- *pickup.00000nIter*

is written out. This file represents the state of the model in a condensed form and is used for restarting the integration. If the C-D scheme is used, there is an additional "pickup" file:

- *pickup_cd.00000nIter*

containing the D-grid velocity data and that has to be written out as well in order to restart the integration. Rolling checkpoint files are the same as the pickup files but are named differently. Their name contain the chain *ckptA* or *ckptB* instead of *00000nIter*. They can be used to restart the model but are overwritten every other time they are output to save disk space during long integrations.

3.6.1.2 MNC output files

Unlike the `mdsio` output, the `mnc`–generated output is usually (though not necessarily) placed within a subdirectory with a name such as `mnc_test_$DATE_$SEQ`. The files within this subdirectory are all in the "self-describing" netCDF format and can thus be browsed and/or plotted using tools such as:

- At a minimum, the `ncdump` utility is typically included with every netCDF install:

  `http://www.unidata.ucar.edu/packages/netcdf/`

- The `ncview` utility is a very convenient and quick way to plot netCDF data and it runs on most OSes:

  `http://meteora.ucsd.edu/~pierce/ncview_home_page.html`

- MatLAB(c) and other common post-processing environments provide various netCDF interfaces including:

  `http://woodshole.er.usgs.gov/staffpages/cdenham/public_html/MexCDF/nc4ml5.html`

3.6.2 Looking at the output

The "traditional" or mdsio model data are written according to a "meta/data" file format. Each variable is associated with two files with suffix names *.data* and *.meta*. The *.data* file contains the data written in binary form (big_endian by default). The *.meta* file is a "header" file that contains information about the size and the structure of the *.data* file. This way of organizing the output is particularly useful when running multi-processors calculations. The base version of the model includes a few matlab utilities to read output files written in this format. The matlab scripts are located in the directory *utils/matlab* under the root tree. The script *rdmds.m* reads the data. Look at the comments inside the script to see how to use it.

Some examples of reading and visualizing some output in *Matlab*:

```
% matlab
>> H=rdmds('Depth');
>> contourf(H');colorbar;
>> title('Depth of fluid as used by model');

>> eta=rdmds('Eta',10);
>> imagesc(eta');axis ij;colorbar;
>> title('Surface height at iter=10');

>> eta=rdmds('Eta',[0:10:100]);
>> for n=1:11; imagesc(eta(:,:,n)');axis ij;colorbar;pause(.5);end
```

Similar scripts for netCDF output (`rdmnc.m`) are available.

3.7 Tutorials

This is the first in a series of tutorials describing example MITgcm numerical experiments. The example experiments include both straightforward examples of idealized geophysical fluid simulations and more involved cases encompassing large scale modeling and automatic differentiation. Both hydrostatic and non-hydrostatic experiments are presented, as well as experiments employing Cartesian, spherical-polar and cube-sphere coordinate systems. These "case study" documents include information describing the experimental configuration and detailed information on how to configure the MITgcm code and input files for each experiment.

3.8 Barotropic Ocean Gyre In Cartesian Coordinates

This example experiment demonstrates using the MITgcm to simulate a Barotropic, wind-forced, ocean gyre circulation. The experiment is a numerical rendition of the gyre circulation problem similar to the problems described analytically by Stommel in 1966 [49] and numerically in Holland et. al [31].

In this experiment the model is configured to represent a rectangular enclosed box of fluid, 1200×1200 km in lateral extent. The fluid is 5 km deep and is forced by a constant in time zonal wind stress, τ_x, that varies sinusoidally in the "north-south" direction. Topologically the grid is Cartesian and the coriolis parameter f is defined according to a mid-latitude beta-plane equation

$$f(y) = f_0 + \beta y \tag{3.1}$$

where y is the distance along the "north-south" axis of the simulated domain. For this experiment f_0 is set to $10^{-4}s^{-1}$ in (3.1) and $\beta = 10^{-11}s^{-1}m^{-1}$.

The sinusoidal wind-stress variations are defined according to

$$\tau_x(y) = \tau_0 \sin(\pi \frac{y}{L_y}) \tag{3.2}$$

where L_y is the lateral domain extent (1200 km) and τ_0 is set to $0.1Nm^{-2}$.

Figure 3.1 summarizes the configuration simulated.

3.8.1 Equations Solved

The model is configured in hydrostatic form. The implicit free surface form of the pressure equation described in Marshall et. al [39] is employed. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation. The wind-stress momentum input is added to the momentum equation for the "zonal flow", u. Other terms in the model are explicitly switched off for this experiment configuration (see

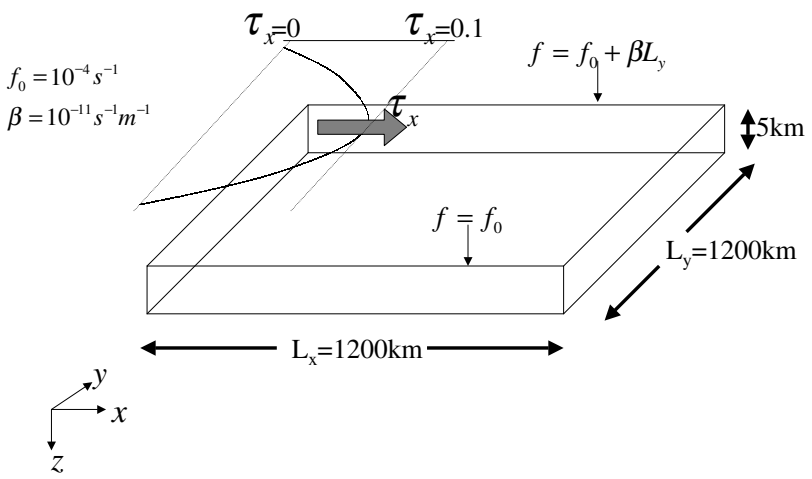


Figure 3.1: Schematic of simulation domain and wind-stress forcing function for barotropic gyre numerical experiment. The domain is enclosed bu solid walls at x = 0,1200km and at y = 0,1200km.

section 3.15.4), yielding an active set of equations solved in this configuration as follows

$$\frac{Du}{Dt} - fv + g\frac{\partial \eta}{\partial x} - A_h \nabla_h^2 u = \frac{\tau_x}{\rho_0 \Delta z} \tag{3.3}$$

$$\frac{Dv}{Dt} + fu + g\frac{\partial \eta}{\partial y} - A_h \nabla_h^2 v = 0 \tag{3.4}$$

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{3.5}$$

where u and v and the x and y components of the flow vector $\vec{u}$.

3.8.2 Discrete Numerical Configuration

The domain is discretised with a uniform grid spacing in the horizontal set to $\Delta x = \Delta y = 20$ km, so that there are sixty grid cells in the x and y directions. Vertically the model is configured with a single layer with depth, Δz, of 5000 m.

3.8.2.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, A_h, is set to $400ms^{-1}$. This value is chosen to yield a Munk layer width [1],

$$M_w = \pi(\frac{A_h}{\beta})^{\frac{1}{3}} \tag{3.6}$$

of $\approx$ 100km. This is greater than the model resolution Δx, ensuring that the frictional boundary layer is well resolved.

The model is stepped forward with a time step $\delta t = 1200$secs. With this time step the stability parameter to the horizontal Laplacian friction [1]

$$S_l = 4\frac{A_h \delta t}{\Delta x^2} \tag{3.7}$$

evaluates to 0.012, which is well below the 0.3 upper limit for stability.

The numerical stability for inertial oscillations [1]

$$S_i = f^2 \delta t^2 \tag{3.8}$$

evaluates to 0.0144, which is well below the 0.5 upper limit for stability.

The advective CFL [1] for an extreme maximum horizontal flow speed of $|\vec{u}| = 2ms^{-1}$

$$S_a = \frac{|\vec{u}|\delta t}{\Delta x} \tag{3.9}$$

evaluates to 0.12. This is approaching the stability limit of 0.5 and limits δt to $1200s$.

3.8.3 Code Configuration

The model configuration for this experiment resides under the directory *verification/exp0/*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/windx.sin_y*,
- *input/topog.box*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for this experiments. Below we describe the customizations to these files associated with this experiment.

3.8.3.1 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Line 7,

  ```
  viscAh=4.E2,
  ```

  this line sets the Laplacian friction coefficient to $400m^2s^{-1}$

- Line 10,

  ```
  beta=1.E-11,
  ```

  this line sets β (the gradient of the coriolis parameter, f) to $10^{-11}s^{-1}m^{-1}$

- Lines 15 and 16

  ```
  rigidLid=.FALSE.,
  implicitFreeSurface=.TRUE.,
  ```

  these lines suppress the rigid lid formulation of the surface pressure inverter and activate the implicit free surface form of the pressure inverter.

- Line 27,

  ```
  startTime=0,
  ```

  this line indicates that the experiment should start from $t = 0$ and implicitly suppresses searching for checkpoint files associated with restarting an numerical integration from a previously saved state.

- Line 29,

  ```
  endTime=12000,
  ```

  this line indicates that the experiment should start finish at $t = 12000s$. A restart file will be written at this time that will enable the simulation to be continued from this point.

- Line 30,

  ```
  deltaTmom=1200,
  ```

  This line sets the momentum equation timestep to $1200s$.

- Line 39,

  ```
  usingCartesianGrid=.TRUE.,
  ```

  This line requests that the simulation be performed in a Cartesian coordinate system.

- Line 41,

  ```
  delX=60*20E3,
  ```

  This line sets the horizontal grid spacing between each x-coordinate line in the discrete grid. The syntax indicates that the discrete grid should be comprise of 60 grid lines each separated by $20 \times 10^3 m$ (20 km).

- Line 42,

  ```
  delY=60*20E3,
  ```

This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to $20 \times 10^3 m$ (20 km).

- Line 43,

  ```
  delZ=5000,
  ```

  This line sets the vertical grid spacing between each z-coordinate line in the discrete grid to $5000m$ (5 km).

- Line 46,

  ```
  bathyFile='topog.box'
  ```

  This line specifies the name of the file from which the domain bathymetry is read. This file is a two-dimensional (x, y) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates a solid wall and a depth of $-5000m$ indicates open ocean. The matlab program *input/gendata.m* shows an example of how to generate a bathymetry file.

- Line 49,

  ```
  zonalWindFile='windx.sin_y'
  ```

  This line specifies the name of the file from which the x-direction surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program *input/gendata.m* includes example code to generate a valid **zonalWindFile** file.

other lines in the file *input/data* are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

```
 1 # Model parameters
 2 # Continuous equation parameters
 3  &PARM01
 4  tRef=20.,
 5  sRef=10.,
 6  viscAz=1.E-2,
 7  viscAh=4.E2,
 8  diffKhT=4.E2,
 9  diffKzT=1.E-2,
10  beta=1.E-11,
11  tAlpha=2.E-4,
```

```
 sBeta =0.,
 gravity=9.81,
 gBaro=9.81,
 rigidLid=.FALSE.,
 implicitFreeSurface=.TRUE.,
 eosType='LINEAR',
 readBinaryPrec=64,
 &
# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=1000,
 cg2dTargetResidual=1.E-7,
 &
# Time stepping parameters
 &PARM03
 startTime=0,
#endTime=311040000,
 endTime=12000.0,
 deltaTmom=1200.0,
 deltaTtracer=1200.0,
 abEps=0.1,
 pChkptFreq=2592000.0,
 chkptFreq=120000.0,
 dumpFreq=2592000.0,
 &
# Gridding parameters
 &PARM04
 usingCartesianGrid=.TRUE.,
 usingSphericalPolarGrid=.FALSE.,
 delX=60*20E3,
 delY=60*20E3,
 delZ=5000.,
 &
 &PARM05
 bathyFile='topog.box',
 hydrogThetaFile=,
 hydrogSaltFile=,
 zonalWindFile='windx.sin_y',
 meridWindFile=,
 &
```

3.8.3.2 File *input/data.pkg*

This file uses standard default values and does not contain customizations for this experiment.

3.8.3.3 File *input/eedata*

This file uses standard default values and does not contain customizations for this experiment.

3.8.3.4 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress ,τ_x, values. The units used are Nm^{-2}. Although τ_x is only a function of yn in this experiment this file must still define a complete two-dimensional map in order to be compatible with the standard code for loading forcing fields in MITgcm. The included matlab program *input/gendata.m* gives a complete code for creating the *input/windx.sin_y* file.

3.8.3.5 File *input/topog.box*

The *input/topog.box* file specifies a two-dimensional (x, y) map of depth values. For this experiment values are either $0m$ or **-delZ**m, corresponding respectively to a wall or to deep ocean. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays. The included matlab program *input/gendata.m* gives a complete code for creating the *input/topog.box* file.

3.8.3.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

```
1 C $Header: /u/gcmpack/manual/part3/case_studies/barotropic_gyre/code/SIZE.h.tex,v 1.1.1.1
2 C $Name:  $
3 C
4 C     /==========================================================\
5 C     | SIZE.h Declare size of underlying computational grid.    |
6 C     |==========================================================|
7 C     | The design here support a three-dimensional model grid   |
8 C     | with indices I,J and K. The three-dimensional domain     |
```

```
 9 C     | is comprised of nPx*nSx blocks of size sNx along one axis|
10 C     | nPy*nSy blocks of size sNy along another axis and one    |
11 C     | block of size Nz along the final axis.                   |
12 C     | Blocks have overlap regions of size OLx and OLy along the|
13 C     | dimensions that are subdivided.                          |
14 C     \==========================================================/
15 C     Voodoo numbers controlling data layout.
16 C     sNx - No. X points in sub-grid.
17 C     sNy - No. Y points in sub-grid.
18 C     OLx - Overlap extent in X.
19 C     OLy - Overlat extent in Y.
20 C     nSx - No. sub-grids in X.
21 C     nSy - No. sub-grids in Y.
22 C     nPx - No. of processes to use in X.
23 C     nPy - No. of processes to use in Y.
24 C     Nx  - No. points in X for the total domain.
25 C     Ny  - No. points in Y for the total domain.
26 C     Nr  - No. points in R for full process domain.
27       INTEGER sNx
28       INTEGER sNy
29       INTEGER OLx
30       INTEGER OLy
31       INTEGER nSx
32       INTEGER nSy
33       INTEGER nPx
34       INTEGER nPy
35       INTEGER Nx
36       INTEGER Ny
37       INTEGER Nr
38       PARAMETER (
39      &           sNx =  60,
40      &           sNy =  60,
41      &           OLx =   3,
42      &           OLy =   3,
43      &           nSx =   1,
44      &           nSy =   1,
45      &           nPx =   1,
46      &           nPy =   1,
47      &           Nx  = sNx*nSx*nPx,
48      &           Ny  = sNy*nSy*nPy,
49      &           Nr  =   1)

50 C     MAX_OLX  - Set to the maximum overlap region size of any array
51 C     MAX_OLY    that will be exchanged. Controls the sizing of exch
52 C                routine buufers.
53       INTEGER MAX_OLX
54       INTEGER MAX_OLY
55       PARAMETER ( MAX_OLX = OLx,
56      &            MAX_OLY = OLy )
```

3.8.3.7 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

3.8.3.8 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

3.9 Four Layer Baroclinic Ocean Gyre In Spherical Coordinates

This document describes an example experiment using MITgcm to simulate a baroclinic ocean gyre in spherical polar coordinates. The barotropic example experiment in section 3.8 illustrated how to configure the code for a single layer simulation in a Cartesian grid. In this example a similar physical problem is simulated, but the code is now configured for four layers and in a spherical polar coordinate system.

3.9.1 Overview

This example experiment demonstrates using the MITgcm to simulate a baroclinic, wind-forced, ocean gyre circulation. The experiment is a numerical rendition of the gyre circulation problem similar to the problems described analytically by Stommel in 1966 [49] and numerically in Holland et. al [31].

In this experiment the model is configured to represent a mid-latitude enclosed sector of fluid on a sphere, $60^\circ \times 60^\circ$ in lateral extent. The fluid is 2 km deep and is forced by a constant in time zonal wind stress, τ_λ, that varies sinusoidally in the north-south direction. Topologically the simulated domain is a sector on a sphere and the coriolis parameter, f, is defined according to latitude, φ

$$f(\varphi) = 2\Omega \sin(\varphi) \tag{3.10}$$

with the rotation rate, Ω set to $\frac{2\pi}{86400s}$.

The sinusoidal wind-stress variations are defined according to

$$\tau_\lambda(\varphi) = \tau_0 \sin(\pi \frac{\varphi}{L_\varphi}) \tag{3.11}$$

where L_φ is the lateral domain extent (60°) and τ_0 is set to $0.1 N m^{-2}$.

Figure 3.2 summarizes the configuration simulated. In contrast to the example in section 3.8, the current experiment simulates a spherical polar domain. As indicated by the axes in the lower left of the figure the model code works internally in a locally orthogonal coordinate (x, y, z). For this experiment description the local orthogonal model coordinate (x, y, z) is synonymous with the coordinates (λ, φ, r) shown in figure 1.16

The experiment has four levels in the vertical, each of equal thickness, $\Delta z =$ 500 m. Initially the fluid is stratified with a reference potential temperature profile, $\theta_{250} = 20^\circ$ C, $\theta_{750} = 10^\circ$ C, $\theta_{1250} = 8^\circ$ C, $\theta_{1750} = 6^\circ$ C. The equation of

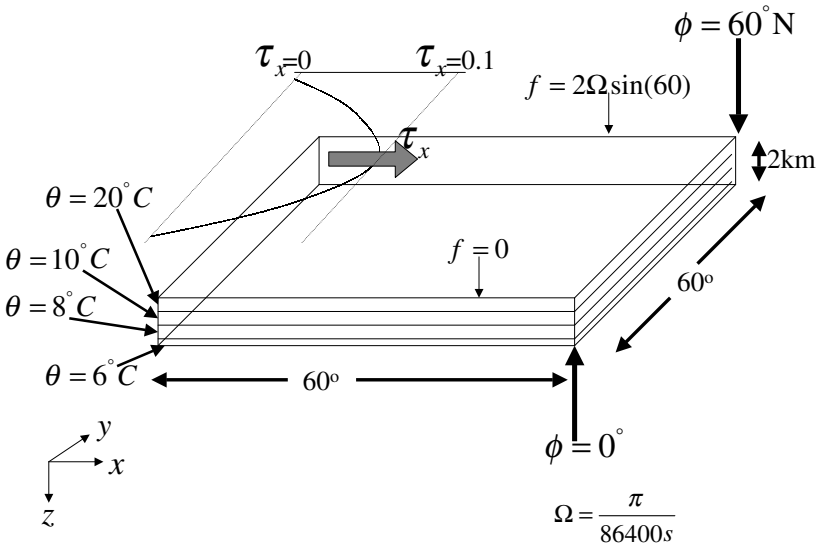


Figure 3.2: Schematic of simulation domain and wind-stress forcing function for the four-layer gyre numerical experiment. The domain is enclosed by solid walls at 0° E, 60° E, 0° N and 60° N. An initial stratification is imposed by setting the potential temperature, θ, in each layer. The vertical spacing, Δz, is constant and equal to 500m.

state used in this experiment is linear

$$\rho = \rho_0(1 - \alpha_\theta \theta^{'}) \tag{3.12}$$

which is implemented in the model as a density anomaly equation

$$\rho^{'} = -\rho_0 \alpha_\theta \theta^{'} \tag{3.13}$$

with $\rho_0 = 999.8\,\mathrm{kg\,m^{-3}}$ and $\alpha_\theta = 2 \times 10^{-4}\,\mathrm{degrees}^{-1}$. Integrated forward in this configuration the model state variable **theta** is equivalent to either in-situ temperature, T, or potential temperature, θ. For consistency with later examples, in which the equation of state is non-linear, we use θ to represent temperature here. This is the quantity that is carried in the model core equations.

3.9.2 Equations solved

For this problem the implicit free surface, **HPE** (see section **??**) form of the equations described in Marshall et. al [39] are employed. The flow is three-dimensional with just temperature, θ, as an active tracer. The equation of

state is linear. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation and provides a diffusive sub-grid scale closure for the temperature equation. A wind-stress momentum forcing is added to the momentum equation for the zonal flow, u. Other terms in the model are explicitly switched off for this experiment configuration (see section 3.9.4). This yields an active set of equations solved in this configuration, written in spherical polar coordinates as follows

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p'}{\partial \lambda} - A_h\nabla_h^2 u - A_z\frac{\partial^2 u}{\partial z^2} = \mathcal{F}_\lambda \tag{3.14}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p'}{\partial \varphi} - A_h\nabla_h^2 v - A_z\frac{\partial^2 v}{\partial z^2} = 0 \tag{3.15}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial H\widehat{u}}{\partial \lambda} + \frac{\partial H\widehat{v}}{\partial \varphi} = 0 \tag{3.16}$$

$$\frac{D\theta}{Dt} - K_h\nabla_h^2\theta - K_z\frac{\partial^2 \theta}{\partial z^2} = 0 \tag{3.17}$$

$$p' = g\rho_0\eta + \int_{-z}^{0}\rho' dz \tag{3.18}$$

$$\rho' = -\alpha_\theta\rho_0\theta' \tag{3.19}$$

$$\mathcal{F}_\lambda|_s = \frac{\tau_\lambda}{\rho_0\Delta z_s} \tag{3.20}$$

$$\mathcal{F}_\lambda|_i = 0 \tag{3.21}$$

where u and v are the components of the horizontal flow vector $\vec{u}$ on the sphere ($u = \dot{\lambda}, v = \dot{\varphi}$). The terms $H\widehat{u}$ and $H\widehat{v}$ are the components of the vertical integral term given in equation 1.35 and explained in more detail in section 2.3. However, for the problem presented here, the continuity relation (equation 3.16) differs from the general form given in section 2.3, equation **??**, because the source terms $\mathcal{P} - \mathcal{E} + \mathcal{R}$ are all 0.

The pressure field, p', is separated into a barotropic part due to variations in sea-surface height, η, and a hydrostatic part due to variations in density, ρ', integrated through the water column.

The suffices s, i indicate surface layer and the interior of the domain. The windstress forcing, $\mathcal{F}_\lambda$, is applied in the surface layer by a source term in the zonal momentum equation. In the ocean interior this term is zero.

In the momentum equations lateral and vertical boundary conditions for the ∇_h^2 and $\frac{\partial^2}{\partial z^2}$ operators are specified when the numerical simulation is run - see section 3.9.4. For temperature the boundary condition is "zero-flux" e.g. $\frac{\partial \theta}{\partial \varphi} = \frac{\partial \theta}{\partial \lambda} = \frac{\partial \theta}{\partial z} = 0$.

3.9.3 Discrete Numerical Configuration

The domain is discretised with a uniform grid spacing in latitude and longitude $\Delta\lambda = \Delta\varphi = 1^\circ$, so that there are sixty grid cells in the zonal and meridional directions. Vertically the model is configured with four layers with constant

depth, Δz, of 500 m. The internal, locally orthogonal, model coordinate variables x and y are initialized from the values of λ, φ, $\Delta\lambda$ and $\Delta\varphi$ in radians according to

$$x = r\cos(\varphi)\lambda,\ \Delta x = r\cos(\varphi)\Delta\lambda \tag{3.22}$$

$$y = r\varphi,\ \Delta y = r\Delta\varphi \tag{3.23}$$

The procedure for generating a set of internal grid variables from a spherical polar grid specification is discussed in section **??**.

S/R INI_SPHERICAL_POLAR_GRID (*model/src/ini_spherical_polar_grid.F*)
A_c, A_ζ, A_w, A_s: **rAc**, **rAz**, **rAw**, **rAs** (*GRID.h*)
Δx_g, Δy_g: **DXg**, **DYg** (*GRID.h*)
Δx_c, Δy_c: **DXc**, **DYc** (*GRID.h*)
Δx_f, Δy_f: **DXf**, **DYf** (*GRID.h*)
Δx_v, Δy_u: **DXv**, **DYu** (*GRID.h*)

As described in 2.15, the time evolution of potential temperature, θ, (equation 3.17) is evaluated prognostically. The centered second-order scheme with Adams-Bashforth time stepping described in section 2.15.1 is used to step forward the temperature equation. Prognostic terms in the momentum equations are solved using flux form as described in section **??**. The pressure forces that drive the fluid motions, ($\frac{\partial p'}{\partial \lambda}$ and $\frac{\partial p'}{\partial \varphi}$), are found by summing pressure due to surface elevation η and the hydrostatic pressure. The hydrostatic part of the pressure is diagnosed explicitly by integrating density. The sea-surface height, η, is diagnosed using an implicit scheme. The pressure field solution method is described in sections 2.3 and **??**.

3.9.3.1 Numerical Stability Criteria

The Laplacian viscosity coefficient, A_h, is set to $400ms^{-1}$. This value is chosen to yield a Munk layer width,

$$M_w = \pi(\frac{A_h}{\beta})^{\frac{1}{3}} \tag{3.24}$$

of $\approx$ 100km. This is greater than the model resolution in mid-latitudes $\Delta x = r\cos(\varphi)\Delta\lambda \approx 80$ km at $\varphi = 45°$, ensuring that the frictional boundary layer is well resolved.

The model is stepped forward with a time step $\delta t = 1200$secs. With this time step the stability parameter to the horizontal Laplacian friction

$$S_l = 4\frac{A_h\delta t}{\Delta x^2} \tag{3.25}$$

evaluates to 0.012, which is well below the 0.3 upper limit for stability for this term under ABII time-stepping.

The vertical dissipation coefficient, A_z, is set to $1 \times 10^{-2}\mathrm{m}^2\mathrm{s}^{-1}$. The associated stability limit

$$S_l = 4\frac{A_z \delta t}{\Delta z^2} \tag{3.26}$$

evaluates to 4.8×10^{-5} which is again well below the upper limit. The values of A_h and A_z are also used for the horizontal (K_h) and vertical (K_z) diffusion coefficients for temperature respectively.

The numerical stability for inertial oscillations

$$S_i = f^2 \delta t^2 \tag{3.27}$$

evaluates to 0.0144, which is well below the 0.5 upper limit for stability.

The advective CFL for a extreme maximum horizontal flow speed of $|\vec{u}| = 2ms^{-1}$

$$C_a = \frac{|\vec{u}|\delta t}{\Delta x} \tag{3.28}$$

evaluates to 5×10^{-2}. This is well below the stability limit of 0.5.

The stability parameter for internal gravity waves propagating at 2 m s^{-1}

$$S_c = \frac{c_g \delta t}{\Delta x} \tag{3.29}$$

evaluates to $\approx 5 \times 10^{-2}$. This is well below the linear stability limit of 0.25.

3.9.4 Code Configuration

The model configuration for this experiment resides under the directory *verification/exp2/*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/windx.sin_y*,
- *input/topog.box*,

- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customisations and parameter settings for this experiment. Below we describe the customisations to these files associated with this experiment.

3.9.4.1 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Line 4,

  ```
  tRef=20.,10.,8.,6.,
  ```

  this line sets the initial and reference values of potential temperature at each model level in units of °C. The entries are ordered from surface to depth. For each depth level the initial and reference profiles will be uniform in x and y. The values specified here are read into the variable `tRef` in the model code, by procedure `INI_PARMS`

S/R INI_THETA(*ini_theta.F*)

  `ini_theta.F`

- Line 6,

  ```
  viscAz=1.E-2,
  ```

  this line sets the vertical Laplacian dissipation coefficient to $1\times10^{-2}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later. The variable `viscAz` is read in the routine `ini_parms.F` and is copied into model general vertical coordinate variable `viscAr` At each time step, the viscous term contribution to the momentum equations is calculated in routine `CALC_DIFFUSIVITY`

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 7,

  ```
  viscAh=4.E2,
  ```

  this line sets the horizontal laplacian frictional dissipation coefficient to $1\times10^{-2}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later. The variable `viscAh` is read in the routine `INI_PARMS` and applied in routines `CALC_MOM_RHS` and `CALC_GW`.

S/R CALC_MOM_RHS(*calc_mom_rhs.F*)
S/R CALC_GW(*calc_gw.F*)

- Line 8,

  ```
  no_slip_sides=.FALSE.
  ```

  this line selects a free-slip lateral boundary condition for the horizontal laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x. The variable no_slip_sides is read in the routine INI_PARMS and the boundary condition is evaluated in routine

S/R CALC_MOM_RHS(*calc_mom_rhs.F*)

  calc_mom_rhs.F

- Lines 9,

  ```
  no_slip_bottom=.TRUE.
  ```

  this line selects a no-slip boundary condition for bottom boundary condition in the vertical laplacian friction operator e.g. $u = v = 0$ at $z = -H$, where H is the local depth of the domain. The variable no_slip_bottom is read in the routine INI_PARMS and is applied in the routine CALC_MOM_RHS.

S/R CALC_MOM_RHS(*calc_mom_rhs.F*)

  calc_mom_rhs.F

- Line 10,

  ```
  diffKhT=4.E2,
  ```

  this line sets the horizontal diffusion coefficient for temperature to 400 $\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ at all boundaries. The variable diffKhT is read in the routine INI_PARMS and used in routine CALC_GT.

S/R CALC_GT(*calc_gt.F*)

  calc_gt.F

- Line 11,

  ```
  diffKzT=1.E-2,
  ```

  this line sets the vertical diffusion coefficient for temperature to 10^{-2} $\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries. The variable diffKzT is read in the routine INI_PARMS. It is copied into model general vertical coordinate variable diffKrT which is used in routine CALC_DIFFUSIVITY.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

  calc_diffusivity.F

- Line 13,

```
tAlpha=2.E-4,
```

This line sets the thermal expansion coefficient for the fluid to 2×10^{-4} degrees^{-1} The variable tAlpha is read in the routine INI_PARMS. The routine FIND_RHO makes use of **tAlpha**.

S/R FIND_RHO(*find_rho.F*)

find_rho.F

- Line 18,

```
eosType='LINEAR'
```

This line selects the linear form of the equation of state. The variable eosType is read in the routine INI_PARMS. The values of **eosType** sets which formula in routine *FIND_RHO* is used to calculate density.

S/R FIND_RHO(*find_rho.F*)

find_rho.F

- Line 40,

```
usingSphericalPolarGrid=.TRUE.,
```

This line requests that the simulation be performed in a spherical polar coordinate system. It affects the interpretation of grid input parameters, for example **delX** and **delY** and causes the grid generation routines to initialize an internal grid based on spherical polar geometry. The variable usingSphericalPolarGrid is read in the routine INI_PARMS. When set to **.TRUE.** the settings of **delX** and **delY** are taken to be in degrees. These values are used in the routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)

ini_spherical_polar_grid.F

- Line 41,

```
phiMin=0.,
```

This line sets the southern boundary of the modeled domain to 0° latitude. This value affects both the generation of the locally orthogonal grid that the model uses internally and affects the initialization of the coriolis force. Note - it is not required to set a longitude boundary, since the absolute longitude does not alter the kernel equation discretisation. The variable phiMin is read in the routine INI_PARMS and is used in routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)

ini_spherical_polar_grid.F

- Line 42,

```
delX=60*1.,
```

This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to 1° in longitude. The variable delX is read in the routine INI_PARMS and is used in routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)
ini_spherical_polar_grid.F

- Line 43,

```
delY=60*1.,
```

This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to 1° in latitude. The variable delY is read in the routine INI_PARMS and is used in routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)
ini_spherical_polar_grid.F

- Line 44,

```
delZ=500.,500.,500.,500.,
```

This line sets the vertical grid spacing between each z-coordinate line in the discrete grid to 500 m, so that the total model depth is 2 km. The variable delZ is read in the routine INI_PARMS. It is copied into the internal model coordinate variable delR which is used in routine

S/R INI_VERTICAL_GRID(*ini_vertical_grid.F*)
ini_vertical_grid.F

- Line 47,

```
bathyFile='topog.box'
```

This line specifies the name of the file from which the domain bathymetry is read. This file is a two-dimensional (x, y) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates a solid wall and a depth of $-2000m$ indicates open ocean. The matlab program *input/gendata.m* shows an example of how to generate a bathymetry file. The variable bathyFile is read in the routine INI_PARMS. The bathymetry file is read in the routine

S/R INI_DEPTHS(*ini_depths.F*)
ini_depths.F

- Line 50,

  ```
  zonalWindFile='windx.sin_y'
  ```

  This line specifies the name of the file from which the x-direction (zonal) surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program *input/gendata.m* includes example code to generate a valid **zonalWindFile** file. The variable `zonalWindFile` is read in the routine `INI_PARMS`. The wind-stress file is read in the routine

S/R EXTERNAL_FIELDS_LOAD(*external_fields_load.F*)

  `external_fields_load.F`

other lines in the file *input/data* are standard values.

```
 1 # Model parameters
 2 # Continuous equation parameters
 3  &PARM01
 4  tRef=20.,10.,8.,6.,
 5  sRef=10.,10.,10.,10.,
 6  viscAz=1.E-2,
 7  viscAh=4.E2,
 8  no_slip_sides=.FALSE.,
 9  no_slip_bottom=.TRUE.,
10  diffKhT=4.E2,
11  diffKzT=1.E-2,
12  beta=1.E-11,
13  tAlpha=2.E-4,
14  sBeta =0.,
15  gravity=9.81,
16  rigidLid=.FALSE.,
17  implicitFreeSurface=.TRUE.,
18  eosType='LINEAR',
19  readBinaryPrec=64,
20  &
21 # Elliptic solver parameters
22  &PARM02
23  cg2dMaxIters=1000,
24  cg2dTargetResidual=1.E-13,
25  &
26 # Time stepping parameters
27  &PARM03
28  startTime=0.,
29  endTime=12000.,
30  deltaTmom=1200.0,
31  deltaTtracer=1200.0,
32  abEps=0.1,
33  pChkptFreq=17000.0,
34  chkptFreq=0.0,
```

```
 dumpFreq=2592000.0,
 &
# Gridding parameters
 &PARM04
 usingCartesianGrid=.FALSE.,
 usingSphericalPolarGrid=.TRUE.,
 phiMin=0.,
 delX=60*1.,
 delY=60*1.,
 delZ=500.,500.,500.,500.,
 &
 &PARM05
 bathyFile='topog.box',
 hydrogThetaFile=,
 hydrogSaltFile=,
 zonalWindFile='windx.sin_y',
 meridWindFile=,
 &
```

3.9.4.2 File *input/data.pkg*

This file uses standard default values and does not contain customisations for this experiment.

3.9.4.3 File *input/eedata*

This file uses standard default values and does not contain customisations for this experiment.

3.9.4.4 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress ,τ_x, values. The units used are Nm^{-2} (the default for MITgcm). Although τ_x is only a function of latitude, y, in this experiment this file must still define a complete two-dimensional map in order to be compatible with the standard code for loading forcing fields in MITgcm (routine *EXTERNAL_FIELDS_LOAD*. The included matlab program *input/gendata.m* gives a complete code for creating the *input/windx.sin_y* file.

3.9.4.5 File *input/topog.box*

The *input/topog.box* file specifies a two-dimensional (x, y) map of depth values. For this experiment values are either 0 m or -2000 m, corresponding respectively to a wall or to deep ocean. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays. The included matlab program *input/gendata.m* gives a complete code for creating the *input/topog.box* file.

3.9.4.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=4,
  ```

  this line sets the vertical domain extent in grid points.

3.9.4.7 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.9.4.8 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.9.4.9 Other Files

Other files relevant to this experiment are

- *model/src/ini_cori.F*. This file initializes the model coriolis variables **fCorU** and **fCorV**.
- *model/src/ini_spherical_polar_grid.F* This file initializes the model grid discretisation variables **dxF, dyF, dxG, dyG, dxC, dyC**.
- *model/src/ini_parms.F*.

3.9.5 Running The Example

3.9.5.1 Code Download

In order to run the examples you must first download the code distribution. Instructions for downloading the code can be found in section 3.2.

3.9.5.2 Experiment Location

This example experiments is located under the release sub-directory

verification/exp2/

3.9.5.3 Running the Experiment

To run the experiment

1. Set the current directory to *input/*

   ```
   % cd input
   ```

2. Verify that current directory is now correct

   ```
   % pwd
   ```

   You should see a response on the screen ending in
   verification/exp2/input

3. Run the genmake script to create the experiment *Makefile*

   ```
   % ../../../tools/genmake -mods=../code
   ```

4. Create a list of header file dependencies in *Makefile*

   ```
   % make depend
   ```

5. Build the executable file.

   ```
   % make
   ```

6. Run the *mitgcmuv* executable

   ```
   % ./mitgcmuv
   ```

3.10 Global Ocean Simulation at 4° Resolution

This example experiment demonstrates using the MITgcm to simulate the planetary ocean circulation. The simulation is configured with realistic geography and bathymetry on a $4^\circ \times 4^\circ$ spherical polar grid. Twenty levels are used in the vertical, ranging in thickness from 50 m at the surface to 815 m at depth, giving a maximum model depth of 6 km. At this resolution, the configuration can be integrated forward for thousands of years on a single processor desktop computer.

3.10.1 Overview

The model is forced with climatological wind stress data and surface flux data from DaSilva [10]. Climatological data from Levitus [36] is used to initialize the model hydrography. Levitus seasonal climatology data is also used throughout the calculation to provide additional air-sea fluxes. These fluxes are combined with the DaSilva climatological estimates of surface heat flux and fresh water, resulting in a mixed boundary condition of the style described in Haney [25]. Altogether, this yields the following forcing applied in the model surface layer.

$$\mathcal{F}_u = \frac{\tau_x}{\rho_0 \Delta z_s} \tag{3.30}$$

$$\mathcal{F}_v = \frac{\tau_y}{\rho_0 \Delta z_s} \tag{3.31}$$

$$\mathcal{F}_\theta = -\lambda_\theta(\theta - \theta^*) - \frac{1}{C_p \rho_0 \Delta z_s}\mathcal{Q} \tag{3.32}$$

$$\mathcal{F}_s = -\lambda_s(S - S^*) + \frac{S_0}{\Delta z_s}(\mathcal{E} - \mathcal{P} - \mathcal{R}) \tag{3.33}$$

where $\mathcal{F}_u$, $\mathcal{F}_v$, $\mathcal{F}_\theta$, $\mathcal{F}_s$ are the forcing terms in the zonal and meridional momentum and in the potential temperature and salinity equations respectively. The term Δz_s represents the top ocean layer thickness in meters. It is used in conjunction with a reference density, ρ_0 (here set to $999.8\,\mathrm{kg\,m^{-3}}$), a reference salinity, S_0 (here set to 35 ppt), and a specific heat capacity, C_p (here set to 4000 J $^\circ\mathrm{C}^{-1}$ kg^{-1}), to convert input dataset values into time tendencies of potential temperature (with units of $^\circ\mathrm{C\ s^{-1}}$), salinity (with units ppt s^{-1}) and velocity (with units $\mathrm{m\ s^{-2}}$). The externally supplied forcing fields used in this experiment are τ_x, τ_y, θ^*, S^*, $\mathcal{Q}$ and $\mathcal{E} - \mathcal{P} - \mathcal{R}$. The wind stress fields (τ_x, τ_y) have units of $\mathrm{N\ m^{-2}}$. The temperature forcing fields (θ^* and Q) have units of $^\circ$C and $\mathrm{W\ m^{-2}}$ respectively. The salinity forcing fields (S^* and $\mathcal{E} - \mathcal{P} - \mathcal{R}$) have units of ppt and $\mathrm{m\ s^{-1}}$ respectively. The source files and procedures for ingesting this data into the simulation are described in the experiment configuration discussion in section 3.10.3.

3.10.2 Discrete Numerical Configuration

The model is configured in hydrostatic form. The domain is discretised with a uniform grid spacing in latitude and longitude on the sphere $\Delta\phi = \Delta\lambda = 4^{\circ}$, so that there are ninety grid cells in the zonal and forty in the meridional direction. The internal model coordinate variables x and y are initialized according to

$$x = r\cos(\phi),\ \Delta x = r\cos(\Delta\phi) \tag{3.34}$$

$$y = r\lambda,\ \Delta y = r\Delta\lambda \tag{3.35}$$

Arctic polar regions are not included in this experiment. Meridionally the model extends from 80°S to 80°N. Vertically the model is configured with twenty layers with the following thicknesses $\Delta z_1 = 50\,\mathrm{m}$, $\Delta z_2 = 50\,\mathrm{m}$, $\Delta z_3 = 55\,\mathrm{m}$, $\Delta z_4 = 60\,\mathrm{m}$, $\Delta z_5 = 65\,\mathrm{m}$, $\Delta z_6 = 70\,\mathrm{m}$, $\Delta z_7 = 80\,\mathrm{m}$, $\Delta z_8 = 95\,\mathrm{m}$, $\Delta z_9 = 120\,\mathrm{m}$, $\Delta z_{10} = 155\,\mathrm{m}$, $\Delta z_{11} = 200\,\mathrm{m}$, $\Delta z_{12} = 260\,\mathrm{m}$, $\Delta z_{13} = 320\,\mathrm{m}$, $\Delta z_{14} = 400\,\mathrm{m}$, $\Delta z_{15} = 480\,\mathrm{m}$, $\Delta z_{16} = 570\,\mathrm{m}$, $\Delta z_{17} = 655\,\mathrm{m}$, $\Delta z_{18} = 725\,\mathrm{m}$, $\Delta z_{19} = 775\,\mathrm{m}$, $\Delta z_{20} = 815\,\mathrm{m}$ (here the numeric subscript indicates the model level index number, `k`) to give a total depth, H, of -5450m. The implicit free surface form of the pressure equation described in Marshall et. al [39] is employed. A Laplacian operator, ∇^2, provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress forcing is added to the momentum equations in (3.36) for both the zonal flow, u and the meridional flow v, according to equations (3.30) and (3.31). Thermodynamic forcing inputs are added to the equations in (3.36) for potential temperature, θ, and salinity, S, according to equations (3.32) and (3.33). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p^{'}}{\partial x} - \nabla_h \cdot A_h \nabla_h u - \frac{\partial}{\partial z} A_z \frac{\partial u}{\partial z} = \begin{cases} \mathcal{F}_u & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.36}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p^{'}}{\partial y} - \nabla_h \cdot A_h \nabla_h v - \frac{\partial}{\partial z} A_z \frac{\partial v}{\partial z} = \begin{cases} \mathcal{F}_v & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.37}$$

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{3.38}$$

$$\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - \frac{\partial}{\partial z}\Gamma(K_z)\frac{\partial \theta}{\partial z} = \begin{cases} \mathcal{F}_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.39}$$

$$\frac{Ds}{Dt} - \nabla_h \cdot K_h \nabla_h s - \frac{\partial}{\partial z}\Gamma(K_z)\frac{\partial s}{\partial z} = \begin{cases} \mathcal{F}_s & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.40}$$

$$g\rho_0\eta + \int_{-z}^{0} \rho^{'}\, dz = p^{'} \tag{3.41}$$

where $u = \frac{Dx}{Dt} = r\cos(\phi)\frac{D\lambda}{Dt}$ and $v = \frac{Dy}{Dt} = r\frac{D\phi}{Dt}$ are the zonal and meridional components of the flow vector, $\vec{u}$, on the sphere. As described in MITgcm Numerical Solution Procedure 2, the time evolution of potential temperature, θ,

equation is solved prognostically. The total pressure, p, is diagnosed by summing pressure due to surface elevation η and the hydrostatic pressure.

3.10.2.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, A_h, is set to $5 \times 10^5 ms^{-1}$. This value is chosen to yield a Munk layer width [1],

$$M_w = \pi(\frac{A_h}{\beta})^{\frac{1}{3}} \tag{3.42}$$

of $\approx$ 600km. This is greater than the model resolution in low-latitudes, $\Delta x \approx$ 400km, ensuring that the frictional boundary layer is adequately resolved.

The model is stepped forward with a time step $\delta t_\theta = 30$ hours for thermodynamic variables and $\delta t_v = 40$ minutes for momentum terms. With this time step, the stability parameter to the horizontal Laplacian friction [1]

$$S_l = 4\frac{A_h \delta t_v}{\Delta x^2} \tag{3.43}$$

evaluates to 0.16 at a latitude of $\phi = 80°$, which is below the 0.3 upper limit for stability. The zonal grid spacing Δx is smallest at $\phi = 80°$ where $\Delta x = r\cos(\phi)\Delta\phi \approx 77$km.

The vertical dissipation coefficient, A_z, is set to $1 \times 10^{-3}\mathrm{m}^2\mathrm{s}^{-1}$. The associated stability limit

$$S_l = 4\frac{A_z \delta t_v}{\Delta z^2} \tag{3.44}$$

evaluates to 0.015 for the smallest model level spacing ($\Delta z_1 = 50$m) which is again well below the upper stability limit.

The values of the horizontal (K_h) and vertical (K_z) diffusion coefficients for both temperature and salinity are set to 1×10^3 $\mathrm{m}^2\mathrm{s}^{-1}$ and 3×10^{-5} $\mathrm{m}^2\mathrm{s}^{-1}$ respectively. The stability limit related to K_h will be at $\phi = 80°$ where $\Delta x \approx$ 77km. Here the stability parameter

$$S_l = \frac{4K_h \delta t_\theta}{\Delta x^2} \tag{3.45}$$

evaluates to 0.07, well below the stability limit of $S_l \approx 0.5$. The stability parameter related to K_z

$$S_l = \frac{4K_z \delta t_\theta}{\Delta z^2} \tag{3.46}$$

evaluates to 0.005 for $\min(\Delta z) = 50$m, well below the stability limit of $S_l \approx 0.5$.

The numerical stability for inertial oscillations [1]

$$S_i = f^2 {\delta t_v}^2 \tag{3.47}$$

evaluates to 0.24 for $f = 2\omega \sin(80°) = 1.43 \times 10^{-4}$ s^{-1}, which is close to the $S_i < 1$ upper limit for stability.

The advective CFL [1] for a extreme maximum horizontal flow speed of $|\vec{u}| = 2ms^{-1}$

$$S_a = \frac{|\vec{u}|\delta t_v}{\Delta x} \tag{3.48}$$

evaluates to 6×10^{-2}. This is well below the stability limit of 0.5.

The stability parameter for internal gravity waves propagating with a maximum speed of $c_g = 10$ ms^{-1} [1]

$$S_c = \frac{c_g \delta t_v}{\Delta x} \tag{3.49}$$

evaluates to 3×10^{-1}. This is close to the linear stability limit of 0.5.

3.10.3 Experiment Configuration

The model configuration for this experiment resides under the directory *tutorial_examples/global_ocean_circulation/*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/windx.bin*,
- *input/windy.bin*,
- *input/salt.bin*,
- *input/theta.bin*,
- *input/SSS.bin*,
- *input/SST.bin*,
- *input/topog.bin*,
- *code/CPP_EEOPTIONS.h*

- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

3.10.3.1 Driving Datasets

Figures (**??**-**??**) show the relaxation temperature (θ^*) and salinity (S^*) fields, the wind stress components (τ_x and τ_y), the heat flux (Q) and the net fresh water flux ($\mathcal{E} - \mathcal{P} - \mathcal{R}$) used in equations **??**-**??**. The figures also indicate the lateral extent and coastline used in the experiment. Figure (**??**) shows the depth contours of the model domain.

3.10.3.2 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Lines 7-10 and 11-14

  ```
  tRef= 16.0 , 15.2 , 14.5 , 13.9 , 13.3 ,
  ```

  ...
  set reference values for potential temperature and salinity at each model level in units of °C and ppt. The entries are ordered from surface to depth. Density is calculated from anomalies at each level evaluated with respect to the reference values set here.

S/R INI_THETA(*ini_theta.F*)

- Line 15,

  ```
  viscAz=1.E-3,
  ```

  this line sets the vertical Laplacian dissipation coefficient to $1 \times 10^{-3} \mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later. This variable is copied into model general vertical coordinate variable **viscAr**.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 16,

  ```
  viscAh=5.E5,
  ```

  this line sets the horizontal Laplacian frictional dissipation coefficient to $5 \times 10^5 \mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later.

- Lines 17,

  ```
  no_slip_sides=.FALSE.
  ```

  this line selects a free-slip lateral boundary condition for the horizontal Laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x.

- Lines 9,

  ```
  no_slip_bottom=.TRUE.
  ```

  this line selects a no-slip boundary condition for bottom boundary condition in the vertical Laplacian friction operator e.g. $u = v = 0$ at $z = -H$, where H is the local depth of the domain.

- Line 19,

  ```
  diffKhT=1.E3,
  ```

  this line sets the horizontal diffusion coefficient for temperature to $1000\,\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries.

- Line 20,

  ```
  diffKzT=3.E-5,
  ```

  this line sets the vertical diffusion coefficient for temperature to $3{\times}10^{-5}\,\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ at both the upper and lower boundaries.

- Line 21,

  ```
  diffKhS=1.E3,
  ```

  this line sets the horizontal diffusion coefficient for salinity to $1000\,\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries.

- Line 22,

  ```
  diffKzS=3.E-5,
  ```

  this line sets the vertical diffusion coefficient for salinity to $3 \times 10^{-5}\,\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ at both the upper and lower boundaries.

- Lines 23-26

  ```
  beta=1.E-11,
  ```

...
These settings do not apply for this experiment.

- Line 27,

  ```
  gravity=9.81,
  ```

  Sets the gravitational acceleration coefficient to 9.81ms^{-1}.

S/R CALC_PHI_HYD (*calc_phi_hyd.F*)
S/R INI_CG2D (*ini_cg2d.F*)
S/R INI_CG3D (*ini_cg3d.F*)
S/R INI_PARMS (*ini_parms.F*)
S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)

- Line 28-29,

  ```
  rigidLid=.FALSE.,
  implicitFreeSurface=.TRUE.,
  ```

  Selects the barotropic pressure equation to be the implicit free surface formulation.

- Line 30,

  ```
  eosType='POLY3',
  ```

  Selects the third order polynomial form of the equation of state.

S/R FIND_RHO (*find_rho.F*)
S/R FIND_ALPHA (*find_alpha.F*)

- Line 31,

  ```
  readBinaryPrec=32,
  ```

  Sets format for reading binary input datasets holding model fields to use 32-bit representation for floating-point numbers.

S/R READ_WRITE_FLD (*read_write_fld.F*)
S/R READ_WRITE_REC (*read_write_rec.F*)

- Line 36,

  ```
  cg2dMaxIters=1000,
  ```

  Sets maximum number of iterations the two-dimensional, conjugate gradient solver will use, **irrespective of convergence criteria being met**.

S/R CG2D (*cg2d.F*)

- Line 37,

  ```
  cg2dTargetResidual=1.E-13,
  ```

  Sets the tolerance which the two-dimensional, conjugate gradient solver will use to test for convergence in equation **??** to 1×10^{-13}. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

S/R CG2D (*cg2d.F*)

- Line 42,

  ```
  startTime=0,
  ```

  Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time steps in the **startTime** value. The internal time counter works in seconds.

- Line 43,

  ```
  endTime=2808000.,
  ```

  Sets the time (in seconds) at which this simulation will terminate. At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages.

- Line 44,

  ```
  #endTime=62208000000,
  ```

  A commented out setting for endTime for a 2000 year simulation.

- Line 45,

  ```
  deltaTmom=2400.0,
  ```

  Sets the timestep δt_v used in the momentum equations to 20 mins. See section **??**.

S/R TIMESTEP(*timestep.F*)

- Line 46,

  ```
  tauCD=321428.,
  ```

Sets the D-grid to C-grid coupling time scale τ_{CD} used in the momentum equations. See section **??**.

S/R INI_PARMS(*ini_parms.F*) *S/R CALC_MOM_RHS*(*calc_mom_rhs.F*)

- Line 47,

  ```
  deltaTtracer=108000.,
  ```

  Sets the default timestep, δt_θ, for tracer equations to 30 hours. See section **??**.

S/R TIMESTEP_TRACER(*timestep_tracer.F*)

- Line 47,

  ```
  bathyFile='topog.box'
  ```

  This line specifies the name of the file from which the domain bathymetry is read. This file is a two-dimensional (x, y) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates a solid wall and a depth of $-2000m$ indicates open ocean. The matlab program *input/gendata.m* shows an example of how to generate a bathymetry file.

- Line 50,

  ```
  zonalWindFile='windx.sin_y'
  ```

  This line specifies the name of the file from which the x-direction surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program *input/gendata.m* includes example code to generate a valid **zonalWindFile** file.

other lines in the file *input/data* are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

```
1 # ====================
2 # | Model parameters |
3 # ====================
4 #
5 # Continuous equation parameters
6  &PARM01
7  tRef= 16.0 , 15.2 , 14.5 , 13.9 , 13.3 ,
```

```
 8        12.4 , 11.3 ,  9.9 ,  8.4 ,  6.7 ,
 9         5.2 ,  3.8 ,  2.9 ,  2.3 ,  1.8 ,
10         1.5 ,  1.1 ,  0.8 ,  0.66,  0.63,
11  sRef= 34.65, 34.75, 34.82, 34.87, 34.90,
12        34.90, 34.86, 34.78, 34.69, 34.60,
13        34.58, 34.62, 34.68, 34.72, 34.73,
14        34.74, 34.73, 34.73, 34.72, 34.72,
15  viscAz=1.E-3,
16  viscAh=5.E5,
17  no_slip_sides=.FALSE.,
18  no_slip_bottom=.TRUE.,
19  diffKhT=1.E3,
20  diffKzT=3.E-5,
21  diffKhS=1.E3,
22  diffKzS=3.E-5,
23  beta=1.E-11,
24  f0=1.e-4,
25  tAlpha=2.E-4,
26  sBeta =7.4E-4,
27  gravity=9.81,
28  rigidLid=.FALSE.,
29  implicitFreeSurface=.TRUE.,
30  eosType='LINEAR',
31  readBinaryPrec=32,
32  &
33
34 # Elliptic solver parameters
35  &PARM02
36  cg2dMaxIters=1000,
37  cg2dTargetResidual=1.E-13,
38  &
39
40 # Time stepping parameters
41  &PARM03
42  startTime=0,
43  endTime=622080000,
44 #endTime=2808000.,
45  deltaTmom=2400.0,
46  tauCD=321428.,
47  deltaTtracer=108000.0,
48  deltaTClock =108000.0,
49  cAdjFreq=-1.,
50  abEps=0.1,
51  pChkptFreq=0.0,
52  chkptFreq=0.0,
53  dumpFreq=2592000.0,
54  dumpFreq=31104000.,
55  tauThetaClimRelax=2592000.0,
56  tauSaltClimRelax=2592000.0,
57  &
```

```
58
59 # Gridding parameters
60  &PARM04
61  usingCartesianGrid=.FALSE.,
62  usingSphericalPolarGrid=.TRUE.,
63  delZ= 5.000000e+01, 5.000000e+01, 5.500000e+01, 6.000000e+01,
64        6.500000e+01, 7.000000e+01, 8.000000e+01, 9.500000e+01,
65        1.200000e+02, 1.550000e+02, 2.000000e+02, 2.600000e+02,
66        3.200000e+02, 4.000000e+02, 4.800000e+02, 5.700000e+02,
67        6.550000e+02, 7.250000e+02, 7.750000e+02, 8.150000e+02,
68  phiMin=-80.,
69  delY=40*4,
70  delX=90*4,
71  &
72
73 # Input datasets
74  &PARM05
75  hydrogThetaFile='theta.bin',
76  hydrogSaltFile='salt.bin',
77  bathyFile='topog.bin',
78  zonalWindFile='windx.bin',
79  meridWindFile='windy.bin',
80  thetaClimFile='SST.bin'
81  saltClimFile='SSS.bin'
82  &
```

3.10.3.3 File *input/data.pkg*

This file uses standard default values and does not contain customisations for this experiment.

3.10.3.4 File *input/eedata*

This file uses standard default values and does not contain customisations for this experiment.

3.10.3.5 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress ,τ_x, values. The units used are Nm^{-2}. Although τ_x is only a function of yn in this experiment this file must still define a complete two-dimensional map in order to be compatible with the standard code for loading forcing fields in MITgcm. The included matlab program *input/gendata.m* gives a complete code for creating the *input/windx.sin_y* file.

3.10.3.6 File *input/topog.box*

The *input/topog.box* file specifies a two-dimensional (x, y) map of depth values. For this experiment values are either $0m$ or -2000 m, corresponding respectively

to a wall or to deep ocean. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays. The included matlab program *input/gendata.m* gives a complete code for creating the *input/topog.box* file.

3.10.3.7 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=4,
  ```

  this line sets the vertical domain extent in grid points.

```
 1 C $Header: /u/gcmpack/manual/part3/case_studies/climatalogical_ogcm/code/SIZE.h.tex,v 1.1
 2 C $Name:  $
 3 C
 4 C     /==========================================================\
 5 C     | SIZE.h Declare size of underlying computational grid.    |
 6 C     |==========================================================|
 7 C     | The design here support a three-dimensional model grid   |
 8 C     | with indices I,J and K. The three-dimensional domain     |
 9 C     | is comprised of nPx*nSx blocks of size sNx along one axis|
10 C     | nPy*nSy blocks of size sNy along another axis and one    |
11 C     | block of size Nz along the final axis.                   |
12 C     | Blocks have overlap regions of size OLx and OLy along the|
13 C     | dimensions that are subdivided.                          |
14 C     \==========================================================/
15 C     Voodoo numbers controlling data layout.
16 C     sNx - No. X points in sub-grid.
17 C     sNy - No. Y points in sub-grid.
18 C     OLx - Overlap extent in X.
19 C     OLy - Overlat extent in Y.
20 C     nSx - No. sub-grids in X.
```

```
21 C     nSy - No. sub-grids in Y.
22 C     nPx - No. of processes to use in X.
23 C     nPy - No. of processes to use in Y.
24 C     Nx  - No. points in X for the total domain.
25 C     Ny  - No. points in Y for the total domain.
26 C     Nr  - No. points in Z for full process domain.
27       INTEGER sNx
28       INTEGER sNy
29       INTEGER OLx
30       INTEGER OLy
31       INTEGER nSx
32       INTEGER nSy
33       INTEGER nPx
34       INTEGER nPy
35       INTEGER Nx
36       INTEGER Ny
37       INTEGER Nr
38       PARAMETER (
39      &           sNx =  90,
40      &           sNy =  40,
41      &           OLx =   3,
42      &           OLy =   3,
43      &           nSx =   1,
44      &           nSy =   1,
45      &           nPx =   1,
46      &           nPy =   1,
47      &           Nx  = sNx*nSx*nPx,
48      &           Ny  = sNy*nSy*nPy,
49      &           Nr  =  20)

50 C     MAX_OLX  - Set to the maximum overlap region size of any array
51 C     MAX_OLY    that will be exchanged. Controls the sizing of exch
52 C                routine buufers.
53       INTEGER MAX_OLX
54       INTEGER MAX_OLY
55       PARAMETER ( MAX_OLX = OLx,
56      &            MAX_OLY = OLy )
```

3.10.3.8 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.10.3.9 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.10.3.10 Other Files

Other files relevant to this experiment are

- *model/src/ini_cori.F*. This file initializes the model coriolis variables **fCorU**.
- *model/src/ini_spherical_polar_grid.F*
- *model/src/ini_parms.F*,
- *input/windx.sin_y*,

contain the code customisations and parameter settings for this experiments. Below we describe the customisations to these files associated with this experiment.

3.11 Global Ocean Simulation at 4° Resolution in Pressure Coordinates

This example experiment demonstrates using the MITgcm to simulate the planetary ocean circulation in pressure coordinates, that is, without making the Boussinesq approximations. The simulation is configured with realistic geography and bathymetry on a $4^\circ \times 4^\circ$ spherical polar grid. Fifteen levels are used in the vertical, ranging in thickness from 50.4089 dbar $\approx$ 50 m at the surface to 710.33 dbar $\approx$ 690 m at depth, giving a maximum model depth of 5302.3122 dbar $\approx$ 5200 km. At this resolution, the configuration can be integrated forward for thousands of years on a single processor desktop computer.

3.11.1 Overview

The model is forced with climatological wind stress data from Trenberth [50] and surface flux data from Jiang et al. [34]. Climatological data from Levitus [36] is used to initialize the model hydrography. Levitus seasonal climatology data is also used throughout the calculation to provide additional air-sea fluxes. These fluxes are combined with the Jiang climatological estimates of surface heat flux, resulting in a mixed boundary condition of the style described in Haney [25]. Altogether, this yields the following forcing applied in the model surface layer.

$$\mathcal{F}_u = g\frac{\tau_x}{\Delta p_s} \tag{3.50}$$

$$\mathcal{F}_v = g\frac{\tau_y}{\Delta p_s} \tag{3.51}$$

$$\mathcal{F}_\theta = -g\lambda_\theta(\theta - \theta^*) - \frac{1}{C_p \Delta p_s}\mathcal{Q} \tag{3.52}$$

$$\mathcal{F}_s = +g\rho_{FW}\frac{S}{\rho \Delta p_s}(\mathcal{E} - \mathcal{P} - \mathcal{R}) \tag{3.53}$$

where $\mathcal{F}_u$, $\mathcal{F}_v$, $\mathcal{F}_\theta$, $\mathcal{F}_s$ are the forcing terms in the zonal and meridional momentum and in the potential temperature and salinity equations respectively. The term Δp_s represents the top ocean layer thickness in Pa. It is used in conjunction with a reference density, ρ_{FW} (here set to 999.8 kg m^{-3}), the surface salinity, S, and a specific heat capacity, C_p (here set to 4000 J °C^{-1} kg^{-1}), to convert input dataset values into time tendencies of potential temperature (with units of °C s^{-1}), salinity (with units ppt s^{-1}) and velocity (with units m s^{-2}). The externally supplied forcing fields used in this experiment are τ_x, τ_y, θ^*, $\mathcal{Q}$ and $\mathcal{E} - \mathcal{P} - \mathcal{R}$. The wind stress fields (τ_x, τ_y) have units of N m^{-2}. The temperature forcing fields (θ^* and Q) have units of °C and W m^{-2} respectively. The salinity forcing fields ($\mathcal{E} - \mathcal{P} - \mathcal{R}$) has units of m s^{-1} respectively. The source files and procedures for ingesting these data into the simulation are described in the experiment configuration discussion in section 3.10.3.

3.11.2 Discrete Numerical Configuration

Due to the pressure coordinate, the model can only be hydrostatic [12]. The domain is discretized with a uniform grid spacing in latitude and longitude on the sphere $\Delta\phi = \Delta\lambda = 4°$, so that there are ninety grid cells in the zonal and forty in the meridional direction. The internal model coordinate variables x and y are initialized according to

$$x = r\cos(\phi),\ \Delta x = r\cos(\Delta\phi) \tag{3.54}$$

$$y = r\lambda,\ \Delta y = r\Delta\lambda \tag{3.55}$$

Arctic polar regions are not included in this experiment. Meridionally the model extends from 80°S to 80°N. Vertically the model is configured with fifteen layers with the following thicknesses

$$\begin{aligned}
\Delta p_1 &= 7103300.720021\,\mathrm{Pa},\\
\Delta p_2 &= 6570548.440790\,\mathrm{Pa},\\
\Delta p_3 &= 6041670.010249\,\mathrm{Pa},\\
\Delta p_4 &= 5516436.666057\,\mathrm{Pa},\\
\Delta p_5 &= 4994602.034410\,\mathrm{Pa},\\
\Delta p_6 &= 4475903.435290\,\mathrm{Pa},\\
\Delta p_7 &= 3960063.245801\,\mathrm{Pa},\\
\Delta p_8 &= 3446790.312651\,\mathrm{Pa},\\
\Delta p_9 &= 2935781.405664\,\mathrm{Pa},\\
\Delta p_{10} &= 2426722.705046\,\mathrm{Pa},\\
\Delta p_{11} &= 1919291.315988\,\mathrm{Pa},\\
\Delta p_{12} &= 1413156.804970\,\mathrm{Pa},\\
\Delta p_{13} &= 1008846.750166\,\mathrm{Pa},\\
\Delta p_{14} &= 705919.025481\,\mathrm{Pa},\\
\Delta p_{15} &= 504089.693499\,\mathrm{Pa},
\end{aligned}$$

(here the numeric subscript indicates the model level index number, `k`; note, that the surface layer has the highest index number 15) to give a total depth, H, of -5200m. In pressure, this is $p_b^0 = 53023122.566084$Pa. The implicit free surface form of the pressure equation described in Marshall et al. [39] with the nonlinear extension by Campin et al. [8] is employed. A Laplacian operator, ∇^2, provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress forcing is added to the momentum equations in (3.56) for both the zonal flow, u and the meridional flow v, according to equations (3.50) and (3.51). Thermodynamic forcing inputs are added to the equations in (3.56) for potential temperature, θ, and salinity, S, according to equations (3.52) and (3.53). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial \Phi^{'}}{\partial x} - \nabla_h \cdot A_h \nabla_h u - (g\rho_0)^2 \frac{\partial}{\partial p} A_r \frac{\partial u}{\partial p} = \begin{cases} \mathcal{F}_u & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.56}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial \Phi^{'}}{\partial y} - \nabla_h \cdot A_h \nabla_h v - (g\rho_0)^2 \frac{\partial}{\partial p} A_r \frac{\partial v}{\partial p} = \begin{cases} \mathcal{F}_v & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.57}$$

$$\frac{\partial p_b}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{3.58}$$

$$\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - (g\rho_0)^2 \frac{\partial}{\partial p} \Gamma(K_r) \frac{\partial \theta}{\partial p} = \begin{cases} \mathcal{F}_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.59}$$

$$\frac{Ds}{Dt} - \nabla_h \cdot K_h \nabla_h s - (g\rho_0)^2 \frac{\partial}{\partial p} \Gamma(K_r) \frac{\partial S}{\partial p} = \begin{cases} \mathcal{F}_s & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.60}$$

$$\Phi'^{(0)}_{-H} + \alpha_0 p_b + \int_0^p \alpha' dp = \Phi' \tag{3.61}$$

where $u = \frac{Dx}{Dt} = r\cos(\phi)\frac{D\lambda}{Dt}$ and $v = \frac{Dy}{Dt} = r\frac{D\phi}{Dt}$ are the zonal and meridional components of the flow vector, $\vec{u}$, on the sphere. As described in MITgcm Numerical Solution Procedure 2, the time evolution of potential temperature, θ, equation is solved prognostically. The full geopotential height, Φ, is diagnosed by summing the geopotential height anomalies Φ' due to bottom pressure p_b and density variations. The integration of the hydrostatic equation is started at the bottom of the domain. The condition of $p = 0$ at the sea surface requires a time-independent integration constant for the height anomaly due to density variations $\Phi'^{(0)}_{-H}$, which is provided as an input field.

3.11.3 Experiment Configuration

The model configuration for this experiment resides under the directory *tutorial_examples/global_ocean_circulation/*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/topog.bin*,
- *input/deltageopotjmd95.bin*,
- *input/lev_s.bin*,
- *input/lev_t.bin*,
- *input/trenberth_taux.bin*,

- *input/trenberth_tauy.bin*,
- *input/lev_sst.bin*,
- *input/shi_qnet.bin*,
- *input/shi_empmr.bin*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

3.11.3.1 Driving Datasets

Figures (3.3-3.8) show the relaxation temperature (θ^*) and salinity (S^*) fields, the wind stress components (τ_x and τ_y), the heat flux (Q) and the net fresh water flux ($\mathcal{E} - \mathcal{P} - \mathcal{R}$) used in equations 3.50-3.53. The figures also indicate the lateral extent and coastline used in the experiment. Figure (3.9) shows the depth contours of the model domain.

3.11.3.2 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Line 15,

  ```
  viscAr=1.721611620915750E+05,
  ```

  this line sets the vertical Laplacian dissipation coefficient to $1.72161162091575 \times 10^5 \mathrm{Pa}^2\mathrm{s}^{-1}$. Note that, the factor $(g\rho)^2$ needs to be included in this line. Boundary conditions for this operator are specified later. This variable is copied into model general vertical coordinate variable **viscAr**.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 9–10,

  ```
  viscAh=3.E5,
  no_slip_sides=.TRUE.
  ```

  these lines set the horizontal Laplacian frictional dissipation coefficient to $3 \times 10^5 \mathrm{m}^2\mathrm{s}^{-1}$ and specify a no-slip boundary condition for this operator, that is, $u = 0$ along boundaries in y and $v = 0$ along boundaries in x.

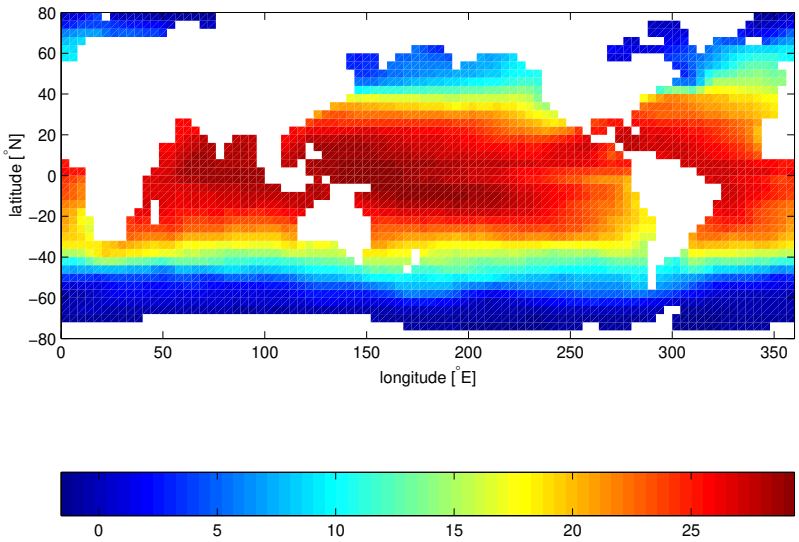


Figure 3.3: Annual mean of relaxation temperature [°C]

- Lines 11-13,

```
 viscAr =1.721611620915750e5,
#viscAz =1.67E-3,
 no_slip_bottom=.FALSE.,
```

  These lines set the vertical Laplacian frictional dissipation coefficient to $1.721611620915750 \times 10^5\,\mathrm{Pa^2 s^{-1}}$, which corresponds to $1.67 \times 10^{-3}\,\mathrm{m^2 s^{-1}}$ in the commented line, and specify a free slip boundary condition for this operator, that is, $\frac{\partial u}{\partial p} = \frac{\partial v}{\partial p} = 0$ at $p = p_b^0$, where p_b^0 is the local bottom pressure of the domain at rest. Note that, the factor $(g\rho)^2$ needs to be included in this line.

- Line 14,

```
 diffKhT=1.E3,
```

  this line sets the horizontal diffusion coefficient for temperature to $1000\,\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries.

- Line 15–16,

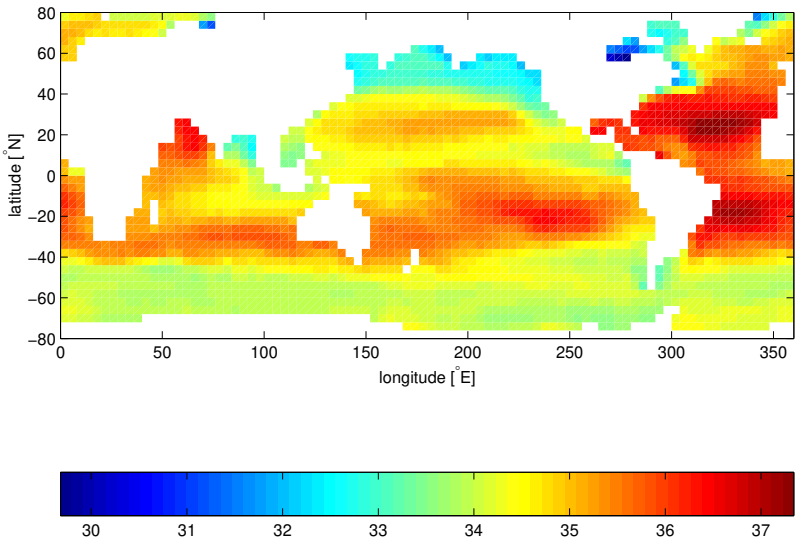


Figure 3.4: Annual mean of relaxation salinity [PSU]

```
 diffKrT=5.154525811125000e3,
#diffKzT=0.5E-4,
```

this line sets the vertical diffusion coefficient for temperature to $5.154525811125 \times 10^3\,\mathrm{Pa^2 s^{-1}}$, which corresponds to $5 \times 10^{-4}\,\mathrm{m^2 s^{-1}}$ in the commented line. Note that, the factor $(g\rho)^2$ needs to be included in this line. The boundary condition on this operator is $\frac{\partial}{\partial p} = 0$ at both the upper and lower boundaries.

- Line 17–19,

```
 diffKhS=1.E3,
 diffKrS=5.154525811125000e3,
#diffKzS=0.5E-4,
```

These lines set the same values for the diffusion coefficients for salinity as for temperature.

- Line 20–22,

```
 implicitDiffusion=.TRUE.,
 ivdc_kappa=1.030905162225000E9,
#ivdc_kappa=10.0,
```

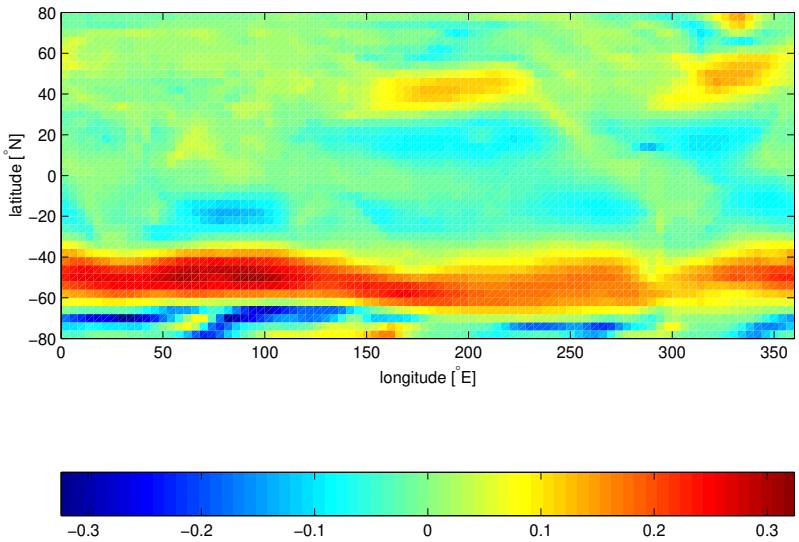


Figure 3.5: Annual mean of zonal wind stress component [$\mathrm{Nm\,m^{-2}}$]

Select implicit diffusion as a convection scheme and set coefficient for implicit vertical diffusion to $1.030905162225 \times 10^9\,\mathrm{Pa^2 s^{-1}}$, which corresponds to $10\,\mathrm{m^2\,s^{-1}}$.

- Line 23-24,

```
gravity=9.81,
gravitySign=-1.D0,
```

These lines set the gravitational acceleration coefficient to $9.81\mathrm{ms}^{-1}$ and define the upward direction relative to the direction of increasing vertical coordinate (in pressure coordinates, up is in the direction of decreasing pressure)

- Line 25,

```
rhoNil=1035.,
```

sets the reference density of sea water to $1035\,\mathrm{kg\,m^{-3}}$.

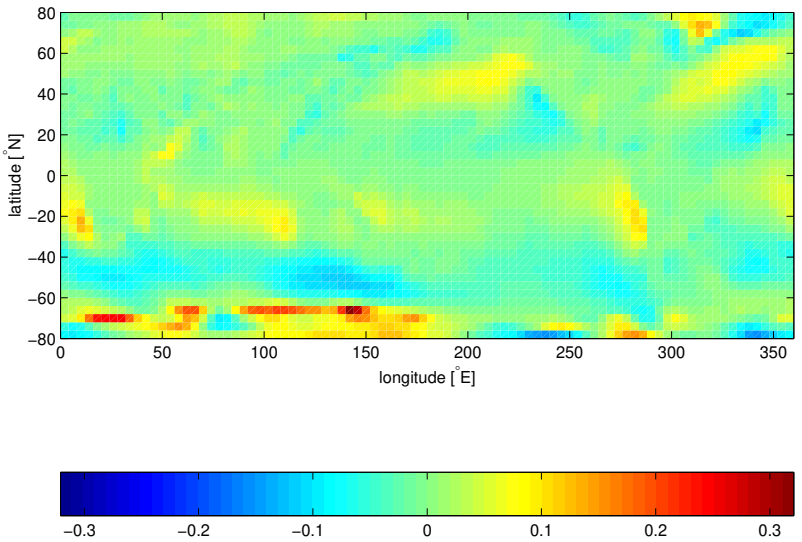


Figure 3.6: Annual mean of meridional wind stress component [$\mathrm{Nm\,m^{-2}}$]

S/R CALC_PHI_HYD (*calc_phi_hyd.F*)
S/R INI_CG2D (*ini_cg2d.F*)
S/R INI_CG3D (*ini_cg3d.F*)
S/R INI_PARMS (*ini_parms.F*)
S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)

- Line 28

```
eosType='JMD95P',
```

Selects the full equation of state according to Jackett and McDougall [33]. The only other sensible choice is the equation of state by McDougall et al. [41], 'MDJFW'. All other equations of state do not make sense in this configuration.

S/R FIND_RHO (*find_rho.F*)
S/R FIND_ALPHA (*find_alpha.F*)

- Line 28-29,

```
rigidLid=.FALSE.,
implicitFreeSurface=.TRUE.,
```

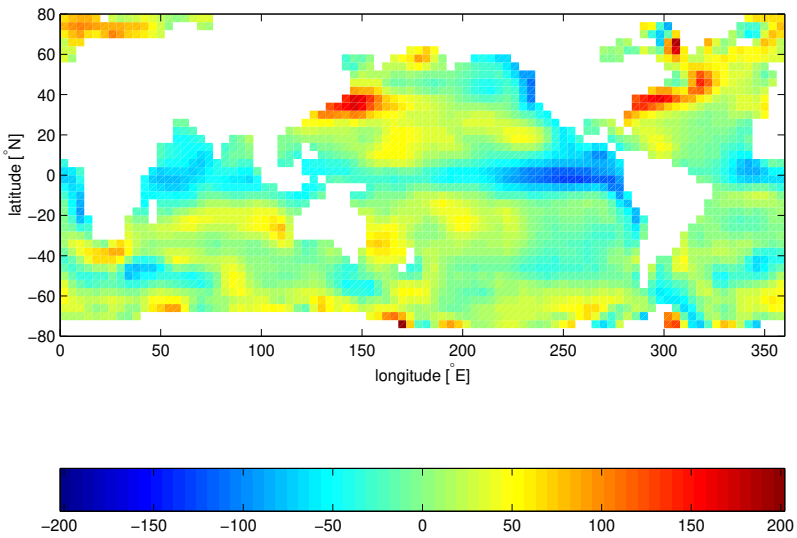


Figure 3.7: Annual mean heat flux [$\mathrm{W\,m^{-2}}$]

Selects the barotropic pressure equation to be the implicit free surface formulation.

- Line 30

```
exactConserv=.TRUE.,
```

Select a more accurate conservation of properties at the surface layer by including the horizontal velocity divergence to update the free surface.

- Line 31–33

```
nonlinFreeSurf=3,
hFacInf=0.2,
hFacSup=2.0,
```

Select the nonlinear free surface formulation and set lower and upper limits for the free surface excursions.

- Line 34

```
useRealFreshWaterFlux=.FALSE.,
```

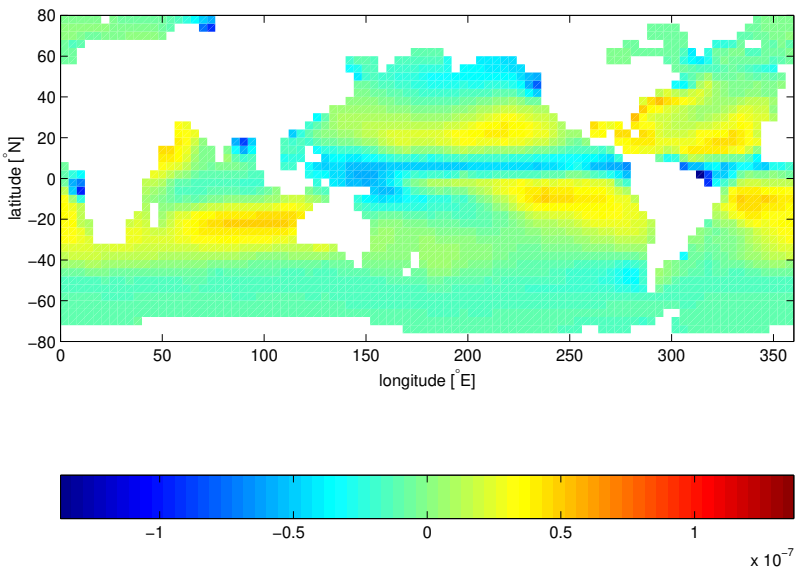


Figure 3.8: Annual mean fresh water flux (Evaporation-Precipitation) [$\mathrm{m\,s^{-1}}$]

Select virtual salt flux boundary condition for salinity. The freshwater flux at the surface only affect the surface salinity, but has no mass flux associated with it

- Line 35–36,

```
readBinaryPrec=64,
writeBinaryPrec=64,
```

Sets format for reading binary input datasets and writing binary output datasets holding model fields to use 64-bit representation for floating-point numbers.

S/R READ_WRITE_FLD (*read_write_fld.F*) *S/R READ_WRITE_REC* (*read_write_rec.F*)

- Line 42,

```
cg2dMaxIters=200,
```

Sets maximum number of iterations the two-dimensional, conjugate gradient solver will use, **irrespective of convergence criteria being met**.

S/R CG2D (*cg2d.F*)

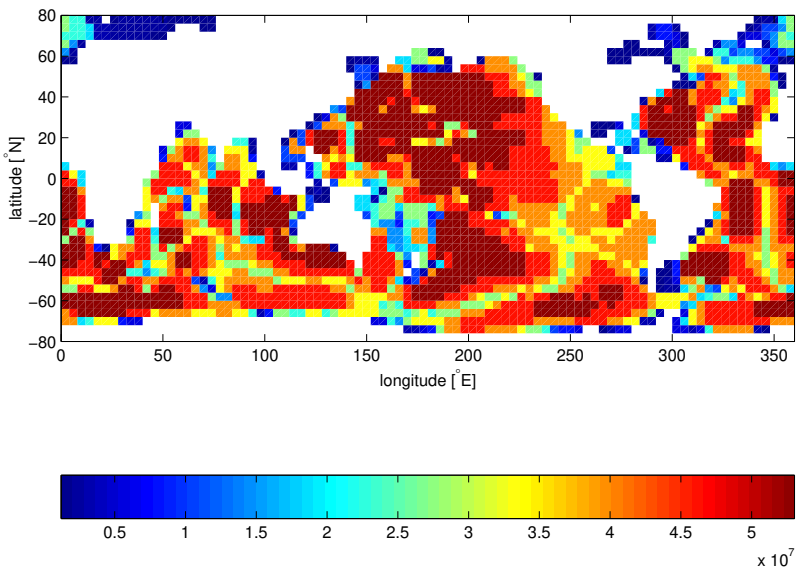


Figure 3.9: Model bathymetry in pressure units [Pa]

- Line 43,

  `cg2dTargetResidual=1.E-13,`

  Sets the tolerance which the two-dimensional, conjugate gradient solver will use to test for convergence in equation **??** to 1×10^{-9}. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

S/R CG2D (*cg2d.F*)

- Line 48,

  `startTime=0,`

  Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time steps in the **startTime** value. The internal time counter works in seconds.

- Line 49–50,

```
 endTime=8640000.,
#endTime=62208000000,
```

Sets the time (in seconds) at which this simulation will terminate. At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages. The commented out setting for endTime is for a 2000 year simulation.

- Line 51–53,

```
deltaTmom      =   1200.0,
deltaTtracer   = 172800.0,
deltaTfreesurf = 172800.0,
```

Sets the timestep δt_v used in the momentum equations to 20 mins and the timesteps δt_θ in the tracer equations and δt_η in the implicit free surface equation to 48 hours. See section **??**.

S/R TIMESTEP(*timestep.F*)
S/R INI_PARMS(*ini_parms.F*)
S/R CALC_MOM_RHS(*calc_mom_rhs.F*)
S/R TIMESTEP_TRACER(*timestep_tracer.F*)

- Line 55,

```
pChkptFreq  =3110400000.,
```

write a pick-up file every 100 years of integration.

- Line 56–58

```
dumpFreq    = 3110400000.,
taveFreq    = 3110400000.,
monitorFreq =   31104000.,
```

write model output and time-averaged model output every 100 years, and monitor statisitics every year.

- Line 59–61

```
periodicExternalForcing=.TRUE.,
externForcingPeriod=2592000.,
externForcingCycle=31104000.,
```

Allow periodic external forcing, set forcing period, during which one set of data is valid, to 1 month and the repeat cycle to 1 year.

S/R EXTERNAL_FORCING_SURF(*external_forcing_surf.F*)

- Line 62

```
tauThetaClimRelax=5184000.0,
```

Set the restoring timescale to 2 months.

S/R EXTERNAL_FORCING_SURF(*external_forcing_surf.F*)

- Line 63

```
abEps=0.1,
```

Adams-Bashford factor (see section 2.4)

- Line 68–69

```
usingCartesianGrid=.FALSE.,
usingSphericalPolarGrid=.TRUE.,
```

Select spherical grid.

- Line 70–71

```
dXspacing=4.,
dYspacing=4.,
```

Set the horizontal grid spacing in degrees spherical distance.

- Line 72

```
Ro_SeaLevel=53023122.566084,
```

specifies the total height (in r-units, i.e., pressure units) of the sea surface at rest. This is a reference value.

- Line 73

```
groundAtK1=.TRUE.,
```

specifies the reversal of the vertical indexing. The vertical index is 1 at the bottom of the doman and maximal (i.e., 15) at the surface.

- Line 74–78

```
delR=7103300.720021, \ldots
```

set the layer thickness in pressure units, starting with the bottom layer.

- Line 84–93,

```
 bathyFile='topog.box'
 ploadFile='deltageopotjmd95.bin'
 hydrogThetaFile='lev_t.bin',
 hydrogSaltFile ='lev_s.bin',
 zonalWindFile  ='trenberth_taux.bin',
 meridWindFile  ='trenberth_tauy.bin',
 thetaClimFile  ='lev_sst.bin',
 surfQFile      ='shi_qnet.bin',
 EmPmRFile      ='shi_empmr.bin',
```

This line specifies the names of the files holding the bathymetry data set, the time-independent geopotential height anomaly at the bottom, initial conditions of temperature and salinity, wind stress forcing fields, sea surface temperature climatology, heat flux, and fresh water flux (evaporation minus precipitation minus run-off) at the surface. See file descriptions in section 3.11.3.

other lines in the file *input/data* are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

```
 1 # ====================
 2 # | Model parameters |
 3 # ====================
 4 #
 5 # Continuous equation parameters
 6  &PARM01
 7  tRef=15*20.,
 8  sRef=15*35.,
 9  viscAh =3.E5,
10  no_slip_sides=.TRUE.,
11  viscAr =1.721611620915750e5,
12 #viscAz =1.67E-3,
13  no_slip_bottom=.FALSE.,
14  diffKhT=1.E3,
15  diffKrT=5.154525811125000e3,
16 #diffKzT=0.5E-4,
17  diffKhS=1.E3,
18  diffKrS=5.154525811125000e3,
19 #diffKzS=0.5E-4,
20  implicitDiffusion=.TRUE.,
21  ivdc_kappa=1.030905162225000e9,
22 #ivdc_kappa=10.0,
23  gravity=9.81,
24  gravitySign=-1.D0,
25  rhonil=1035.,
26  buoyancyRelation='OCEANICP',
27  eosType='JMD95P',
28  rigidLid=.FALSE.,
29  implicitFreeSurface=.TRUE.,
```

```
 exactConserv=.TRUE.,
 nonlinFreeSurf=3,
 hFacInf=0.2,
 hFacSup=2.0,
 useRealFreshWaterFlux=.FALSE.,
 readBinaryPrec=64,
 writeBinaryPrec=64,
 cosPower=0.5,
 &

# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=200,
 cg2dTargetResidual=1.E-9,
 &

# Time stepping parameters
 &PARM03
 startTime =           0.,
 endTime   =     8640000.,
#endTime   = 62208000000.,
 deltaTmom      =   1200.0,
 deltaTtracer   = 172800.0,
 deltaTfreesurf = 172800.0,
 deltaTClock    = 172800.0,
 pChkptFreq  = 3110400000.,
 dumpFreq    = 3110400000.,
 taveFreq    = 3110400000.,
 monitorFreq =   31104000.,
 periodicExternalForcing=.TRUE.,
 externForcingPeriod=2592000.,
 externForcingCycle=31104000.,
 tauThetaClimRelax=5184000.0,
 abEps=0.1,
 &

# Gridding parameters
 &PARM04
 usingCartesianGrid=.FALSE.,
 usingSphericalPolarGrid=.TRUE.,
 dXspacing=4.,
 dYspacing=4.,
 Ro_SeaLevel=53023122.566084,
 groundAtK1=.TRUE.,
 delR=7103300.720021, 6570548.440790, 6041670.010249,
      5516436.666057, 4994602.034410, 4475903.435290,
      3960063.245801, 3446790.312651, 2935781.405664,
      2426722.705046, 1919291.315988, 1413156.804970,
      1008846.750166,  705919.025481,  504089.693499,
 phiMin=-80.,
```

```
 80  &
 81
 82 # Input datasets
 83  &PARM05
 84  topoFile       ='topog.bin',
 85  pLoadFile      ='deltageopotjmd95.bin',
 86  hydrogThetaFile='lev_t.bin',
 87  hydrogSaltFile ='lev_s.bin',
 88  zonalWindFile  ='trenberth_taux.bin',
 89  meridWindFile  ='trenberth_tauy.bin',
 90  thetaClimFile  ='lev_sst.bin',
 91 #saltClimFile   ='lev_sss.bin',
 92  surfQFile      ='shi_qnet.bin',
 93  EmPmRFile      ='shi_empmr.bin',
 94  &
```

3.11.3.3 File *input/data.pkg*

This file uses standard default values and does not contain customisations for this experiment.

3.11.3.4 File *input/eedata*

This file uses standard default values and does not contain customisations for this experiment.

3.11.3.5 File *input/topog.bin*

This file is a two-dimensional (x, y) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code (Pa for this experiment). In this experiment, a depth of 0 Pa indicates a land point wall and a depth of > 0 Pa indicates open ocean.

3.11.3.6 File *input/deltageopotjmd95.box*

The file contains 12 identical two dimensional maps (x, y) of geopotential height anomaly at the bottom at rest. The values have been obtained by vertically integrating the hydrostatic equation with the initial density field (from *input/lev_t/s.bin*). This file has to be consitent with the temperature and salinity field at rest and the choice of equation of state!

3.11.3.7 File *input/lev_t/s.bin*

The files *input/lev_t/s.bin* specify the initial conditions for temperature and salinity for every grid point in a three dimensional array (x, y, z). The data are

obtain by interpolating Levitus [36] monthly mean values for January onto the model grid. Keep in mind, that the first index corresponds to the bottom layer and highest index to the surface layer.

3.11.3.8 File *input/trenberth_taux/y.bin*

Each of the *input/trenberth_taux/y.bin* files specifies 12 two-dimensional (x, y, t) maps of zonal and meridional wind stress values, τ_x and τ_y, that is monthly mean values from Trenberth [50]. The units used are Nm^{-2}.

3.11.3.9 File *input/lev_sst.bin*

The file *input/lev_sst.bin* contains 12 monthly surface temperature climatologies from Levitus [36] in a three dimensional array (x, y, t).

3.11.3.10 File *input/shi_qnet/empmr.bin*

The files *input/shi_qnet/empmr.bin* contain 12 monthly surface fluxes of heat (qnet) and freshwater (empmr) by Jiang et al. [34] in three dimensional arrays (x, y, t). Both fluxes are normalized so that of one year there is no net flux into the ocean. The freshwater flux is actually constant in time.

3.11.3.11 File *code/SIZE.h*

Three lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=90,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=40,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=15,
  ```

  this line sets the vertical domain extent in grid points.

```
 2 C $Name:  $
 3 C
 4 C     /==========================================================\
 5 C     | SIZE.h Declare size of underlying computational grid.    |
 6 C     |==========================================================|
 7 C     | The design here support a three-dimensional model grid   |
 8 C     | with indices I,J and K. The three-dimensional domain     |
 9 C     | is comprised of nPx*nSx blocks of size sNx along one axis|
10 C     | nPy*nSy blocks of size sNy along another axis and one    |
11 C     | block of size Nz along the final axis.                   |
12 C     | Blocks have overlap regions of size OLx and OLy along the|
13 C     | dimensions that are subdivided.                          |
14 C     \==========================================================/
15 C     Voodoo numbers controlling data layout.
16 C     sNx - No. X points in sub-grid.
17 C     sNy - No. Y points in sub-grid.
18 C     OLx - Overlap extent in X.
19 C     OLy - Overlat extent in Y.
20 C     nSx - No. sub-grids in X.
21 C     nSy - No. sub-grids in Y.
22 C     nPx - No. of processes to use in X.
23 C     nPy - No. of processes to use in Y.
24 C     Nx  - No. points in X for the total domain.
25 C     Ny  - No. points in Y for the total domain.
26 C     Nr  - No. points in Z for full process domain.
27       INTEGER sNx
28       INTEGER sNy
29       INTEGER OLx
30       INTEGER OLy
31       INTEGER nSx
32       INTEGER nSy
33       INTEGER nPx
34       INTEGER nPy
35       INTEGER Nx
36       INTEGER Ny
37       INTEGER Nr
38       PARAMETER (
39      &           sNx =  90,
40      &           sNy =  40,
41      &           OLx =   3,
42      &           OLy =   3,
43      &           nSx =   1,
44      &           nSy =   1,
45      &           nPx =   1,
46      &           nPy =   1,
47      &           Nx  = sNx*nSx*nPx,
48      &           Ny  = sNy*nSy*nPy,
49      &           Nr  =  15)

50 C     MAX_OLX  - Set to the maximum overlap region size of any array
```

```
51 C     MAX_OLY    that will be exchanged. Controls the sizing of exch
52 C                routine buufers.
53       INTEGER MAX_OLX
54       INTEGER MAX_OLY
55       PARAMETER ( MAX_OLX = OLx,
56      &            MAX_OLY = OLy )
```

3.11.3.12 File *code/CPP_OPTIONS.h*

This file uses mostly standard default values except for:

- `#define ATMOSPHERIC_LOADING`
  enable pressure loading which is abused to include the initial geopotential height anomaly
- `#define EXACT_CONSERV`
  enable more accurate conservation properties to include the horizontal mass divergence in the free surface
- `#define NONLIN_FRSURF`
  enable the nonlinear free surface

3.11.3.13 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.12 Held-Suarez forcing atmospheric simulation on a latitude-longitude grid 2.8° resolution and on a cube-sphere grid with 32 square cube faces.

This example illustrates the use of the MITgcm for large scale atmospheric circulation simulation. Two simulations are described

- global atmospheric circulation on a latitude-longitude grid and
- global atmospheric circulation on a cube-sphere grid

The examples show how to use the isomorphic 'p-coordinate' scheme in MITgcm to enable atmospheric simulation.

3.12.1 Overview

This example demonstrates using the MITgcm to simulate the planetary atmospheric circulation in two ways. In both cases the simulation is configured with flat orography. In the first case shown a $2.8^\circ \times 2.8^\circ$ spherical polar horizontal grid is employed. In the second case a cube-sphere horizontal grid is used that projects a cube with face size of 32×32 onto a sphere. Five pressure corrdinate levels are used in the vertical, ranging in thickness from 100 mb at the bottom of the atmosphere to 300 mb in the middle atmosphere. The total depth of the atmosphere is 1000mb. At this resolution, the configuration can be integrated forward for many years on a single processor desktop computer.

The model is forced by relaxation to a radiative equilibrium profile from Held and Suarez [26]. Initial conditions are a statically stable thermal gradient and no motion. The atmosphere in these experiments is dry and the only active "physics" are the terms in the Held and Suarez [26] formula. The MITgcm intermediate atmospheric physics package (see 6.10) and MITgcm high-end physics package (see **??**) are turned off. Altogether, this yields the following forcing (from Held and Suarez [26]) that is applied to the fluid:

$$\vec{\mathcal{F}_u} = -k_v(p)\vec{u} \tag{3.62}$$
$$\mathcal{F}_\theta = -k_T(\phi,p)[\theta - \theta_{eq}(\phi,p)] \tag{3.63}$$
$$\tag{3.64}$$

where $\vec{\mathcal{F}}_u$, $\mathcal{F}_\theta$, are the forcing terms in the zonal and meridional momentum and in the potential temperature equations respectively. The term k_v in equation (3.62) applies a linear frictional drag (Rayleigh damping) that is active within the planetary boundary layer. It is defined so as to decay with height according

to

$$k_v = k_f \max(0, (p_{\rm k}/p_s^0 - \sigma_b)/(1-\sigma_b)) \quad (3.65)$$
$$\sigma_b = 0.7 \quad (3.66)$$
$$k_f = 1{\rm day}^{-1} \quad (3.67)$$

where $p_{\rm k}$ is the pressure level of the cell center for level k and p_s^0 is the pressure at the base of the atmospheric column.

3.12.2 Discrete Numerical Configuration

The model is configured in hydrostatic form. The domain is discretised with a uniform grid spacing in latitude and longitude on the sphere $\Delta\phi = \Delta\lambda = 4^\circ$, so that there are ninety grid cells in the zonal and forty in the meridional direction. The internal model coordinate variables x and y are initialized according to

$$x = r\cos(\phi),\ \Delta x = r\cos(\Delta\phi) \quad (3.68)$$
$$y = r\lambda,\ \Delta x = r\Delta\lambda \quad (3.69)$$

Arctic polar regions are not included in this experiment. Meridionally the model extends from 80°S to 80°N. Vertically the model is configured with twenty layers with the following thicknesses $\Delta z_1 = 50\,{\rm m}$, $\Delta z_2 = 50\,{\rm m}$, $\Delta z_3 = 55\,{\rm m}$, $\Delta z_4 = 60\,{\rm m}$, $\Delta z_5 = 65\,{\rm m}$, $\Delta z_6 = 70\,{\rm m}$, $\Delta z_7 = 80\,{\rm m}$, $\Delta z_8 = 95\,{\rm m}$, $\Delta z_9 = 120\,{\rm m}$, $\Delta z_{10} = 155\,{\rm m}$, $\Delta z_{11} = 200\,{\rm m}$, $\Delta z_{12} = 260\,{\rm m}$, $\Delta z_{13} = 320\,{\rm m}$, $\Delta z_{14} = 400\,{\rm m}$, $\Delta z_{15} = 480\,{\rm m}$, $\Delta z_{16} = 570\,{\rm m}$, $\Delta z_{17} = 655\,{\rm m}$, $\Delta z_{18} = 725\,{\rm m}$, $\Delta z_{19} = 775\,{\rm m}$, $\Delta z_{20} = 815\,{\rm m}$ (here the numeric subscript indicates the model level index number, `k`). The implicit free surface form of the pressure equation described in Marshall et. al [39] is employed. A Laplacian operator, ∇^2, provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress forcing is added to the momentum equations for both the zonal flow, u and the meridional flow v, according to equations (3.62) and (**??**). Thermodynamic forcing inputs are added to the equations for potential temperature, θ, and salinity, S, according to equations (3.63) and (**??**). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p^{'}}{\partial x} - \nabla_h \cdot A_h \nabla_h u - \frac{\partial}{\partial z} A_z \frac{\partial u}{\partial z} = \begin{cases} \mathcal{F}_u & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.70)$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p^{'}}{\partial y} - \nabla_h \cdot A_h \nabla_h v - \frac{\partial}{\partial z} A_z \frac{\partial v}{\partial z} = \begin{cases} \mathcal{F}_v & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.71)$$

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = 0 \quad (3.72)$$

$$\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - \frac{\partial}{\partial z}\Gamma(K_z)\frac{\partial \theta}{\partial z} = \begin{cases} \mathcal{F}_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.73)$$

$$\frac{Ds}{Dt} - \nabla_h \cdot K_h \nabla_h s - \frac{\partial}{\partial z}\Gamma(K_z)\frac{\partial s}{\partial z} = \begin{cases} \mathcal{F}_s & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.74)$$

$$g\rho_0 \eta + \int_{-z}^{0} \rho^{'} dz = p^{'} \quad (3.75)$$

where $u = \frac{Dx}{Dt} = r\cos(\phi)\frac{D\lambda}{Dt}$ and $v = \frac{Dy}{Dt} = r\frac{D\phi}{Dt}$ are the zonal and meridional components of the flow vector, $\vec{u}$, on the sphere. As described in MITgcm Numerical Solution Procedure 2, the time evolution of potential temperature, θ, equation is solved prognostically. The total pressure, p, is diagnosed by summing pressure due to surface elevation η and the hydrostatic pressure.

3.12.2.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, A_h, is set to $5 \times 10^5 ms^{-1}$. This value is chosen to yield a Munk layer width [1],

$$M_w = \pi(\frac{A_h}{\beta})^{\frac{1}{3}} \quad (3.76)$$

of $\approx$ 600km. This is greater than the model resolution in low-latitudes, $\Delta x \approx$ 400km, ensuring that the frictional boundary layer is adequately resolved.

The model is stepped forward with a time step $\delta t_\theta = 30$ hours for thermodynamic variables and $\delta t_v = 40$ minutes for momentum terms. With this time step, the stability parameter to the horizontal Laplacian friction [1]

$$S_l = 4\frac{A_h \delta t_v}{\Delta x^2} \quad (3.77)$$

evaluates to 0.16 at a latitude of $\phi = 80°$, which is below the 0.3 upper limit for stability. The zonal grid spacing Δx is smallest at $\phi = 80°$ where $\Delta x = r\cos(\phi)\Delta\phi \approx$ 77km.

The vertical dissipation coefficient, A_z, is set to $1 \times 10^{-3}\mathrm{m}^2\mathrm{s}^{-1}$. The associated stability limit

$$S_l = 4\frac{A_z \delta t_v}{\Delta z^2} \tag{3.78}$$

evaluates to 0.015 for the smallest model level spacing ($\Delta z_1 = 50\mathrm{m}$) which is again well below the upper stability limit.

The values of the horizontal (K_h) and vertical (K_z) diffusion coefficients for both temperature and salinity are set to 1×10^3 $\mathrm{m}^2\mathrm{s}^{-1}$ and 3×10^{-5} $\mathrm{m}^2\mathrm{s}^{-1}$ respectively. The stability limit related to K_h will be at $\phi = 80^\circ$ where $\Delta x \approx$ 77km. Here the stability parameter

$$S_l = \frac{4K_h \delta t_\theta}{\Delta x^2} \tag{3.79}$$

evaluates to 0.07, well below the stability limit of $S_l \approx 0.5$. The stability parameter related to K_z

$$S_l = \frac{4K_z \delta t_\theta}{\Delta z^2} \tag{3.80}$$

evaluates to 0.005 for $\min(\Delta z) = 50\mathrm{m}$, well below the stability limit of $S_l \approx 0.5$.

The numerical stability for inertial oscillations [1]

$$S_i = f^2 \delta {t_v}^2 \tag{3.81}$$

evaluates to 0.24 for $f = 2\omega \sin(80^\circ) = 1.43 \times 10^{-4}$ s^{-1}, which is close to the $S_i < 1$ upper limit for stability.

The advective CFL [1] for a extreme maximum horizontal flow speed of $|\vec{u}| = 2ms^{-1}$

$$S_a = \frac{|\vec{u}| \delta t_v}{\Delta x} \tag{3.82}$$

evaluates to 6×10^{-2}. This is well below the stability limit of 0.5.

The stability parameter for internal gravity waves propagating with a maximum speed of $c_g = 10$ ms^{-1} [1]

$$S_c = \frac{c_g \delta t_v}{\Delta x} \tag{3.83}$$

evaluates to 3×10^{-1}. This is close to the linear stability limit of 0.5.

3.12.3 Experiment Configuration

The model configuration for this experiment resides under the directory *verification/hs94.128x64x5*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/windx.bin*,
- *input/windy.bin*,
- *input/salt.bin*,
- *input/theta.bin*,
- *input/SSS.bin*,
- *input/SST.bin*,
- *input/topog.bin*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

3.12.3.1 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Lines 7-10 and 11-14

  ```
  tRef= 16.0 , 15.2 , 14.5 , 13.9 , 13.3 ,
  ```

  ...
  set reference values for potential temperature and salinity at each model level in units of °C and ppt. The entries are ordered from surface to depth. Density is calculated from anomalies at each level evaluated with respect to the reference values set here.
  S/R INI_THETA(*ini_theta.F*)

- Line 15,

```
viscAz=1.E-3,
```

this line sets the vertical Laplacian dissipation coefficient to $1\times10^{-3}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later. This variable is copied into model general vertical coordinate variable **viscAr**.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 16,

  ```
  viscAh=5.E5,
  ```

  this line sets the horizontal Laplacian frictional dissipation coefficient to $5 \times 10^5\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later.

- Lines 17,

  ```
  no_slip_sides=.FALSE.
  ```

  this line selects a free-slip lateral boundary condition for the horizontal Laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x.

- Lines 9,

  ```
  no_slip_bottom=.TRUE.
  ```

  this line selects a no-slip boundary condition for bottom boundary condition in the vertical Laplacian friction operator e.g. $u = v = 0$ at $z = -H$, where H is the local depth of the domain.

- Line 19,

  ```
  diffKhT=1.E3,
  ```

  this line sets the horizontal diffusion coefficient for temperature to $1000\,\mathrm{m}^2\mathrm{s}^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries.

- Line 20,

  ```
  diffKzT=3.E-5,
  ```

  this line sets the vertical diffusion coefficient for temperature to $3\times10^{-5}\,\mathrm{m}^2\mathrm{s}^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ at both the upper and lower boundaries.

- Line 21,

  ```
  diffKhS=1.E3,
  ```

this line sets the horizontal diffusion coefficient for salinity to $1000\,\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries.

- Line 22,

  ```
  diffKzS=3.E-5,
  ```

  this line sets the vertical diffusion coefficient for salinity to $3 \times 10^{-5}\,\mathrm{m^2 s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ at both the upper and lower boundaries.

- Lines 23-26

  ```
  beta=1.E-11,
  ...
  ```

  These settings do not apply for this experiment.

- Line 27,

  ```
  gravity=9.81,
  ```

  Sets the gravitational acceleration coefficient to $9.81\mathrm{ms}^{-1}$.

S/R CALC_PHI_HYD (*calc_phi_hyd.F*)
S/R INI_CG2D (*ini_cg2d.F*)
S/R INI_CG3D (*ini_cg3d.F*)
S/R INI_PARMS (*ini_parms.F*)
S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)

- Line 28-29,

  ```
  rigidLid=.FALSE.,
  implicitFreeSurface=.TRUE.,
  ```

  Selects the barotropic pressure equation to be the implicit free surface formulation.

- Line 30,

  ```
  eosType='POLY3',
  ```

  Selects the third order polynomial form of the equation of state.

S/R FIND_RHO (*find_rho.F*)
S/R FIND_ALPHA (*find_alpha.F*)

- Line 31,

  ```
  readBinaryPrec=32,
  ```

Sets format for reading binary input datasets holding model fields to use 32-bit representation for floating-point numbers.

S/R READ_WRITE_FLD (*read_write_fld.F*) *S/R READ_WRITE_REC* (*read_write_rec.F*)

- Line 36,

```
cg2dMaxIters=1000,
```

Sets maximum number of iterations the two-dimensional, conjugate gradient solver will use, **irrespective of convergence criteria being met**.

S/R CG2D (*cg2d.F*)

- Line 37,

```
cg2dTargetResidual=1.E-13,
```

Sets the tolerance which the two-dimensional, conjugate gradient solver will use to test for convergence in equation **??** to 1×10^{-13}. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

S/R CG2D (*cg2d.F*)

- Line 42,

```
startTime=0,
```

Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time steps in the **startTime** value. The internal time counter works in seconds.

- Line 43,

```
endTime=2808000.,
```

Sets the time (in seconds) at which this simulation will terminate. At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages.

- Line 44,

```
#endTime=62208000000,
```

A commented out setting for endTime for a 2000 year simulation.

- Line 45,

```
deltaTmom=2400.0,
```

Sets the timestep δt_v used in the momentum equations to 20 mins. See section **??**.

S/R TIMESTEP(*timestep.F*)

- Line 46,

```
tauCD=321428.,
```

Sets the D-grid to C-grid coupling time scale τ_{CD} used in the momentum equations. See section **??**.

S/R INI_PARMS(*ini_parms.F*) *S/R CALC_MOM_RHS*(*calc_mom_rhs.F*)

- Line 47,

```
deltaTtracer=108000.,
```

Sets the default timestep, δt_θ, for tracer equations to 30 hours. See section **??**.

S/R TIMESTEP_TRACER(*timestep_tracer.F*)

- Line 47,

```
bathyFile='topog.box'
```

This line specifies the name of the file from which the domain bathymetry is read. This file is a two-dimensional (x, y) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates a solid wall and a depth of $-2000m$ indicates open ocean. The matlab program *input/gendata.m* shows an example of how to generate a bathymetry file.

- Line 50,

```
zonalWindFile='windx.sin_y'
```

This line specifies the name of the file from which the x-direction surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program *input/gendata.m* includes example code to generate a valid **zonalWindFile** file.

other lines in the file *input/data* are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

```
 1 # ====================
 2 # | Model parameters |
 3 # ====================
 4 #
 5 # Continuous equation parameters
 6  &PARM01
 7  tRef= 16.0 , 15.2 , 14.5 , 13.9 , 13.3 ,
 8        12.4 , 11.3 ,  9.9 ,  8.4 ,  6.7 ,
 9         5.2 ,  3.8 ,  2.9 ,  2.3 ,  1.8 ,
10         1.5 ,  1.1 ,  0.8 ,  0.66,  0.63,
11  sRef= 34.65, 34.75, 34.82, 34.87, 34.90,
12        34.90, 34.86, 34.78, 34.69, 34.60,
13        34.58, 34.62, 34.68, 34.72, 34.73,
14        34.74, 34.73, 34.73, 34.72, 34.72,
15  viscAz=1.E-3,
16  viscAh=5.E5,
17  no_slip_sides=.FALSE.,
18  no_slip_bottom=.TRUE.,
19  diffKhT=1.E3,
20  diffKzT=3.E-5,
21  diffKhS=1.E3,
22  diffKzS=3.E-5,
23  beta=1.E-11,
24  f0=1.e-4,
25  tAlpha=2.E-4,
26  sBeta =7.4E-4,
27  gravity=9.81,
28  rigidLid=.FALSE.,
29  implicitFreeSurface=.TRUE.,
30  eosType='LINEAR',
31  readBinaryPrec=32,
32  &
33
34 # Elliptic solver parameters
35  &PARM02
36  cg2dMaxIters=1000,
37  cg2dTargetResidual=1.E-13,
38  &
39
40 # Time stepping parameters
41  &PARM03
42  startTime=0,
43  endTime=622080000,
44 #endTime=2808000.,
45  deltaTmom=2400.0,
46  tauCD=321428.,
47  deltaTtracer=108000.0,
```

```
 deltaTClock =108000.0,
 cAdjFreq=-1.,
 abEps=0.1,
 pChkptFreq=0.0,
 chkptFreq=0.0,
 dumpFreq=2592000.0,
 dumpFreq=31104000.,
 tauThetaClimRelax=2592000.0,
 tauSaltClimRelax=2592000.0,
 &

# Gridding parameters
 &PARM04
 usingCartesianGrid=.FALSE.,
 usingSphericalPolarGrid=.TRUE.,
 delZ= 5.000000e+01, 5.000000e+01, 5.500000e+01, 6.000000e+01,
       6.500000e+01, 7.000000e+01, 8.000000e+01, 9.500000e+01,
       1.200000e+02, 1.550000e+02, 2.000000e+02, 2.600000e+02,
       3.200000e+02, 4.000000e+02, 4.800000e+02, 5.700000e+02,
       6.550000e+02, 7.250000e+02, 7.750000e+02, 8.150000e+02,
 phiMin=-80.,
 delY=40*4,
 delX=90*4,
 &

# Input datasets
 &PARM05
 hydrogThetaFile='theta.bin',
 hydrogSaltFile='salt.bin',
 bathyFile='topog.bin',
 zonalWindFile='windx.bin',
 meridWindFile='windy.bin',
 thetaClimFile='SST.bin'
 saltClimFile='SSS.bin'
 &
```

3.12.3.2 File *input/data.pkg*

This file uses standard default values and does not contain customisations for this experiment.

3.12.3.3 File *input/eedata*

This file uses standard default values and does not contain customisations for this experiment.

3.12.3.4 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress ,τ_x, values. The units used are Nm^{-2}. Although τ_x is only a function of yn in this experiment this file must still define a complete two-dimensional map in order to be compatible with the standard code for loading forcing fields in MITgcm. The included matlab program *input/gendata.m* gives a complete code for creating the *input/windx.sin_y* file.

3.12.3.5 File *input/topog.box*

The *input/topog.box* file specifies a two-dimensional (x, y) map of depth values. For this experiment values are either $0m$ or -2000 m, corresponding respectively to a wall or to deep ocean. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays. The included matlab program *input/gendata.m* gives a complete code for creating the *input/topog.box* file.

3.12.3.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=4,
  ```

  this line sets the vertical domain extent in grid points.

```
1 C $Header: /u/gcmpack/manual/part3/case_studies/climatalogical_ogcm/code/SIZE.h.tex,v 1.1.1.1 2001/08
2 C $Name:  $
3 C
4 C     /==========================================================\
5 C     | SIZE.h Declare size of underlying computational grid.    |
6 C     |==========================================================|
7 C     | The design here support a three-dimensional model grid   |
```

```
 8 C     | with indices I,J and K. The three-dimensional domain     |
 9 C     | is comprised of nPx*nSx blocks of size sNx along one axis|
10 C     | nPy*nSy blocks of size sNy along another axis and one    |
11 C     | block of size Nz along the final axis.                   |
12 C     | Blocks have overlap regions of size OLx and OLy along the|
13 C     | dimensions that are subdivided.                          |
14 C     \==========================================================/
15 C     Voodoo numbers controlling data layout.
16 C     sNx - No. X points in sub-grid.
17 C     sNy - No. Y points in sub-grid.
18 C     OLx - Overlap extent in X.
19 C     OLy - Overlat extent in Y.
20 C     nSx - No. sub-grids in X.
21 C     nSy - No. sub-grids in Y.
22 C     nPx - No. of processes to use in X.
23 C     nPy - No. of processes to use in Y.
24 C     Nx  - No. points in X for the total domain.
25 C     Ny  - No. points in Y for the total domain.
26 C     Nr  - No. points in Z for full process domain.
27       INTEGER sNx
28       INTEGER sNy
29       INTEGER OLx
30       INTEGER OLy
31       INTEGER nSx
32       INTEGER nSy
33       INTEGER nPx
34       INTEGER nPy
35       INTEGER Nx
36       INTEGER Ny
37       INTEGER Nr
38       PARAMETER (
39      &           sNx =  90,
40      &           sNy =  40,
41      &           OLx =   3,
42      &           OLy =   3,
43      &           nSx =   1,
44      &           nSy =   1,
45      &           nPx =   1,
46      &           nPy =   1,
47      &           Nx  = sNx*nSx*nPx,
48      &           Ny  = sNy*nSy*nPy,
49      &           Nr  =  20)

50 C     MAX_OLX  - Set to the maximum overlap region size of any array
51 C     MAX_OLY    that will be exchanged. Controls the sizing of exch
52 C                routine buufers.
53       INTEGER MAX_OLX
54       INTEGER MAX_OLY
55       PARAMETER ( MAX_OLX = OLx,
56      &            MAX_OLY = OLy )
```

3.12.3.7 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.12.3.8 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.12.3.9 Other Files

Other files relevant to this experiment are

- *model/src/ini_cori.F*. This file initializes the model coriolis variables **fCorU**.
- *model/src/ini_spherical_polar_grid.F*
- *model/src/ini_parms.F*,
- *input/windx.sin_y*,

contain the code customisations and parameter settings for this experiments. Below we describe the customisations to these files associated with this experiment.

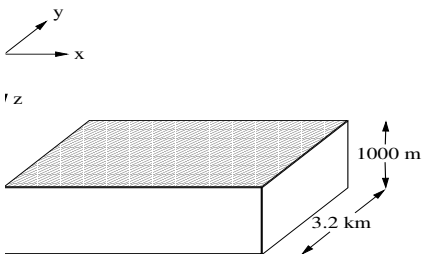


Figure 3.10: Schematic of simulation domain for the surface driven convection experiment. The domain is doubly periodic with an initially uniform temperature of 20 ^oC.

3.13 Surface Driven Convection

This experiment, figure 3.10, showcasing MITgcm's non-hydrostatic capability, was designed to explore the temporal and spatial characteristics of convection plumes as they might exist during a period of oceanic deep convection. It is

- non-hydrostatic
- doubly-periodic with cubic geometry
- has 50 m resolution
- Cartesian
- is on an f-plane
- with a linear equation of state

3.13.1 Overview

The model domain consists of an approximately 3 km square by 1 km deep box of initially unstratified, resting fluid. The domain is doubly periodic.

The experiment has 20 levels in the vertical, each of equal thickness $\Delta z =$ 50 m (the horizontal resolution is also 50 m). The fluid is initially unstratified with a uniform reference potential temperature $\theta = 20\ ^o$C. The equation of state used in this experiment is linear

$$\rho = \rho_0(1 - \alpha_\theta \theta^{'}) \tag{3.84}$$

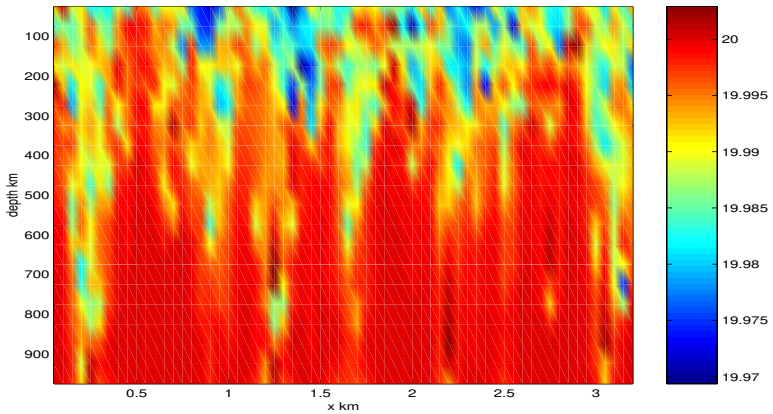


Figure 3.11:

which is implemented in the model as a density anomaly equation

$$\rho^{'} = -\rho_0 \alpha_\theta \theta^{'} \tag{3.85}$$

with $\rho_0 = 1000\,\mathrm{kg\,m^{-3}}$ and $\alpha_\theta = 2 \times 10^{-4}\,\mathrm{degrees}^{-1}$. Integrated forward in this configuration the model state variable **theta** is equivalent to either in-situ temperature, T, or potential temperature, θ. For consistency with other examples, in which the equation of state is non-linear, we use θ to represent temperature here. This is the quantity that is carried in the model core equations.

As the fluid in the surface layer is cooled (at a mean rate of 800 Wm^2), it becomes convectively unstable and overturns, at first close to the grid-scale, but, as the flow matures, on larger scales (figures 3.11 and 3.12), under the influence of rotation ($f_o = 10^{-4}\ \mathrm{s}^{-1}$) .

Model parameters are specified in file *input/data*. The grid dimensions are prescribed in *code/SIZE.h*. The forcing (file *input/Qsurf.bin*) is specified in a binary data file generated using the Matlab script *input/gendata.m*.

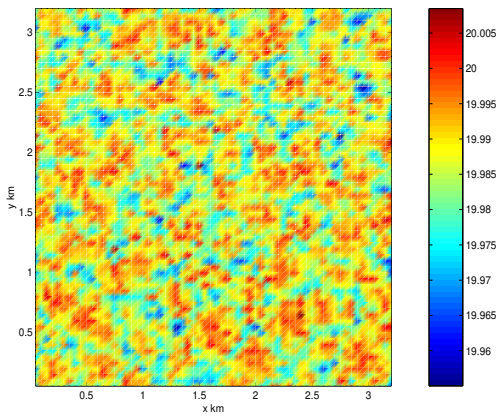

3
2.5
2
1.5
1
0.5
y km
0.5
1
1.5
2
2.5
3
x km
20.005
20
19.995
19.99
19.985
19.98
19.975
19.97
19.965
19.96

Figure 3.12:

3.13.2 Equations solved

The model is configured in nonhydrostatic form, that is, all terms in the Navier Stokes equations are retained and the pressure field is found, subject to appropriate bounday condintions, through inversion of a three-dimensional elliptic equation.

The implicit free surface form of the pressure equation described in Marshall et. al [39] is employed. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation. The thermodynamic forcing appears as a sink in the potential temperature, θ, equation (**??**). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p^{'}}{\partial x} - \nabla_h \cdot A_h \nabla_h u - \frac{\partial}{\partial z} A_z \frac{\partial u}{\partial z} = \begin{cases} 0 & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.86)$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p^{'}}{\partial y} - \nabla_h \cdot A_h \nabla_h v - \frac{\partial}{\partial z} A_z \frac{\partial v}{\partial z} = \begin{cases} 0 & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.87)$$

$$\frac{Dw}{Dt} + g\frac{\rho^{'}}{\rho} + \frac{1}{\rho}\frac{\partial p^{'}}{\partial z} - \nabla_h \cdot A_h \nabla_h w - \frac{\partial}{\partial z} A_z \frac{\partial w}{\partial z} = \begin{cases} 0 & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.88)$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} + = 0 \quad (3.89)$$

$$\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - \frac{\partial}{\partial z} K_z \frac{\partial \theta}{\partial z} = \begin{cases} \mathcal{F}_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \quad (3.90)$$

where $u = \frac{Dx}{Dt}$, $v = \frac{Dy}{Dt}$ and $w = \frac{Dz}{Dt}$ are the components of the flow vector in directions x, y and z. The pressure is diagnosed through inversion (subject to appropriate boundary conditions) of a 3-D elliptic equation derived from the divergence of the momentum equations and continuity (see section 1.3.6).

3.13.3 Discrete numerical configuration

The domain is discretised with a uniform grid spacing in each direction. There are 64 grid cells in directions x and y and 20 vertical levels thus the domain comprises a total of just over 80 000 gridpoints.

3.13.4 Numerical stability criteria and other considerations

For a heat flux of 800 Wm^2 and a rotation rate of 10^{-4} s^{-1} the plume-scale can be expected to be a few hundred meters guiding our choice of grid resolution. This in turn restricts the timestep we can take. It is also desirable to minimise the level of diffusion and viscosity we apply.

For this class of problem it is generally the advective time-scale which restricts the timestep.

For an extreme maximum flow speed of $|\vec{u}| = 1ms^{-1}$, at a resolution of 50 m, the implied maximum timestep for stability, δt_u is

(3.91)

The choice of $\delta t = 10$ s is a safe 20 percent of this maximum.

Interpreted in terms of a mixing-length hypothesis, a magnitude of Laplacian diffusion coefficient $\kappa_h (= \kappa_v) = 0.1$ m^{2}s^{-1} is consistent with an eddy velocity of 2 mm s^{-1} correlated over 50 m.

3.13.5 Experiment configuration

The model configuration for this experiment resides under the directory *verification/convection/*. The experiment files

- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.
- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/Qsurf.bin*,

contain the code customisations and parameter settings for this experiment. Below we describe these experiment-specific customisations.

3.13.5.1 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.13.5.2 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customisations for this experiment.

3.13.5.3 File *code/SIZE.h*

Three lines are customized in this file. These prescribe the domain grid dimensions.

- Line 36,

  ```
  sNx=64,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 37,

  ```
  sNy=64,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 46,

  ```
  Nr=20,
  ```

  this line sets the vertical domain extent in grid points.

```
 1 C     /==========================================================\
 2 C     | SIZE.h Declare size of underlying computational grid.    |
 3 C     |==========================================================|
 4 C     | The design here support a three-dimensional model grid   |
 5 C     | with indices I,J and K. The three-dimensional domain     |
 6 C     | is comprised of nPx*nSx blocks of size sNx along one axis|
 7 C     | nPy*nSy blocks of size sNy along another axis and one    |
 8 C     | block of size Nz along the final axis.                   |
 9 C     | Blocks have overlap regions of size OLx and OLy along the|
10 C     | dimensions that are subdivided.                          |
11  C     \==========================================================/
12  C     Voodoo numbers controlling data layout.
13  C     sNx - No. X points in sub-grid.
14  C     sNy - No. Y points in sub-grid.
15  C     OLx - Overlap extent in X.
16  C     OLy - Overlat extent in Y.
17  C     nSx - No. sub-grids in X.
18  C     nSy - No. sub-grids in Y.
19  C     nPx - No. of processes to use in X.
20  C     nPy - No. of processes to use in Y.
21  C     Nx  - No. points in X for the total domain.
22  C     Ny  - No. points in Y for the total domain.
23  C     Nr  - No. points in Z for full process domain.
24        INTEGER sNx
25        INTEGER sNy
```

```
26        INTEGER OLx
27        INTEGER OLy
28        INTEGER nSx
29        INTEGER nSy
30        INTEGER nPx
31       INTEGER nPy
32       INTEGER Nx
33       INTEGER Ny
34       INTEGER Nr
35       PARAMETER (
36      &           sNx =  64,
37      &           sNy =  64,
38      &           OLx =   3,
39      &           OLy =   3,
40      &           nSx =   1,
41      &           nSy =   1,
42      &           nPx =   1,
43      &           nPy =   1,
44      &           Nx  = sNx*nSx*nPx,
45      &           Ny  = sNy*nSy*nPy,
46      &           Nr  =  20)

47 C     MAX_OLX  - Set to the maximum overlap region size of any array
48 C     MAX_OLY    that will be exchanged. Controls the sizing of exch
49 C                routine buufers.
50       INTEGER MAX_OLX
51       INTEGER MAX_OLY
52       PARAMETER ( MAX_OLX = OLx,
53      &            MAX_OLY = OLy )
```

3.13.5.4 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Line 4,

  ```
  4   tRef=20*20.0,
  ```

  this line sets the initial and reference values of potential temperature at each model level in units of °C. Here the value is arbitrary since, in this case, the flow evolves independently of the absolute magnitude of the reference temperature. For each depth level the initial and reference profiles will be uniform in x and y. The values specified are read into the variable **tRef** in the model code, by procedure *S/R INI_PARMS (ini_parms.F)* . The temperature field is initialised, by procedure *S/R INI_THETA (ini_theta.F)* .

- Line 5,

```
5   sRef=20*35.0,
```

this line sets the initial and reference values of salinity at each model level in units of ppt. In this case salinity is set to an (arbitrary) uniform value of 35.0 ppt. However since, in this example, density is independent of salinity, an appropriatly defined initial salinity could provide a useful passive tracer. For each depth level the initial and reference profiles will be uniform in x and y. The values specified are read into the variable **sRef** in the model code, by procedure *S/R INI_PARMS (ini_parms.F)* . The salinity field is initialised, by procedure *S/R INI_SALT (ini_salt.F)* .

- Line 6,

```
6   viscAh=0.1,
```

this line sets the horizontal laplacian dissipation coefficient to 0.1 m^2s^{-1}. Boundary conditions for this operator are specified later. The variable **viscAh** is read in the routine *S/R INI_PARMS (ini_params.F)* and applied in routines *S/R CALC_MOM_RHS (calc_mom_rhs.F)* and *S/R CALC_GW (calc_gw.F)* .

- Line 7,

```
7   viscAz=0.1,
```

this line sets the vertical laplacian frictional dissipation coefficient to 0.1 m^2s^{-1}. Boundary conditions for this operator are specified later. The variable **viscAz** is read in the routine *S/R INI_PARMS (ini_parms.F)* and is copied into model general vertical coordinate variable **viscAr** . At each time step, the viscous term contribution to the momentum equations is calculated in routine *S/R CALC_DIFFUSIVITY (calc_diffusivity.F)* .

- Line 8,

```
no_slip_sides=.FALSE.
```

this line selects a free-slip lateral boundary condition for the horizontal laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x. The variable **no_slip_sides** is read in the routine *S/R INI_PARMS (ini_parms.F)* and the boundary condition is evaluated in routine *S/R CALC_MOM_RHS (calc_mom_rhs.F)* .

- Lines 9,

```
no_slip_bottom=.TRUE.
```

this line selects a no-slip boundary condition for the bottom boundary condition in the vertical laplacian friction operator e.g. $u = v = 0$ at $z = -H$, where H is the local depth of the domain. The variable **no_slip_bottom** is read in the routine *S/R INI_PARMS (ini_parms.F)* and is applied in the routine *S/R CALC_MOM_RHS (calc_mom_rhs.F)* .

- Line 11,

```
diffKhT=0.1,
```

this line sets the horizontal diffusion coefficient for temperature to 0.1 m^2s^{-1}. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ at all boundaries. The variable **diffKhT** is read in the routine *S/R INI_PARMS (ini_parms.F)* and used in routine *S/R CALC_GT (calc_gt.F)* .

- Line 12,

```
diffKzT=0.1,
```

this line sets the vertical diffusion coefficient for temperature to 0.1 m^2s^{-1}. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries. The variable **diffKzT** is read in the routine *S/R INI_PARMS (ini_parms.F)* . It is copied into model general vertical coordinate variable **diffKrT** which is used in routine *S/R CALC_DIFFUSIVITY (calc_diffusivity.F)* .

- Line 13,

```
diffKhS=0.1,
```

this line sets the horizontal diffusion coefficient for salinity to 0.1 m^2s^{-1}. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries. The variable **diffKsT** is read in the routine *S/R INI_PARMS (ini_parms.F)* and used in routine *S/R CALC_GS (calc_gs.F)* .

- Line 14,

```
diffKzS=0.1,
```

this line sets the vertical diffusion coefficient for temperature to 0.1 m^2s^{-1}. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries. The variable **diffKzS** is read in the routine *S/R INI_PARMS (ini_parms.F)* . It is copied into model general vertical coordinate variable **diffKrS** which is used in routine *S/R CALC_DIFFUSIVITY (calc_diffusivity.F)* .

- Line 15,

```
f0=1E-4,
```

this line sets the Coriolis parameter to 1×10^{-4} s^{-1}. Since $\beta = 0.0$ this value is then adopted throughout the domain.

- Line 16,

```
beta=0.E-11,
```

this line sets the the variation of Coriolis parameter with latitude to 0.

- Line 17,

```
tAlpha=2.E-4,
```

This line sets the thermal expansion coefficient for the fluid to 2×10^{-4} ^{o} C^{-1}. The variable **tAlpha** is read in the routine *S/R INI_PARMS (ini_parms.F)* . The routine *S/R FIND_RHO (find_rho.F)* makes use of **tAlpha**.

- Line 18,

```
sBeta=0,
```

This line sets the saline expansion coefficient for the fluid to 0 consistent with salt's passive role in this example.

- Line 23-24,

```
rigidLid=.FALSE.,
implicitFreeSurface=.TRUE.,
```

Selects the barotropic pressure equation to be the implicit free surface formulation.

- Line 25,

```
eosType='LINEAR',
```

Selects the linear form of the equation of state.

- Line 26,

```
nonHydrostatic=.TRUE.,
```

Selects for non-hydrostatic code.

- Line 27,

`readBinaryPrec=64,`

Sets format for reading binary input datasets holding model fields to use 64-bit representation for floating-point numbers.

- Line 31,

  `cg2dMaxIters=1000,`

  Inactive - the pressure field in a non-hydrostatic simulation is inverted through a 3D elliptic equation.

- Line 32,

  `cg2dTargetResidual=1.E-9,`

  Inactive - the pressure field in a non-hydrostatic simulation is inverted through a 3D elliptic equation.

- Line 33,

  `cg3dMaxIters=40,`

  This line sets the maximum number of iterations the three-dimensional, conjugate gradient solver will use to 40, **irrespective of the convergence criteria being met**. Used in routine *S/R CG3D (cg3d.F)* .

- Line 34,

  `cg3dTargetResidual=1.E-9,`

  Sets the tolerance which the three-dimensional, conjugate gradient solver will use to test for convergence in equation **??** to 1×10^{-9}. The solver will iterate until the tolerance falls below this value or until the maximum number of solver iterations is reached. Used in routine *S/R CG3D (cg3d.F)* .

- Line 38,

  `startTime=0,`

  Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time steps in the **startTime** value. The internal time counter works in seconds.

- Line 39,

```
nTimeSteps=8640.,
```

Sets the number of timesteps at which this simulation will terminate (in this case 8640 timesteps or 1 day or $\delta t = 10$ s). At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages.

- Line 40,

```
deltaT=10,
```

Sets the timestep δt to 10 s.

- Line 51,

```
dXspacing=50.0,
```

Sets horizontal (x-direction) grid interval to 50 m.

- Line 52,

```
dYspacing=50.0,
```

Sets horizontal (y-direction) grid interval to 50 m.

- Line 53,

```
delZ=20*50.0,
```

Sets vertical grid spacing to 50 m. Must be consistent with *code/SIZE.h*. Here, 20 corresponds to the number of vertical levels.

- Line 57,

```
surfQfile='Qsurf.bin'
```

This line specifies the name of the file from which the surface heat flux is read. This file is a two-dimensional (x, y) map. It is assumed to contain 64-bit binary numbers giving the value of Q (W m^2) to be applied in each surface grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The matlab program *input/gendata.m* shows how to generate the surface heat flux file used in the example. The variable **Qsurf** is read in the routine *S/R INI_PARMS (ini_parms.F)* and applied in *S/R EXTERNAL_FORCING_SURF (external_forcing_surf.F)* where the flux is converted to a temperature tendency.

```
 1 # Model parameters
 2 # Continuous equation parameters
 3  &PARM01
 4  tRef=20*20.0,
 5  sRef=20*35.0,
 6  viscAh=0.1,
 7  viscAz=0.1,
 8  no_slip_sides=.FALSE.,
 9  no_slip_bottom=.FALSE.,
10   viscA4=0.E12,
11   diffKhT=0.1,
12   diffKzT=0.1,
13   diffKhS=0.1,
14   diffKzS=0.1,
15   f0=1E-4,
16   beta=0.E-11,
17   tAlpha=2.0E-4,
18   sBeta =0.,
19   gravity=9.81,
20   rhoConst=1000.0,
21   rhoNil=1000.0,
22   heatCapacity_Cp=3900.0,
23   rigidLid=.FALSE.,
24   implicitFreeSurface=.TRUE.,
25   eosType='LINEAR',
26   nonHydrostatic=.TRUE.,
27   readBinaryPrec=64,
28   &
29  # Elliptic solver parameters
30   &PARM02
31   cg2dMaxIters=1000,
32   cg2dTargetResidual=1.E-9,
33   cg3dMaxIters=40,
34   cg3dTargetResidual=1.E-9,
35   &
36  # Time stepping parameters
37   &PARM03
38   nIter0=0,
39   nTimeSteps=1440,
40   deltaT=60,
41   abEps=0.1,
42   pChkptFreq=0.0,
43   chkptFreq=0.0,
44   dumpFreq=600,
45   monitorFreq=1.E-5,
46   &
47  # Gridding parameters
48   &PARM04
49   usingCartesianGrid=.TRUE.,
50   usingSphericalPolarGrid=.FALSE.,
```

Figure 3.13:

```
51   dXspacing=50.0,
52    dYspacing=50.0,
53    delZ=20*50.0,
54    &
55   # Input datasets
56    &PARM05
57    surfQfile='Qsurf.bin',
58    &
```

3.13.5.5 File *input/data.pkg*

This file uses standard default values and does not contain customisations for this experiment.

3.13.5.6 File *input/eedata*

This file uses standard default values and does not contain customisations for this experiment.

3.13.5.7 File *input/Qsurf.bin*

The file *input/Qsurf.bin* specifies a two-dimensional (x, y) map of heat flux values where $Q = Q_o \times (0.5 + \text{random number between 0 and 1})$.

In the example $Q_o = 800$ W m^{-2} so that values of Q lie in the range 400 to 1200 W m^{-2} with a mean of Q_o. Although the flux models a loss, because it is directed upwards, according to the model's sign convention, Q is positive.

3.13.6 Running the example

3.13.6.1 Code download

In order to run the examples you must first download the code distribution. Instructions for downloading the code can be found in 3.2.

3.13.6.2 Experiment Location

This example experiments is located under the release sub-directory

verification/convection/

3.13.6.3 Running the Experiment

To run the experiment

1. Set the current directory to *input/*

   ```
   % cd input
   ```

2. Verify that current directory is now correct

   ```
   % pwd
   ```

   You should see a response on the screen ending in *verification/convection/input*

3. Run the genmake script to create the experiment *Makefile*

   ```
   % ../../../tools/genmake -mods=../code
   ```

4. Create a list of header file dependencies in *Makefile*

   ```
   % make depend
   ```

5. Build the executable file.

   ```
   % make
   ```

6. Run the *mitgcmuv* executable

   ```
   % ./mitgcmuv
   ```

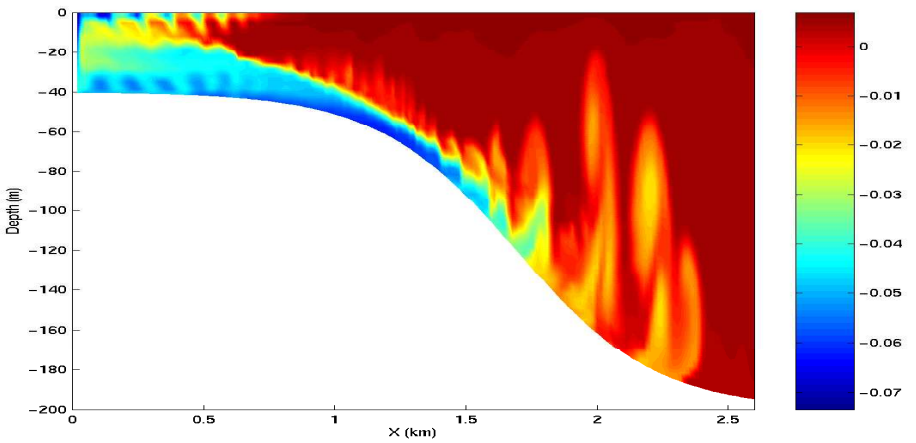


Figure 3.14: Temperature after 23 hours of cooling. The cold dense water is mixed with ambient water as it accelerates down the slope and hence is warmed than the unmixed plume.

3.14 Gravity Plume On a Continental Slope

An important test of any ocean model is the ability to represent the flow of dense fluid down a slope. One example of such a flow is a non-rotating gravity plume on a continental slope, forced by a limited area of surface cooling above a continental shelf. Because the flow is non-rotating, a two dimensional model can be used in the across slope direction. The experiment is non-hydrostatic and uses open-boundaries to radiate transients at the deep water end. (Dense flow down a slope can also be forced by a dense inflow prescribed on the continental shelf; this configuration is being implemented by the DOME (Dynamics of Overflow Mixing and Entrainment) collaboration to compare solutions in different models).

The fluid is initially unstratified. The surface buoyancy loss B_0 (dimensions of $\mathrm{L}^2\mathrm{T}^{-3}$) over a cross-shelf distance R causes vertical convective mixing and modifies the density of the fluid by an amount

$$\Delta\rho = \frac{B_0\rho_0 t}{gH} \tag{3.92}$$

where H is the depth of the shelf, g is the acceleration due to gravity, t is time since onset of cooling and ρ_0 is the reference density. Dense fluid slumps under gravity, with a flow speed close to the gravity wave speed:

$$U \sim \sqrt{g'H} \sim \sqrt{\frac{g\Delta\rho H}{\rho_0}} \sim \sqrt{B_0 t} \tag{3.93}$$

A steady state is rapidly established in which the buoyancy flux out of the

cooling region is balanced by the surface buoyancy loss. Then

$$U \sim (B_0 R)^{1/3} \; ; \; \Delta\rho \sim \frac{\rho_0}{gH}(B_0 R)^{2/3} \tag{3.94}$$

The Froude number of the flow on the shelf is close to unity (but in practice slightly less than unity, giving subcritical flow). When the flow reaches the slope, it accelerates, so that it may become supercritical (provided the slope angle α is steep enough). In this case, a hydraulic control is established at the shelf break. On the slope, where the Froude number is greater than one, and gradient Richardson number (defined as $Ri \sim g' h^*/U^2$ where h^* is the thickness of the interface between dense and ambient fluid) is reduced below 1/4, Kelvin-Helmholtz instability is possible, and leads to entrainment of ambient fluid into the plume, modifying the density, and hence the acceleration down the slope. Kelvin-Helmholtz instability is suppressed at low Reynolds and Peclet numbers given by

$$Re \sim \frac{Uh}{\nu} \sim \frac{(B_0 R)^{1/3} h}{\nu} \; ; \; Pe = RePr \tag{3.95}$$

where h is the depth of the dense fluid on the slope. Hence this experiment is carried out in the high Re, Pe regime. A further constraint is that the convective heat flux must be much greater than the diffusive heat flux (Nusselt number $>> 1$). Then

$$Nu = \frac{Uh^*}{\kappa} >> 1 \tag{3.96}$$

Finally, since we have assumed that the convective mixing on the shelf occurs in a much shorter time than the horizontal equilibration, this implies $H/R << 1$.

Hence to summarize the important nondimensional parameters, and the limits we are considering:

$$\frac{H}{R} << 1 \; ; \; Re >> 1 \; ; \; Pe >> 1 \; ; \; Nu >> 1 \; ; \; ; \; Ri < 1/4 \tag{3.97}$$

In addition we are assuming that the slope is steep enough to provide sufficient acceleration to the gravity plume, but nonetheless much less that 1 : 1, since many Kelvin-Helmholtz billows appear on the slope, implying horizontal lengthscale of the slope >> the depth of the dense fluid.

3.14.1 Configuration

The topography, spatial grid, forcing and initial conditions are all specified in binary data files generated using a *Matlab* script called `gendata.m` and detailed in section 3.14.2. Other model parameters are specified in file `data` and `data.obcs` and detailed in section 3.14.4.

3.14.2 Binary input data

The domain is 200 m deep and 6.4 km across. Uniform resolution of $60 \times 3^1/_3$ m is used in the vertical and variable resolution of the form shown in Fig. 3.15 with

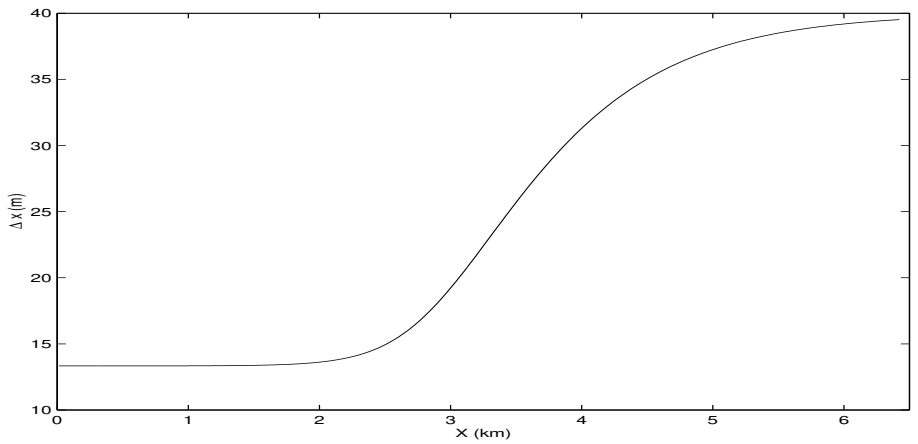


Figure 3.15: Horizontal grid spacing, Δx, in the across-slope direction for the gravity plume experiment.

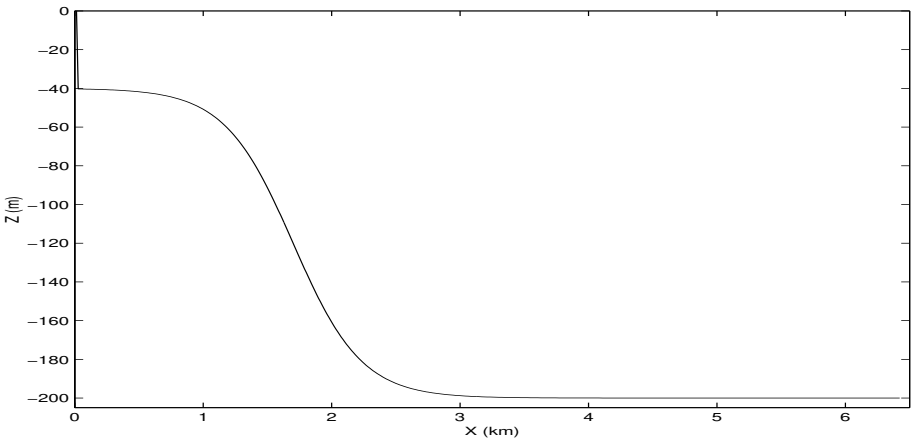


Figure 3.16: Topography, $h(x)$, used for the gravity plume experiment.

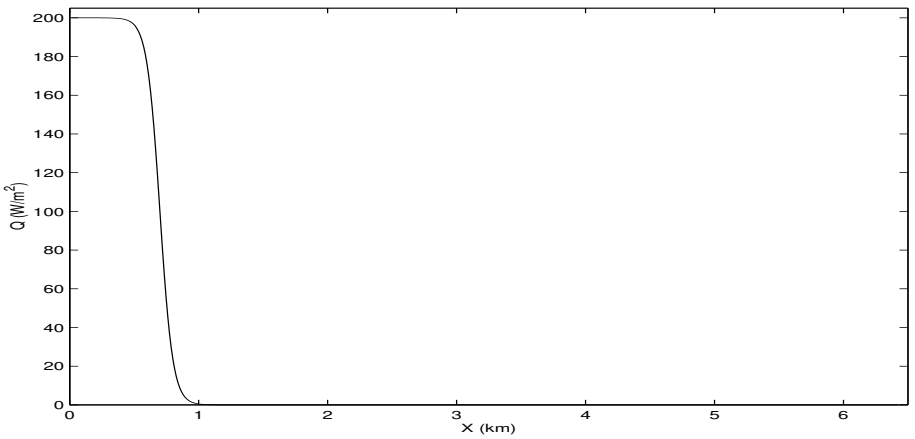


Figure 3.17: Upward surface heat flux, $Q(x)$, used as forcing in the gravity plume experiment.

320 points is usedin the horizontal. The formula for Δx is:

$$\Delta x(i) = \Delta x_1 + (\Delta x_2 - \Delta x_1)(1 + \tanh\left(\frac{i - i_s}{w}\right))/2$$

where

$$\begin{aligned} Nx &= 320 \\ Lx &= 6400 \ \text{(m)} \\ \Delta x_1 &= \frac{2}{3}\frac{Lx}{Nx} \ \text{(m)} \\ \Delta x_2 &= \frac{Lx/2}{Nx - Lx/2\Delta x_1} \ \text{(m)} \\ i_s &= Lx/(2\Delta x_1) \\ w &= 40 \end{aligned}$$

Here, Δx_1 is the resolution on the shelf, Δx_2 is the resolution in deep water and Nx is the number of points in the horizontal.

The topography, shown in Fig. 3.16, is given by:

$$H(x) = -H_o + (H_o - h_s)(1 + \tanh\left(\frac{x - x_s}{L_s}\right))/2$$

where

$$\begin{aligned} H_o &= 200 \ \text{(m)} \\ h_s &= 40 \ \text{(m)} \\ x_s &= 1500 + Lx/2 \ \text{(m)} \\ L_s &= \frac{(H_o - h_s)}{2s} \ \text{(m)} \\ s &= 0.15 \end{aligned}$$

Here, s is the maximum slope, H_o is the maximum depth, h_s is the shelf depth, x_s is the lateral position of the shelf-break and L_s is the length-scale of the slope.

The forcing is through heat loss over the shelf, shown in Fig. 3.17 and takes the form of a fixed flux with profile:

$$Q(x) = Q_o(1 + \tanh\left(\frac{x - x_q}{L_q}\right))/2$$

where

$$\begin{aligned} Q_o &= 200 \ \text{(W m}^{-2}\text{)} \\ x_q &= 2500 + Lx/2 \ \text{(m)} \\ L_q &= 100 \ \text{(m)} \end{aligned}$$

Here, Q_o, is the maximum heat flux, x_q is the position of the cut-off and L_q is the width of the cut-off.

The initial tempeture field is unstratified but with random perturbations, to induce convection early on in the run. The random perturbation are calculated in computational space and because of the variable resolution introduce some spatial correlations but this does not matter for this experiment. The perturbations have range $0 - 0.01$ °K.

3.14.3 Code configuration

The computational domain (number of points) is specified in `code/SIZE.h` and is configured as a single tile of dimensions $320 \times 1 \times 60$. There are no experiment specific source files.

Optional code required to for this experiment are the non-hydrostatic algorithm and open-boundaries:

- Non-hydrostatic terms and algorithm are enabled with **#define ALLOW_NONHYDROSTATIC** in `code/CPP_OPTIONS.h` and activated with **nonHydrostatic=.TRUE.,** in namelist *PARM01* of `input/data`.
- Open boundaries are enabled with **#define ALLOW_OBCS** in `code/CPP_OPTIONS.h` and activated with **use_OBCS=.TRUE,** in namelist *PACKAGES* of `input/data.pkg`.

g	9.81 m s^{-2}	acceleration due to gravity
ρ_o	999.8 kg m^{-3}	reference density
α	2×10^{-4} K^{-1}	expansion coefficient
A_h	1×10^{-2} $m^2 s^{-1}$	horizontal viscosity
A_v	1×10^{-3} $m^2 s^{-1}$	vertical viscosity
κ_h	0 $m^2 s^{-1}$	(explicit) horizontal diffusion
κ_v	0 $m^2 s^{-1}$	(explicit) vertical diffusion
Δt	20 s	time step
Δz	$3.3\dot{3}$ m	vertical grid spacing
Δx	$13.\dot{3} - 39.5$ m	horizontal grid spacing

Table 3.2: Model parameters used in the gravity plume experiment.

3.14.4 Model parameters

The model parameters (Table 3.2) are specified in `input/data` and if not assume the default values defined in `model/src/set_defaults.F`. A linear equation of state is used, **eosType='LINEAR'**, but only temperature is active, **sBeta=0.E-4**. For the given heat flux, Q_o, the buoyancy forcing is $B_o = \frac{g\alpha Q}{\rho_o c_p} \sim 10^{-7}$ $m^2 s^{-3}$. Using $R = 10^3$ m, the shelf width, then this gives a velocity scale of $U \sim 5 \times 10^{-2}$ m s^-1 for the initial front but will accelerate by an order of magnitude over the slope. The temperature anomaly will be of order $\Delta\theta \sim 3 \times 10^{-2}$ K. The viscosity is constant and gives a Reynolds number of 100, using $h = 20$ m for the initial front and will be an order magnitude bigger over the slope. There is no explicit diffusion but a non-linear advection scheme is used for temperature which adds enough diffusion so as to keep the model stable. The time-step is set to 20 s and gives Courant number order one when the flow reaches the bottom of the slope.

3.14.5 Build and run the model

Build the model per usual. For example:

```
% cd verification/plume_on_slope
% mkdir build
% cd build
% ../../../tools/genmake -mods=../code -disable=gmredi,kpp,zonal_filt
  ,shap_filt
% make depend
% make
```

When compilation is complete, run the model as usual, for example:

```
% cd ../
% mkdir run
```

```
% cp input/* build/mitgcmuv run/
% cd run
% ./mitgcmuv > output.txt
```

3.15 Centennial Time Scale Tracer Injection

3.15.1 Introduction

This document describes the fourth example MITgcm experiment. This example illustrates the use of the MITgcm to perform sensitivity analysis in a large scale ocean circulation simulation.

3.15.2 Overview

This example experiment demonstrates using the MITgcm to simulate the planetary ocean circulation. The simulation is configured with realistic geography and bathymetry on a $4^\circ \times 4^\circ$ spherical polar grid. Twenty vertical layers are used in the vertical, ranging in thickness from 50 m at the surface to 815 m at depth, giving a maximum model depth of 6 km. At this resolution, the configuration can be integrated forward for thousands of years on a single processor desktop computer.

The model is forced with climatological wind stress data and surface flux data from Da Silva [10]. Climatological data from Levitus [36] is used to initialize the model hydrography. Levitus data is also used throughout the calculation to derive air-sea fluxes of heat at the ocean surface. These fluxes are combined with climatological estimates of surface heat flux and fresh water, resulting in a mixed boundary condition of the style described in Haney [25]. Altogether, this yields the following forcing applied in the model surface layer.

$$\mathcal{F}_u = \frac{\tau_x}{\rho_0 \Delta z_s} \tag{3.98}$$

$$\mathcal{F}_v = \frac{\tau_y}{\rho_0 \Delta z_s} \tag{3.99}$$

$$\mathcal{F}_\theta = -\lambda_\theta(\theta - \theta^*) - \frac{1}{C_p \rho_0 \Delta z_s}\mathcal{Q} \tag{3.100}$$

$$\mathcal{F}_s = -\lambda_s(S - S^*) + \frac{S_0}{\Delta z_s}(\mathcal{E} - \mathcal{P} - \mathcal{R}) \tag{3.101}$$

where $\mathcal{F}_u$, $\mathcal{F}_v$, $\mathcal{F}_\theta$, $\mathcal{F}_s$ are the forcing terms in the zonal and meridional momentum and in the potential temperature and salinity equations respectively. The term Δz_s represents the top ocean layer thickness. It is used in conjunction with the reference density, ρ_0 (here set to $999.8\,\mathrm{kg\,m^{-3}}$), the reference salinity, S_0 (here set to 35ppt), and a specific heat capacity C_p to convert wind-stress fluxes given in $\mathrm{N}\,m^{-2}$,

The configuration is illustrated in figure **??**.

3.15.3 Discrete Numerical Configuration

The model is configured in hydrostatic form. The domain is discretised with a uniform grid spacing in latitude and longitude of $\Delta x = \Delta y = 4^{\circ}$, so that there are ninety grid cells in the x and forty in the y direction (Arctic polar regions are not included in this experiment). Vertically the model is configured with twenty layers with the following thicknesses $\Delta z_1 = 50\,\mathrm{m}$, $\Delta z_2 = 50\,\mathrm{m}$, $\Delta z_3 = 55\,\mathrm{m}$, $\Delta z_4 = 60\,\mathrm{m}$, $\Delta z_5 = 65\,\mathrm{m}$, $\Delta z_6 = 70\,\mathrm{m}$, $\Delta z_7 = 80\,\mathrm{m}$, $\Delta z_8 = 95\,\mathrm{m}$, $\Delta z_9 = 120\,\mathrm{m}$, $\Delta z_{10} = 155\,\mathrm{m}$, $\Delta z_{11} = 200\,\mathrm{m}$, $\Delta z_{12} = 260\,\mathrm{m}$, $\Delta z_{13} = 320\,\mathrm{m}$, $\Delta z_{14} = 400\,\mathrm{m}$, $\Delta z_{15} = 480\,\mathrm{m}$, $\Delta z_{16} = 570\,\mathrm{m}$, $\Delta z_{17} = 655\,\mathrm{m}$, $\Delta z_{18} = 725\,\mathrm{m}$, $\Delta z_{19} = 775\,\mathrm{m}$, $\Delta z_{20} = 815\,\mathrm{m}$ (here the numeric subscript indicates the model level index number, `k`). The implicit free surface form of the pressure equation described in Marshall et. al [39] is employed. A Laplacian operator, ∇^2, provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress momentum inputs are added to the momentum equations for both the zonal flow, u and the meridional flow v, according to equations (3.98) and (3.99). Thermodynamic forcing inputs are added to the equations for potential temperature, θ, and salinity, S, according to equations (3.100) and (3.101). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p^{'}}{\partial x} - A_h\nabla_h^2 u - A_z\frac{\partial^2 u}{\partial z^2} = \mathcal{F}_u \tag{3.102}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p^{'}}{\partial y} - A_h\nabla_h^2 v - A_z\frac{\partial^2 v}{\partial z^2} = \mathcal{F}_v \tag{3.103}$$

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{3.104}$$

$$\frac{D\theta}{Dt} - K_h\nabla_h^2\theta - \Gamma(K_z)\frac{\partial^2\theta}{\partial z^2} = \mathcal{F}_\theta \tag{3.105}$$

$$\frac{Ds}{Dt} - K_h\nabla_h^2 s - \Gamma(K_z)\frac{\partial^2 s}{\partial z^2} = \mathcal{F}_s \tag{3.106}$$

$$g\rho_0\eta + \int_{-z}^{0}\rho^{'}dz = p^{'} \tag{3.107}$$

$$\tag{3.108}$$

where u and v are the x and y components of the flow vector $\vec{u}$. The suffices s, i indicate surface and interior model levels respectively. As described in MITgcm Numerical Solution Procedure 2, the time evolution of potential temperature, θ, equation is solved prognostically. The total pressure, p, is diagnosed by summing pressure due to surface elevation η and the hydrostatic pressure.

3.15.3.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, A_h, is set to $400ms^{-1}$. This value is chosen to yield a Munk layer width [1],

$$M_w = \pi(\frac{A_h}{\beta})^{\frac{1}{3}} \tag{3.109}$$

of $\approx$ 100km. This is greater than the model resolution in mid-latitudes Δx, ensuring that the frictional boundary layer is well resolved.

The model is stepped forward with a time step $\delta t = 1200$secs. With this time step the stability parameter to the horizontal Laplacian friction [1]

$$S_l = 4\frac{A_h \delta t}{\Delta x^2} \tag{3.110}$$

evaluates to 0.012, which is well below the 0.3 upper limit for stability.

The vertical dissipation coefficient, A_z, is set to $1 \times 10^{-2}\mathrm{m}^2\mathrm{s}^{-1}$. The associated stability limit

$$S_l = 4\frac{A_z \delta t}{\Delta z^2} \tag{3.111}$$

evaluates to 4.8×10^{-5} which is again well below the upper limit. The values of A_h and A_z are also used for the horizontal (K_h) and vertical (K_z) diffusion coefficients for temperature respectively.

The numerical stability for inertial oscillations [1]

$$S_i = f^2 \delta t^2 \tag{3.112}$$

evaluates to 0.0144, which is well below the 0.5 upper limit for stability.

The advective CFL [1] for a extreme maximum horizontal flow speed of $|\vec{u}| = 2ms^{-1}$

$$S_a = \frac{|\vec{u}|\delta t}{\Delta x} \tag{3.113}$$

evaluates to 5×10^{-2}. This is well below the stability limit of 0.5.

The stability parameter for internal gravity waves [1]

$$S_c = \frac{c_g \delta t}{\Delta x} \tag{3.114}$$

evaluates to 5×10^{-2}. This is well below the linear stability limit of 0.25.

3.15.4 Code Configuration

The model configuration for this experiment resides under the directory *verification/exp1/*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/windx.sin_y*,
- *input/topog.box*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for this experiments. Below we describe the customizations to these files associated with this experiment.

3.15.4.1 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Line 4,

  ```
  tRef=20.,10.,8.,6.,
  ```

  this line sets the initial and reference values of potential temperature at each model level in units of °C. The entries are ordered from surface to depth. For each depth level the initial and reference profiles will be uniform in x and y.

S/R INI_THETA(*ini_theta.F*)

- Line 6,

  ```
  viscAz=1.E-2,
  ```

  this line sets the vertical Laplacian dissipation coefficient to $1\times10^{-2}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later. This variable is copied into model general vertical coordinate variable **viscAr**.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 7,

```
viscAh=4.E2,
```

this line sets the horizontal Laplacian frictional dissipation coefficient to $1 \times 10^{-2} \mathrm{m}^2 \mathrm{s}^{-1}$. Boundary conditions for this operator are specified later.

- Lines 8,

```
no_slip_sides=.FALSE.
```

this line selects a free-slip lateral boundary condition for the horizontal Laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x.

- Lines 9,

```
no_slip_bottom=.TRUE.
```

this line selects a no-slip boundary condition for bottom boundary condition in the vertical Laplacian friction operator e.g. $u = v = 0$ at $z = -H$, where H is the local depth of the domain.

- Line 10,

```
diffKhT=4.E2,
```

this line sets the horizontal diffusion coefficient for temperature to $400\,\mathrm{m}^2\mathrm{s}^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ at all boundaries.

- Line 11,

```
diffKzT=1.E-2,
```

this line sets the vertical diffusion coefficient for temperature to $10^{-2}\,\mathrm{m}^2\mathrm{s}^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries.

- Line 13,

```
tAlpha=2.E-4,
```

This line sets the thermal expansion coefficient for the fluid to $2{\times}10^{-4}\,\mathrm{degrees}^{-1}$

S/R FIND_RHO(*find_rho.F*)

- Line 18,

```
eosType='LINEAR'
```

This line selects the linear form of the equation of state.

S/R FIND_RHO(*find_rho.F*)

- Line 40,

  ```
  usingSphericalPolarGrid=.TRUE.,
  ```

  This line requests that the simulation be performed in a spherical polar coordinate system. It affects the interpretation of grid input parameters, for example **delX** and **delY** and causes the grid generation routines to initialize an internal grid based on spherical polar geometry.

  S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)

- Line 41,

  ```
  phiMin=0.,
  ```

  This line sets the southern boundary of the modeled domain to 0° latitude. This value affects both the generation of the locally orthogonal grid that the model uses internally and affects the initialization of the coriolis force. Note - it is not required to set a longitude boundary, since the absolute longitude does not alter the kernel equation discretisation.

- Line 42,

  ```
  delX=60*1.,
  ```

  This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to 1° in longitude.

- Line 43,

  ```
  delY=60*1.,
  ```

  This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to 1° in latitude.

- Line 44,

  ```
  delZ=500.,500.,500.,500.,
  ```

  This line sets the vertical grid spacing between each z-coordinate line in the discrete grid to 500 m, so that the total model depth is 2 km. The variable **delZ** is copied into the internal model coordinate variable **delR**

  S/R INI_VERTICAL_GRID(*ini_vertical_grid.F*)

- Line 47,

  ```
  bathyFile='topog.box'
  ```

This line specifies the name of the file from which the domain bathymetry is read. This file is a two-dimensional (x, y) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates a solid wall and a depth of $-2000m$ indicates open ocean. The matlab program *input/gendata.m* shows an example of how to generate a bathymetry file.

- Line 50,

  ```
  zonalWindFile='windx.sin_y'
  ```

  This line specifies the name of the file from which the x-direction surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program *input/gendata.m* includes example code to generate a valid **zonalWindFile** file.

other lines in the file *input/data* are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

3.15.4.2 File *input/data.pkg*

This file uses standard default values and does not contain customizations for this experiment.

3.15.4.3 File *input/eedata*

This file uses standard default values and does not contain customizations for this experiment.

3.15.4.4 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress ,τ_x, values. The units used are Nm^{-2}. Although τ_x is only a function of yn in this experiment this file must still define a complete two-dimensional map in order to be compatible with the standard code for loading forcing fields in MITgcm. The included matlab program *input/gendata.m* gives a complete code for creating the *input/windx.sin_y* file.

3.15.4.5 File *input/topog.box*

The *input/topog.box* file specifies a two-dimensional (x, y) map of depth values. For this experiment values are either $0m$ or -2000 m, corresponding respectively to a wall or to deep ocean. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal

arrays. The included matlab program *input/gendata.m* gives a complete code for creating the *input/topog.box* file.

3.15.4.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=4,
  ```

  this line sets the vertical domain extent in grid points.

3.15.4.7 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

3.15.4.8 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

3.15.4.9 Other Files

Other files relevant to this experiment are

- *model/src/ini_cori.F*. This file initializes the model coriolis variables **fCorU**.
- *model/src/ini_spherical_polar_grid.F*
- *model/src/ini_parms.F*,
- *input/windx.sin_y*,

contain the code customizations and parameter settings for this experiments. Below we describe the customizations to these files associated with this experiment.

3.16 Doing it yourself: customizing the code

When you are ready to run the model in the configuration you want, the easiest thing is to use and adapt the setup of the case studies experiment (described previously) that is the closest to your configuration. Then, the amount of setup will be minimized. In this section, we focus on the setup relative to the "numerical model" part of the code (the setup relative to the "execution environment" part is covered in the parallel implementation section) and on the variables and parameters that you are likely to change.

3.16.1 Building/compiling the code elsewhere

In the example above (section 3.5) we built the executable in the *input* directory of the experiment for convenience. You can also configure and compile the code in other locations, for example on a scratch disk with out having to copy the entire source tree. The only requirement to do so is you have `genmake2` in your path or you know the absolute path to `genmake2`.

The following sections outline some possible methods of organizing your source and data.

3.16.1.1 Building from the *../code directory*

This is just as simple as building in the *input/* directory:

```
% cd verification/exp2/code
% ../../../tools/genmake2
% make depend
% make
```

However, to run the model the executable (*mitgcmuv*) and input files must be in the same place. If you only have one calculation to make:

```
% cd ../input
% cp ../code/mitgcmuv ./
% ./mitgcmuv > output.txt
```

or if you will be making multiple runs with the same executable:

```
% cd ../
% cp -r input run1
% cp code/mitgcmuv run1
% cd run1
% ./mitgcmuv > output.txt
```

3.16.1.2 Building from a new directory

Since the *input* directory contains input files it is often more useful to keep *input* pristine and build in a new directory within *verification/exp2/*:

```
% cd verification/exp2
% mkdir build
% cd build
% ../../../tools/genmake2 -mods=../code
% make depend
% make
```

This builds the code exactly as before but this time you need to copy either the executable or the input files or both in order to run the model. For example,

```
% cp ../input/* ./
% ./mitgcmuv > output.txt
```

or if you tend to make multiple runs with the same executable then running in a new directory each time might be more appropriate:

```
% cd ../
% mkdir run1
% cp build/mitgcmuv run1/
% cp input/* run1/
% cd run1
% ./mitgcmuv > output.txt
```

3.16.1.3 Building on a scratch disk

Model object files and output data can use up large amounts of disk space so it is often the case that you will be operating on a large scratch disk. Assuming the model source is in */MITgcm* then the following commands will build the model in */scratch/exp2-run1*:

```
% cd /scratch/exp2-run1
% ~/MITgcm/tools/genmake2 -rootdir=~/MITgcm \
  -mods=~/MITgcm/verification/exp2/code
% make depend
% make
```

To run the model here, you'll need the input files:

```
% cp ~/MITgcm/verification/exp2/input/* ./
% ./mitgcmuv > output.txt
```

As before, you could build in one directory and make multiple runs of the one experiment:

```
% cd /scratch/exp2
% mkdir build
% cd build
% ~/MITgcm/tools/genmake2 -rootdir=~/MITgcm \
  -mods=~/MITgcm/verification/exp2/code
```

```
% make depend
% make
% cd ../
% cp -r ~/MITgcm/verification/exp2/input run2
% cd run2
% ./mitgcmuv > output.txt
```

3.16.2 Using genmake2

To compile the code, first use the program `genmake2` (located in the `tools` directory) to generate a Makefile. `genmake2` is a shell script written to work with all "sh"–compatible shells including bash v1, bash v2, and Bourne. Internally, `genmake2` determines the locations of needed files, the compiler, compiler options, libraries, and Unix tools. It relies upon a number of "optfiles" located in the `tools/build_options` directory.

The purpose of the optfiles is to provide all the compilation options for particular "platforms" (where "platform" roughly means the combination of the hardware and the compiler) and code configurations. Given the combinations of possible compilers and library dependencies (*eg.* MPI and NetCDF) there may be numerous optfiles available for a single machine. The naming scheme for the majority of the optfiles shipped with the code is

OS_HARDWARE_COMPILER

where

OS is the name of the operating system (generally the lower-case output of the `'uname'` command)

HARDWARE is a string that describes the CPU type and corresponds to output from the `'uname -m'` command:

ia32 is for "x86" machines such as i386, i486, i586, i686, and athlon

ia64 is for Intel IA64 systems (eg. Itanium, Itanium2)

amd64 is AMD x86_64 systems

ppc is for Mac PowerPC systems

COMPILER is the compiler name (generally, the name of the FORTRAN executable)

In many cases, the default optfiles are sufficient and will result in usable Makefiles. However, for some machines or code configurations, new "optfiles" must be written. To create a new optfile, it is generally best to start with one of the defaults and modify it to suit your needs. Like `genmake2`, the optfiles are all written using a simple "sh"–compatible syntax. While nearly all variables used within `genmake2` may be specified in the optfiles, the critical ones that should be defined are:

FC the FORTRAN compiler (executable) to use

DEFINES the command-line DEFINE options passed to the compiler

CPP the C pre-processor to use

NOOPTFLAGS options flags for special files that should not be optimized

For example, the optfile for a typical Red Hat Linux machine ("ia32" architecture) using the GCC (g77) compiler is

```
FC=g77
DEFINES='-D_BYTESWAPIO -DWORDLENGTH=4'
CPP='cpp  -traditional -P'
NOOPTFLAGS='-O0'
#  For IEEE, use the "-ffloat-store" option
if test "x$IEEE" = x ; then
    FFLAGS='-Wimplicit -Wunused -Wuninitialized'
    FOPTIM='-O3 -malign-double -funroll-loops'
else
    FFLAGS='-Wimplicit -Wunused -ffloat-store'
    FOPTIM='-O0 -malign-double'
fi
```

If you write an optfile for an unrepresented machine or compiler, you are strongly encouraged to submit the optfile to the MITgcm project for inclusion. Please send the file to the

MITgcm-support@mitgcm.org

mailing list.

In addition to the optfiles, `genmake2` supports a number of helpful command-line options. A complete list of these options can be obtained from:

```
% genmake2 -h
```

The most important command-line options are:

`--optfile=/PATH/FILENAME` specifies the optfile that should be used for a particular build.

If no "optfile" is specified (either through the command line or the MITGCM_OPTFILE environment variable), genmake2 will try to make a reasonable guess from the list provided in *tools/build_options*. The method used for making this guess is to first determine the combination of operating system and hardware (eg. "linux_ia32") and then find a working FORTRAN compiler within the user's path. When these three items have been identified, genmake2 will try to find an optfile that has a matching name.

`--pdefault='PKG1 PKG2 PKG3 ...'` specifies the default set of packages to be used. The normal order of precedence for packages is as follows:

1. If available, the command line (`--pdefault`) settings over-rule any others.
2. Next, `genmake2` will look for a file named "`packages.conf`" in the local directory or in any of the directories specified with the `--mods` option.
3. Finally, if neither of the above are available, `genmake2` will use the `/pkg/pkg_default` file.

`--pdepend=/PATH/FILENAME` specifies the dependency file used for packages.

If not specified, the default dependency file *pkg/pkg_depend* is used. The syntax for this file is parsed on a line-by-line basis where each line containes either a comment ("#") or a simple "PKGNAME1 (+—-)PKGNAME2" pairwise rule where the "+" or "-" symbol specifies a "must be used with" or a "must not be used with" relationship, respectively. If no rule is specified, then it is assumed that the two packages are compatible and will function either with or without each other.

`--adof=/path/to/file` specifies the "adjoint" or automatic differentiation options file to be used. The file is analogous to the "optfile" defined above but it specifies information for the AD build process.

The default file is located in *tools/adjoint_options/adjoint_default* and it defines the "TAF" and "TAMC" compilers. An alternate version is also available at *tools/adjoint_options/adjoint_staf* that selects the newer "STAF" compiler. As with any compilers, it is helpful to have their directories listed in your `$PATH` environment variable.

`--mods='DIR1 DIR2 DIR3 ...'` specifies a list of directories containing "modifications". These directories contain files with names that may (or may not) exist in the main MITgcm source tree but will be overridden by any identically-named sources within the "MODS" directories.

The order of precedence for this "name-hiding" is as follows:

- "MODS" directories (in the order given)
- Packages either explicitly specified or provided by default (in the order given)
- Packages included due to package dependencies (in the order that that package dependencies are parsed)
- The "standard dirs" (which may have been specified by the "-standarddirs" option)

`--mpi` This option enables certain MPI features (using CPP `#define`s) within the code and is necessary for MPI builds (see Section 3.16.3).

`--make=/path/to/gmake` Due to the poor handling of soft-links and other bugs common with the `make` versions provided by commercial Unix vendors, GNU `make` (sometimes called `gmake`) should be preferred. This option provides a means for specifying the make executable to be used.

`--bash=/path/to/sh` On some (usually older UNIX) machines, the "bash" shell is unavailable. To run on these systems, `genmake2` can be invoked using an "sh" (that is, a Bourne, POSIX, or compatible) shell. The syntax in these circumstances is:

```
% /bin/sh genmake2 -bash=/bin/sh [...options...]
```

where `/bin/sh` can be replaced with the full path and name of the desired shell.

3.16.3 Building with MPI

Building MITgcm to use MPI libraries can be complicated due to the variety of different MPI implementations available, their dependencies or interactions with different compilers, and their often ad-hoc locations within file systems. For these reasons, its generally a good idea to start by finding and reading the documentation for your machine(s) and, if necessary, seeking help from your local systems administrator.

The steps for building MITgcm with MPI support are:

1. Determine the locations of your MPI-enabled compiler and/or MPI libraries and put them into an options file as described in Section 3.16.2. One can start with one of the examples in:

   ```
   MITgcm/tools/build_options/
   ```

   such as `linux_ia32_g77+mpi_cg01` or `linux_ia64_efc+mpi` and then edit it to suit the machine at hand. You may need help from your user guide or local systems administrator to determine the exact location of the MPI libraries. If libraries are not installed, MPI implementations and related tools are available including:

   - MPICH
   - LAM/MPI
   - MPIexec

2. Build the code with the `genmake2 -mpi` option (see Section 3.16.2) using commands such as:

   ```
   %  ../../../tools/genmake2 -mods=../code -mpi -of=YOUR_OPTFILE
   %  make depend
   %  make
   ```

3. Run the code with the appropriate MPI "run" or "exec" program provided with your particular implementation of MPI. Typical MPI packages such as MPICH will use something like:

   ```
   %  mpirun -np 4 -machinefile mf ./mitgcmuv
   ```

   Sightly more complicated scripts may be needed for many machines since execution of the code may be controlled by both the MPI library and a job scheduling and queueing system such as PBS, LoadLeveller, Condor, or any of a number of similar tools. A few example scripts (those used for our regular verification runs) are available at: `http://mitgcm.org/cgi-bin/viewcvs.cgi/MITgcm_contrib/test_`

An example of the above process on the MITgcm cluster ("cg01") using the GNU g77 compiler and the mpich MPI library is:

```
%  cd MITgcm/verification/exp5
%  mkdir build
%  cd build
%  ../../../tools/genmake2 -mpi -mods=../code \
     -of=../../../tools/build_options/linux_ia32_g77+mpi_cg01
%  make depend
%  make
%  cd ../input
%  /usr/local/pkg/mpi/mpi-1.2.4..8a-gm-1.5/g77/bin/mpirun.ch_gm \
     -machinefile mf --gm-kill 5 -v -np 2  ../build/mitgcmuv
```

3.16.4 Configuration and setup

The CPP keys relative to the "numerical model" part of the code are all defined and set in the file *CPP_OPTIONS.h* in the directory *model/inc* or in one of the *code* directories of the case study experiments under *verification.* The model parameters are defined and declared in the file *model/inc/PARAMS.h* and their default values are set in the routine *model/src/set_defaults.F.* The default values can be modified in the namelist file *data* which needs to be located in the directory where you will run the model. The parameters are initialized in the routine *model/src/ini_parms.F.* Look at this routine to see in what part of the namelist the parameters are located.

In what follows the parameters are grouped into categories related to the computational domain, the equations solved in the model, and the simulation controls.

3.16.5 Computational domain, geometry and time-discretization

dimensions

The number of points in the x, y, and r directions are represented by the variables **sNx**, **sNy** and **Nr** respectively which are declared and set in the file *model/inc/SIZE.h*. (Again, this assumes a mono-processor calculation. For multiprocessor calculations see the section on parallel implementation.)

grid

Three different grids are available: cartesian, spherical polar, and curvilinear (which includes the cubed sphere). The grid is set through the logical variables **usingCartesianGrid**, **usingSphericalPolarGrid**, and **usingCurvilinearGrid**. In the case of spherical and curvilinear grids, the southern boundary is defined through the variable **phiMin** which corresponds to the latitude of the southern most cell face (in degrees). The resolution along the x and y directions is controlled by the 1D arrays **delx** and **dely** (in meters in the case of a cartesian grid, in degrees otherwise). The vertical grid spacing is set through the 1D array **delz** for the ocean (in meters) or **delp** for the atmosphere (in Pa). The variable **Ro_SeaLevel** represents the standard position of Sea-Level in "R" coordinate. This is typically set to 0m for the ocean (default value) and 10^5Pa for the atmosphere. For the atmosphere, also set the logical variable **groundAtK1** to `'.TRUE.'` which puts the first level (k=1) at the lower boundary (ground).

For the cartesian grid case, the Coriolis parameter f is set through the variables **f0** and **beta** which correspond to the reference Coriolis parameter (in s^{-1}) and $\frac{\partial f}{\partial y}$(in $m^{-1}s^{-1}$) respectively. If **beta** is set to a nonzero value, **f0** is the value of f at the southern edge of the domain.

topography - full and partial cells

The domain bathymetry is read from a file that contains a 2D (x,y) map of depths (in m) for the ocean or pressures (in Pa) for the atmosphere. The file name is represented by the variable **bathyFile**. The file is assumed to contain binary numbers giving the depth (pressure) of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The model code applies without modification to enclosed, periodic, and double periodic domains. Periodicity is assumed by default and is suppressed by setting the depths to 0m for the cells at the limits of the computational domain (note: not sure this is the case for the atmosphere). The precision with which to read the binary data is controlled by the integer variable **readBinaryPrec** which can take the value `32` (single precision) or `64` (double precision). See the matlab program *gendata.m* in the *input* directories under *verification* to see how the bathymetry files are generated for the case study experiments.

To use the partial cell capability, the variable **hFacMin** needs to be set to a value between 0 and 1 (it is set to 1 by default) corresponding to the minimum fractional size of the cell. For example if the bottom cell is 500m thick and **hFacMin** is set to 0.1, the actual thickness of the cell (i.e. used in the code) can cover a range of discrete values 50m apart from 50m to 500m depending on the value of the bottom depth (in **bathyFile**) at this point.

Note that the bottom depths (or pressures) need not coincide with the models levels as deduced from **delz** or **delp**. The model will interpolate

the numbers in **bathyFile** so that they match the levels obtained from **delz** or **delp** and **hFacMin**.

(Note: the atmospheric case is a bit more complicated than what is written here I think. To come soon...)

time-discretization

The time steps are set through the real variables **deltaTMom** and **deltaTtracer** (in s) which represent the time step for the momentum and tracer equations, respectively. For synchronous integrations, simply set the two variables to the same value (or you can prescribe one time step only through the variable **deltaT**). The Adams-Bashforth stabilizing parameter is set through the variable **abEps** (dimensionless). The stagger baroclinic time stepping can be activated by setting the logical variable **staggerTimeStep** to `'.TRUE.'`.

3.16.6 Equation of state

First, because the model equations are written in terms of perturbations, a reference thermodynamic state needs to be specified. This is done through the 1D arrays **tRef** and **sRef**. **tRef** specifies the reference potential temperature profile (in ^{o}C for the ocean and ^{o}K for the atmosphere) starting from the level k=1. Similarly, **sRef** specifies the reference salinity profile (in ppt) for the ocean or the reference specific humidity profile (in g/kg) for the atmosphere.

The form of the equation of state is controlled by the character variables **buoyancyRelation** and **eosType**. **buoyancyRelation** is set to `'OCEANIC'` by default and needs to be set to `'ATMOSPHERIC'` for atmosphere simulations. In this case, **eosType** must be set to `'IDEALGAS'`. For the ocean, two forms of the equation of state are available: linear (set **eosType** to `'LINEAR'`) and a polynomial approximation to the full nonlinear equation (set **eosType** to `'POLYNOMIAL'`). In the linear case, you need to specify the thermal and haline expansion coefficients represented by the variables **tAlpha** (in K^{-1}) and **sBeta** (in ppt^{-1}). For the nonlinear case, you need to generate a file of polynomial coefficients called *POLY3.COEFFS*. To do this, use the program *utils/knudsen2/knudsen2.f* under the model tree (a Makefile is available in the same directory and you will need to edit the number and the values of the vertical levels in *knudsen2.f* so that they match those of your configuration).

There there are also higher polynomials for the equation of state:

`'UNESCO'`: The UNESCO equation of state formula of Fofonoff and Millard [15]. This equation of state assumes in-situ temperature, which is not a model variable; *its use is therefore discouraged, and it is only listed for completeness.*

`'JMD95Z'`: A modified UNESCO formula by Jackett and McDougall [33], which uses the model variable potential temperature as input. The `'Z'` indicates that this equation of state uses a horizontally and temporally constant pressure $p_0 = -g\rho_0 z$.

'JMD95P': A modified UNESCO formula by Jackett and McDougall [33], which uses the model variable potential temperature as input. The **'P'** indicates that this equation of state uses the actual hydrostatic pressure of the last time step. Lagging the pressure in this way requires an additional pickup file for restarts.

'MDJWF': The new, more accurate and less expensive equation of state by McDougall et al. [41]. It also requires lagging the pressure and therefore an additional pickup file for restarts.

For none of these options an reference profile of temperature or salinity is required.

3.16.7 Momentum equations

In this section, we only focus for now on the parameters that you are likely to change, i.e. the ones relative to forcing and dissipation for example. The details relevant to the vector-invariant form of the equations and the various advection schemes are not covered for the moment. We assume that you use the standard form of the momentum equations (i.e. the flux-form) with the default advection scheme. Also, there are a few logical variables that allow you to turn on/off various terms in the momentum equation. These variables are called **momViscosity, momAdvection, momForcing, useCoriolis, momPressureForcing, momStepping** and **metricTerms** and are assumed to be set to **'.TRUE.'** here. Look at the file *model/inc/PARAMS.h* for a precise definition of these variables.

initialization

The velocity components are initialized to 0 unless the simulation is starting from a pickup file (see section on simulation control parameters).

forcing

This section only applies to the ocean. You need to generate wind-stress data into two files **zonalWindFile** and **meridWindFile** corresponding to the zonal and meridional components of the wind stress, respectively (if you want the stress to be along the direction of only one of the model horizontal axes, you only need to generate one file). The format of the files is similar to the bathymetry file. The zonal (meridional) stress data are assumed to be in Pa and located at U-points (V-points). As for the bathymetry, the precision with which to read the binary data is controlled by the variable **readBinaryPrec**. See the matlab program *gendata.m* in the *input* directories under *verification* to see how simple analytical wind forcing data are generated for the case study experiments.

There is also the possibility of prescribing time-dependent periodic forcing. To do this, concatenate the successive time records into a single file

(for each stress component) ordered in a (x,y,t) fashion and set the following variables: **periodicExternalForcing** to `'.TRUE.'`, **externForcingPeriod** to the period (in s) of which the forcing varies (typically 1 month), and **externForcingCycle** to the repeat time (in s) of the forcing (typically 1 year – note: **externForcingCycle** must be a multiple of **externForcingPeriod**). With these variables set up, the model will interpolate the forcing linearly at each iteration.

dissipation

The lateral eddy viscosity coefficient is specified through the variable **viscAh** (in m^2s^{-1}). The vertical eddy viscosity coefficient is specified through the variable **viscAz** (in m^2s^{-1}) for the ocean and **viscAp** (in Pa^2s^{-1}) for the atmosphere. The vertical diffusive fluxes can be computed implicitly by setting the logical variable **implicitViscosity** to `'.TRUE.'`. In addition, biharmonic mixing can be added as well through the variable **viscA4** (in m^4s^{-1}). On a spherical polar grid, you might also need to set the variable **cosPower** which is set to 0 by default and which represents the power of cosine of latitude to multiply viscosity. Slip or no-slip conditions at lateral and bottom boundaries are specified through the logical variables **no_slip_sides** and **no_slip_bottom**. If set to `'.FALSE.'`, free-slip boundary conditions are applied. If no-slip boundary conditions are applied at the bottom, a bottom drag can be applied as well. Two forms are available: linear (set the variable **bottomDragLinear** in s^{-1}) and quadratic (set the variable **bottomDragQuadratic** in m^{-1}).

The Fourier and Shapiro filters are described elsewhere.

C-D scheme

If you run at a sufficiently coarse resolution, you will need the C-D scheme for the computation of the Coriolis terms. The variable **tauCD**, which represents the C-D scheme coupling timescale (in s) needs to be set.

calculation of pressure/geopotential

First, to run a non-hydrostatic ocean simulation, set the logical variable **nonHydrostatic** to `'.TRUE.'`. The pressure field is then inverted through a 3D elliptic equation. (Note: this capability is not available for the atmosphere yet.) By default, a hydrostatic simulation is assumed and a 2D elliptic equation is used to invert the pressure field. The parameters controlling the behaviour of the elliptic solvers are the variables **cg2dMaxIters** and **cg2dTargetResidual** for the 2D case and **cg3dMaxIters** and **cg3dTargetResidual** for the 3D case. You probably won't need to alter the default values (are we sure of this?).

For the calculation of the surface pressure (for the ocean) or surface geopotential (for the atmosphere) you need to set the logical variables **rigidLid** and **implicitFreeSurface** (set one to `'.TRUE.'` and the other to `'.FALSE.'` depending on how you want to deal with the ocean upper or atmosphere lower boundary).

3.16.8 Tracer equations

This section covers the tracer equations i.e. the potential temperature equation and the salinity (for the ocean) or specific humidity (for the atmosphere) equation. As for the momentum equations, we only describe for now the parameters that you are likely to change. The logical variables **tempDiffusion tempAdvection tempForcing**, and **tempStepping** allow you to turn on/off terms in the temperature equation (same thing for salinity or specific humidity with variables **saltDiffusion**, **saltAdvection** etc.). These variables are all assumed here to be set to `'.TRUE.'`. Look at file *model/inc/PARAMS.h* for a precise definition.

initialization

The initial tracer data can be contained in the binary files **hydrogThetaFile** and **hydrogSaltFile**. These files should contain 3D data ordered in an (x,y,r) fashion with k=1 as the first vertical level. If no file names are provided, the tracers are then initialized with the values of **tRef** and **sRef** mentioned above (in the equation of state section). In this case, the initial tracer data are uniform in x and y for each depth level.

forcing

This part is more relevant for the ocean, the procedure for the atmosphere not being completely stabilized at the moment.

A combination of fluxes data and relaxation terms can be used for driving the tracer equations. For potential temperature, heat flux data (in W/m^2) can be stored in the 2D binary file **surfQfile**. Alternatively or in addition, the forcing can be specified through a relaxation term. The SST data to which the model surface temperatures are restored to are supposed to be stored in the 2D binary file **thetaClimFile**. The corresponding relaxation time scale coefficient is set through the variable **tauThetaClimRelax** (in s). The same procedure applies for salinity with the variable names **EmPmRfile**, **saltClimFile**, and **tauSaltClimRelax** for freshwater flux (in m/s) and surface salinity (in ppt) data files and relaxation time scale coefficient (in s), respectively. Also for salinity, if the CPP key **USE_NATURAL_BCS** is turned on, natural boundary conditions are applied i.e. when computing the surface salinity tendency, the freshwater flux is multiplied by the model surface salinity instead of a constant salinity value.

As for the other input files, the precision with which to read the data is controlled by the variable **readBinaryPrec**. Time-dependent, periodic forcing can be applied as well following the same procedure used for the wind forcing data (see above).

dissipation

Lateral eddy diffusivities for temperature and salinity/specific humidity are specified through the variables **diffKhT** and **diffKhS** (in m^2/s). Ver-

tical eddy diffusivities are specified through the variables **diffKzT** and **diffKzS** (in m^2/s) for the ocean and **diffKpT** and **diffKpS** (in Pa^2/s) for the atmosphere. The vertical diffusive fluxes can be computed implicitly by setting the logical variable **implicitDiffusion** to `'.TRUE.'`. In addition, biharmonic diffusivities can be specified as well through the coefficients **diffK4T** and **diffK4S** (in m^4/s). Note that the cosine power scaling (specified through **cosPower**—see the momentum equations section) is applied to the tracer diffusivities (Laplacian and biharmonic) as well. The Gent and McWilliams parameterization for oceanic tracers is described in the package section. Finally, note that tracers can be also subject to Fourier and Shapiro filtering (see the corresponding section on these filters).

ocean convection

Two options are available to parameterize ocean convection: one is to use the convective adjustment scheme. In this case, you need to set the variable **cadjFreq**, which represents the frequency (in s) with which the adjustment algorithm is called, to a non-zero value (if set to a negative value by the user, the model will set it to the tracer time step). The other option is to parameterize convection with implicit vertical diffusion. To do this, set the logical variable **implicitDiffusion** to `'.TRUE.'` and the real variable **ivdc_kappa** to a value (in m^2/s) you wish the tracer vertical diffusivities to have when mixing tracers vertically due to static instabilities. Note that **cadjFreq** and **ivdc_kappa**can not both have non-zero value.

3.16.9 Simulation controls

The model "clock" is defined by the variable **deltaTClock** (in s) which determines the IO frequencies and is used in tagging output. Typically, you will set it to the tracer time step for accelerated runs (otherwise it is simply set to the default time step **deltaT**). Frequency of checkpointing and dumping of the model state are referenced to this clock (see below).

run duration

The beginning of a simulation is set by specifying a start time (in s) through the real variable **startTime** or by specifying an initial iteration number through the integer variable **nIter0**. If these variables are set to nonzero values, the model will look for a "pickup" file *pickup.0000nIter0* to restart the integration. The end of a simulation is set through the real variable **endTime** (in s). Alternatively, you can specify instead the number of time steps to execute through the integer variable **nTimeSteps**.

frequency of output

Real variables defining frequencies (in s) with which output files are written on disk need to be set up. **dumpFreq** controls the frequency with

which the instantaneous state of the model is saved. **chkPtFreq** and **pchkPtFreq** control the output frequency of rolling and permanent checkpoint files, respectively. See section 1.5.1 Output files for the definition of model state and checkpoint files. In addition, time-averaged fields can be written out by setting the variable **taveFreq** (in s). The precision with which to write the binary data is controlled by the integer variable w**riteBinaryPrec** (set it to `32` or `64`).

3.17 Testing

A script (`testreport`) for automated testing is included in the model within the *verification* directory. While intended mostly for advanced users, the script can be helpful for beginners.

3.17.1 Using `testreport`

On many systems, the program can be run with the command:

```
% cd verification
% ./testreport -ieee
```

which will do the following:

1. Locate all "valid" test directories. Here, valid tests are defined to be those directories within the current directory (which is generally *verification*) that contain a subdirectory and file with the names *results/output.txt*.
2. Then within each valid test:
   (a) run `genmake2` to produce a *Makefile*
   (b) build an executable
   (c) run the executable
   (d) compare and the output of the executable with the contents of certain variables within *TESTNAME/results/output.txt*
   (e) print and, if requested (with the `-addr=EMAIL_ADDRESS` option), send a MIME-encoded email with the testing results

For further details, please see the MITgcm Developers' HOWTO at:

`http://mitgcm.org/docs.html`

3.17.2 Automated testing

Automated testing results are produced on a regular basis and they can be viewed at:

`http://mitgcm.org/testing.html`

which also includes links to various scripts for batch job submission on a variety of different machines.

Chapter 4

Software Architecture

This chapter focuses on describing the **WRAPPER** environment within which both the core numerics and the pluggable packages operate. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The tutorial sections of this manual (see sections 3.7 and 5.3) contain more succinct, step-by-step instructions on running basic numerical experiments, of varous types, both sequentially and in parallel. For many projects simply starting from an example code and adapting it to suit a particular situation will be all that is required. The first part of this chapter discusses the MITgcm architecture at an abstract level. In the second part of the chapter we described practical details of the MITgcm implementation and of current tools and operating system features that are employed.

4.1 Overall architectural goals

Broadly, the goals of the software architecture employed in MITgcm are threefold

- We wish to be able to study a very broad range of interesting and challenging rotating fluids problems.
- We wish the model code to be readily targeted to a wide range of platforms
- On any given platform we would like to be able to achieve performance comparable to an implementation developed and specialized specifically for that platform.

These points are summarized in figure 4.1 which conveys the goals of the MITgcm design. The goals lead to a software architecture which at the high-level can be viewed as consisting of

1. A core set of numerical and support code. This is discussed in detail in section **??**.

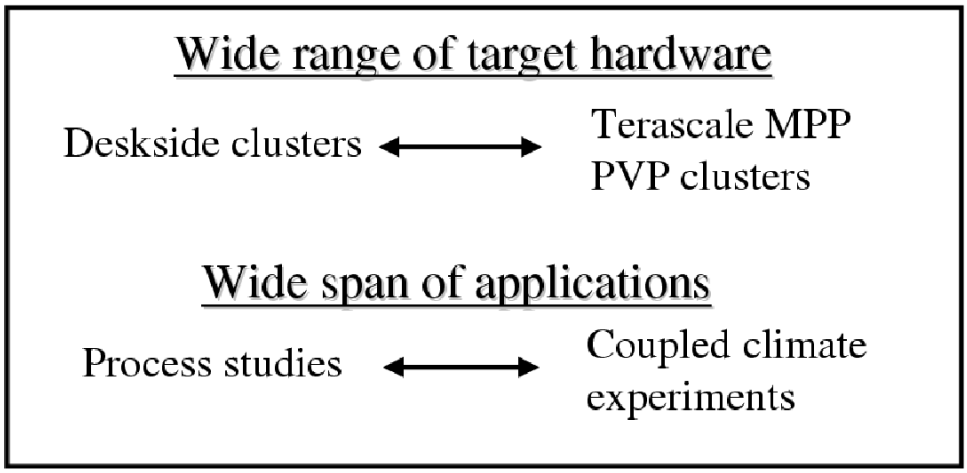


Figure 4.1: The MITgcm architecture is designed to allow simulation of a wide range of physical problems on a wide range of hardware. The computational resource requirements of the applications targeted range from around 10^7 bytes ($\approx$ 10 megabytes) of memory to 10^{11} bytes ($\approx$ 100 gigabytes). Arithmetic operation counts for the applications of interest range from 10^9 floating point operations to more than 10^{17} floating point operations.

2. A scheme for supporting optional "pluggable" **packages** (containing for example mixed-layer schemes, biogeochemical schemes, atmospheric physics). These packages are used both to overlay alternate dynamics and to introduce specialized physical content onto the core numerical code. An overview of the **package** scheme is given at the start of part **??**.

3. A support framework called **WRAPPER** (Wrappable Application Parallel Programming Environment Resource), within which the core numerics and pluggable packages operate.

This chapter focuses on describing the **WRAPPER** environment under which both the core numerics and the pluggable packages function. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The examples section of this manual (part **??**) contains more succinct, step-by-step instructions on running basic numerical experiments both sequentially and in parallel. For many projects simply starting from an example code and adapting it to suit a particular situation will be all that is required.

4.2 WRAPPER

A significant element of the software architecture utilized in MITgcm is a software superstructure and substructure collectively called the WRAPPER (Wrappable Application Parallel Programming Environment Resource). All numerical and support code in MITgcm is written to "fit" within the WRAPPER infrastructure. Writing code to "fit" within the WRAPPER means that coding has to follow certain, relatively straightforward, rules and conventions (these are discussed further in section 4.3.1).

The approach taken by the WRAPPER is illustrated in figure 4.2 which shows how the WRAPPER serves to insulate code that fits within it from architectural differences between hardware platforms and operating systems. This allows numerical code to be easily retargetted.

4.2.1 Target hardware

The WRAPPER is designed to target as broad as possible a range of computer systems. The original development of the WRAPPER took place on a multi-processor, CRAY Y-MP system. On that system, numerical code performance and scaling under the WRAPPER was in excess of that of an implementation that was tightly bound to the CRAY systems proprietary multi-tasking and micro-tasking approach. Later developments have been carried out on uniprocessor and multi-processor Sun systems with both uniform memory access (UMA) and non-uniform memory access (NUMA) designs. Significant work has also been undertaken on x86 cluster systems, Alpha processor based clustered SMP systems, and on cache-coherent NUMA (CC-NUMA) systems from Silicon Graphics. The MITgcm code, operating within the WRAPPER, is also routinely used on large scale MPP systems (for example T3E systems and IBM SP systems). In all cases numerical code, operating within the WRAPPER, performs and scales very competitively with equivalent numerical code that has been modified to contain native optimizations for a particular system ??.

4.2.2 Supporting hardware neutrality

The different systems listed in section 4.2.1 can be categorized in many different ways. For example, one common distinction is between shared-memory parallel systems (SMP's, PVP's) and distributed memory parallel systems (for example x86 clusters and large MPP systems). This is one example of a difference between compute platforms that can impact an application. Another common distinction is between vector processing systems with highly specialized CPU's and memory subsystems and commodity microprocessor based systems. There are numerous other differences, especially in relation to how parallel execution is supported. To capture the essential differences between different platforms the WRAPPER uses a *machine model*.

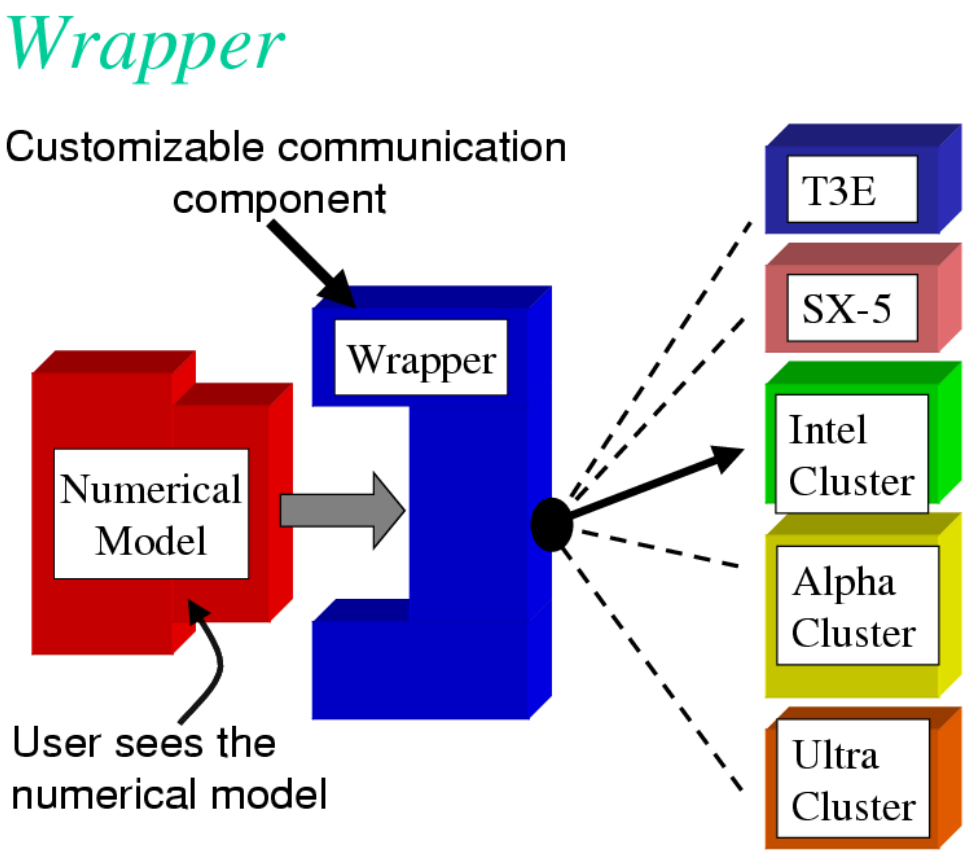


Figure 4.2: Numerical code is written to fit within a software support infrastructure called WRAPPER. The WRAPPER is portable and can be specialized for a wide range of specific target hardware and programming environments, without impacting numerical code that fits within the WRAPPER. Codes that fit within the WRAPPER can generally be made to run as fast on a particular platform as codes specially optimized for that platform.

4.2.3 WRAPPER machine model

Applications using the WRAPPER are not written to target just one particular machine (for example an IBM SP2) or just one particular family or class of machines (for example Parallel Vector Processor Systems). Instead the WRAPPER provides applications with an abstract *machine model*. The machine model is very general, however, it can easily be specialized to fit, in a computationally efficient manner, any computer architecture currently available to the scientific computing community.

4.2.4 Machine model parallelism

Codes operating under the WRAPPER target an abstract machine that is assumed to consist of one or more logical processors that can compute concurrently. Computational work is divided among the logical processors by allocating "ownership" to each processor of a certain set (or sets) of calculations. Each set of calculations owned by a particular processor is associated with a specific region of the physical space that is being simulated, only one processor will be associated with each such region (domain decomposition).

In a strict sense the logical processors over which work is divided do not need to correspond to physical processors. It is perfectly possible to execute a configuration decomposed for multiple logical processors on a single physical processor. This helps ensure that numerical code that is written to fit within the WRAPPER will parallelize with no additional effort and is also useful when debugging codes. Generally, however, the computational domain will be subdivided over multiple logical processors in order to then bind those logical processors to physical processor resources that can compute in parallel.

4.2.4.1 Tiles

Computationally, associated with each region of physical space allocated to a particular logical processor, there will be data structures (arrays, scalar variables etc...) that hold the simulated state of that region. We refer to these data structures as being **owned** by the processor to which their associated region of physical space has been allocated. Individual regions that are allocated to processors are called **tiles**. A processor can own more than one tile. Figure 4.3 shows a physical domain being mapped to a set of logical processors, with each processors owning a single region of the domain (a single tile). Except for periods of communication and coordination, each processor computes autonomously, working only with data from the tile (or tiles) that the processor owns. When multiple tiles are alloted to a single processor, each tile is computed on independently of the other tiles, in a sequential fashion.

4.2.4.2 Tile layout

Tiles consist of an interior region and an overlap region. The overlap region of a tile corresponds to the interior region of an adjacent tile. In figure 4.4 each tile

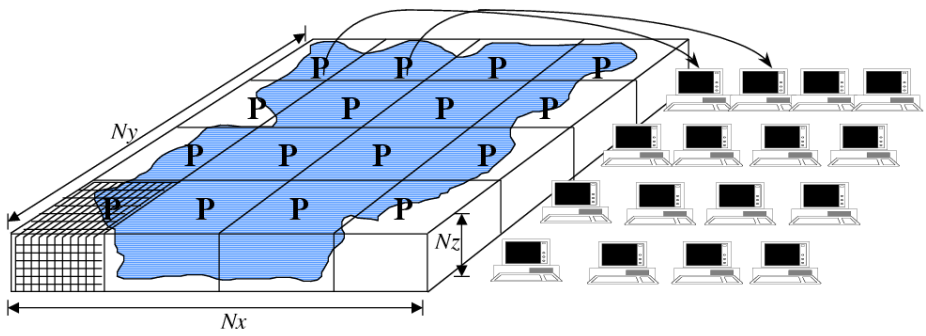


Figure 4.3: The WRAPPER provides support for one and two dimensional decompositions of grid-point domains. The figure shows a hypothetical domain of total size $N_x N_y N_z$. This hypothetical domain is decomposed in two-dimensions along the N_x and N_y directions. The resulting **tiles** are **owned** by different processors. The **owning** processors perform the arithmetic operations associated with a **tile**. Although not illustrated here, a single processor can **own** several **tiles**. Whenever a processor wishes to transfer data between tiles or communicate with other processors it calls a WRAPPER supplied function.

would own the region within the black square and hold duplicate information for overlap regions extending into the tiles to the north, south, east and west. During computational phases a processor will reference data in an overlap region whenever it requires values that outside the domain it owns. Periodically processors will make calls to WRAPPER functions to communicate data between tiles, in order to keep the overlap regions up to date (see section 4.2.8). The WRAPPER functions can use a variety of different mechanisms to communicate data between tiles.

4.2.5 Communication mechanisms

Logical processors are assumed to be able to exchange information between tiles and between each other using at least one of two possible mechanisms.

- **Shared memory communication**. Under this mode of communication data transfers are assumed to be possible using direct addressing of regions of memory. In this case a CPU is able to read (and write) directly to regions of memory "owned" by another CPU using simple programming language level assignment operations of the the sort shown in figure 4.5. In this way one CPU (CPU1 in the figure) can communicate information to another CPU (CPU2 in the figure) by assigning a particular value to a particular memory location.

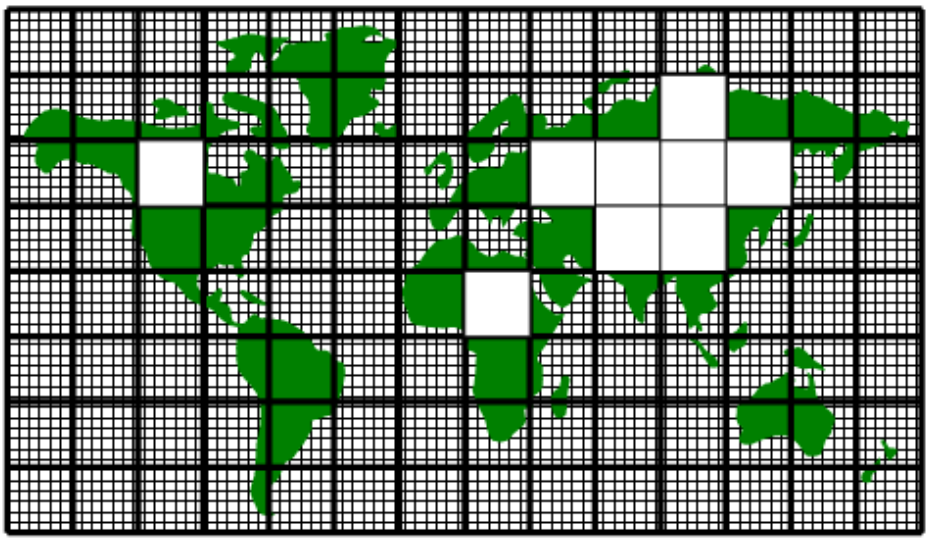


Figure 4.4: A global grid subdivided into tiles. Tiles contain a interior region and an overlap region. Overlap regions are periodically updated from neighboring tiles.

- **Distributed memory communication**. Under this mode of communication there is no mechanism, at the application code level, for directly addressing regions of memory owned and visible to another CPU. Instead a communication library must be used as illustrated in figure 4.6. In this case CPU's must call a function in the API of the communication library to communicate data from a tile that it owns to a tile that another CPU owns. By default the WRAPPER binds to the MPI communication library **??** for this style of communication.

The WRAPPER assumes that communication will use one of these two styles of communication. The underlying hardware and operating system support for the style used is not specified and can vary from system to system.

4.2.6 Shared memory communication

Under shared communication independent CPU's are operating on the exact same global address space at the application level. This means that CPU 1 can directly write into global data structures that CPU 2 "owns" using a simple assignment at the application level. This is the model of memory access is supported at the basic system design level in "shared-memory" systems such as PVP systems, SMP systems, and on distributed shared memory systems (the SGI Origin). On such systems the WRAPPER will generally use simple

```
  CPU1                    |        CPU2
  ====                    |        ====
                          |
a(3) = 8                  |        WHILE ( a(3) .NE. 8 )
                          |         WAIT
                          |        END WHILE
                          |
```

Figure 4.5: In the WRAPPER shared memory communication model, simple writes to an array can be made to be visible to other CPU's at the application code level. So that for example, if one CPU (CPU1 in the figure above) writes the value 8 to element 3 of array a, then other CPU's (for example CPU2 in the figure above) will be able to see the value 8 when they read from $a(3)$. This provides a very low latency and high bandwidth communication mechanism.

```
  CPU1                    |        CPU2
  ====                    |        ====
                          |
a(3) = 8                  |        WHILE ( a(3) .NE. 8 )
CALL SEND( CPU2,a(3) )    |         CALL RECV( CPU1, a(3) )
                          |        END WHILE
                          |
```

Figure 4.6: In the WRAPPER distributed memory communication model data can not be made directly visible to other CPU's. If one CPU writes the value 8 to element 3 of array a, then at least one of CPU1 and/or CPU2 in the figure above will need to call a bespoke communication library in order for the updated value to be communicated between CPU's.

read and write statements to access directly application data structures when communicating between CPU's.

In a system where assignments statements, like the one in figure 4.5 map directly to hardware instructions that transport data between CPU and memory banks, this can be a very efficient mechanism for communication. In this case two CPU's, CPU1 and CPU2, can communicate simply be reading and writing to an agreed location and following a few basic rules. The latency of this sort of communication is generally not that much higher than the hardware latency of other memory accesses on the system. The bandwidth available between CPU's communicating in this way can be close to the bandwidth of the systems main-memory interconnect. This can make this method of communication very efficient provided it is used appropriately.

4.2.6.1 Memory consistency

When using shared memory communication between multiple processors the WRAPPER level shields user applications from certain counter-intuitive system behaviors. In particular, one issue the WRAPPER layer must deal with is a systems memory model. In general the order of reads and writes expressed by the textual order of an application code may not be the ordering of instructions executed by the processor performing the application. The processor performing the application instructions will always operate so that, for the application instructions the processor is executing, any reordering is not apparent. However, in general machines are often designed so that reordering of instructions is not hidden from other second processors. This means that, in general, even on a shared memory system two processors can observe inconsistent memory values.

The issue of memory consistency between multiple processors is discussed at length in many computer science papers, however, from a practical point of view, in order to deal with this issue, shared memory machines all provide some mechanism to enforce memory consistency when it is needed. The exact mechanism employed will vary between systems. For communication using shared memory, the WRAPPER provides a place to invoke the appropriate mechanism to ensure memory consistency for a particular platform.

4.2.6.2 Cache effects and false sharing

Shared-memory machines often have local to processor memory caches which contain mirrored copies of main memory. Automatic cache-coherence protocols are used to maintain consistency between caches on different processors. These cache-coherence protocols typically enforce consistency between regions of memory with large granularity (typically 128 or 256 byte chunks). The coherency protocols employed can be expensive relative to other memory accesses and so care is taken in the WRAPPER (by padding synchronization structures appropriately) to avoid unnecessary coherence traffic.

4.2.6.3 Operating system support for shared memory.

Applications running under multiple threads within a single process can use shared memory communication. In this case *all* the memory locations in an application are potentially visible to all the compute threads. Multiple threads operating within a single process is the standard mechanism for supporting shared memory that the WRAPPER utilizes. Configuring and launching code to run in multi-threaded mode on specific platforms is discussed in section **??**. However, on many systems, potentially very efficient mechanisms for using shared memory communication between multiple processes (in contrast to multiple threads within a single process) also exist. In most cases this works by making a limited region of memory shared between processes. The MMAP **??** and IPC **??** facilities in UNIX systems provide this capability as do vendor specific tools like LAPI **??** and IMC **??**. Extensions exist for the WRAPPER that allow these mechanisms to be used for shared memory communication. However, these mechanisms are not distributed with the default WRAPPER sources, because of their proprietary nature.

4.2.7 Distributed memory communication

Many parallel systems are not constructed in a way where it is possible or practical for an application to use shared memory for communication. For example cluster systems consist of individual computers connected by a fast network. On such systems their is no notion of shared memory at the system level. For this sort of system the WRAPPER provides support for communication based on a bespoke communication library (see figure 4.6). The default communication library used is MPI **??**. However, it is relatively straightforward to implement bindings to optimized platform specific communication libraries. For example the work described in **??** substituted standard MPI communication for a highly optimized library.

4.2.8 Communication primitives

Optimized communication support is assumed to be possibly available for a small number of communication operations. It is assumed that communication performance optimizations can be achieved by optimizing a small number of communication primitives. Three optimizable primitives are provided by the WRAPPER

- **EXCHANGE** This operation is used to transfer data between interior and overlap regions of neighboring tiles. A number of different forms of this operation are supported. These different forms handle

  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.

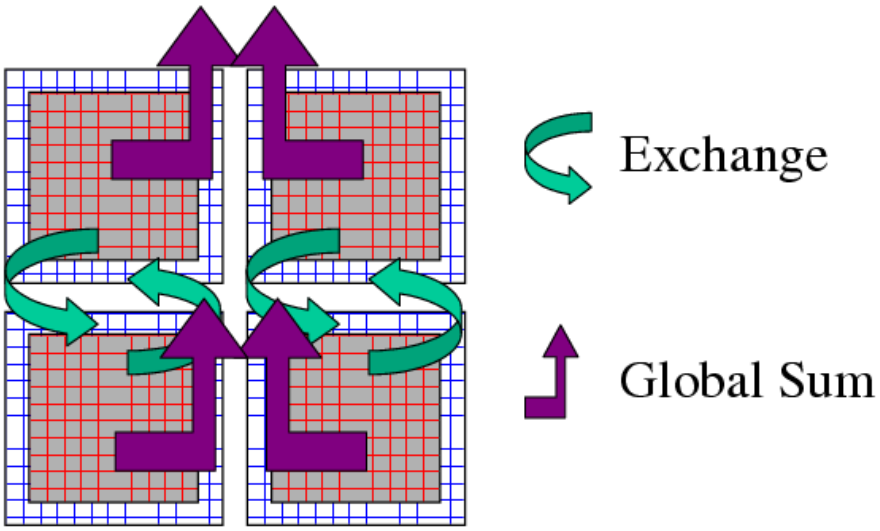


Figure 4.7: Three performance critical parallel primitives are provided by the WRAPPER. These primitives are always used to communicate data between tiles. The figure shows four tiles. The curved arrows indicate exchange primitives which transfer data between the overlap regions at tile edges and interior regions for nearest-neighbor tiles. The straight arrows symbolize global sum operations which connect all tiles. The global sum operation provides both a key arithmetic primitive and can serve as a synchronization primitive. A third barrier primitive is also provided, it behaves much like the global sum primitive.

  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.
  - Transformation operations required when transporting data between different grid regions. Transferring data between faces of a cube-sphere grid, for example, involves a rotation of vector components.
  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.

- **GLOBAL SUM** The global sum operation is a central arithmetic operation for the pressure inversion phase of the MITgcm algorithm. For certain configurations scaling can be highly sensitive to the performance of the global sum primitive. This operation is a collective operation involving all tiles of the simulated domain. Different forms of the global sum primitive exist for handling
  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.
  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.
  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.

- **BARRIER** The WRAPPER provides a global synchronization function called barrier. This is used to synchronize computations over all tiles. The **BARRIER** and **GLOBAL SUM** primitives have much in common and in some cases use the same underlying code.

4.2.9 Memory architecture

The WRAPPER machine model is aimed to target efficiently systems with highly pipelined memory architectures and systems with deep memory hierarchies that favor memory reuse. This is achieved by supporting a flexible tiling strategy as shown in figure 4.8. Within a CPU computations are carried out sequentially on each tile in turn. By reshaping tiles according to the target platform it is possible to automatically tune code to improve memory performance. On a vector machine a given domain might be sub-divided into a few long, thin regions. On a commodity microprocessor based system, however, the same region could be simulated use many more smaller sub-domains.

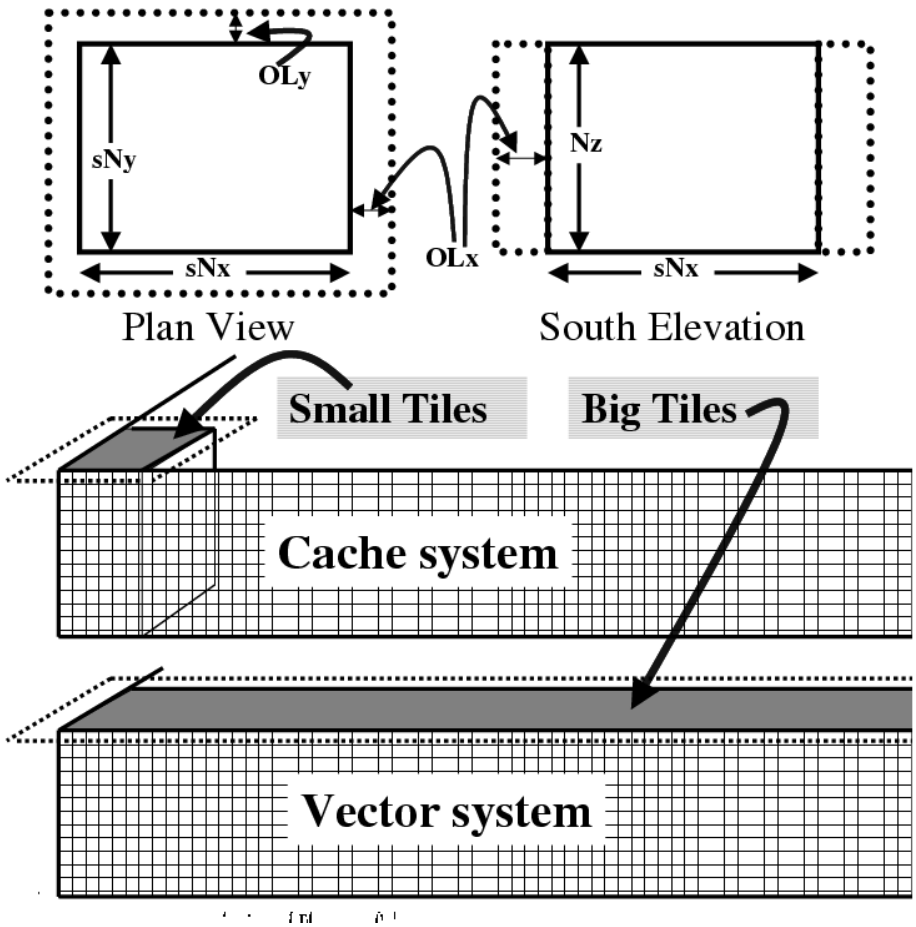


Figure 4.8: The tiling strategy that the WRAPPER supports allows tiles to be shaped to suit the underlying system memory architecture. Compact tiles that lead to greater memory reuse can be used on cache based systems (upper half of figure) with deep memory hierarchies, long tiles with large inner loops can be used to exploit vector systems having highly pipelined memory systems.

4.2.10 Summary

Following the discussion above, the machine model that the WRAPPER presents to an application has the following characteristics

- The machine consists of one or more logical processors.
- Each processor operates on tiles that it owns.
- A processor may own more than one tile.
- Processors may compute concurrently.
- Exchange of information between tiles is handled by the machine (WRAPPER) not by the application.

Behind the scenes this allows the WRAPPER to adapt the machine model functions to exploit hardware on which

- Processors may be able to communicate very efficiently with each other using shared memory.
- An alternative communication mechanism based on a relatively simple inter-process communication API may be required.
- Shared memory may not necessarily obey sequential consistency, however some mechanism will exist for enforcing memory consistency.
- Memory consistency that is enforced at the hardware level may be expensive. Unnecessary triggering of consistency protocols should be avoided.
- Memory access patterns may need to either repetitive or highly pipelined for optimum hardware performance.

This generic model captures the essential hardware ingredients of almost all successful scientific computer systems designed in the last 50 years.

4.3 Using the WRAPPER

In order to support maximum portability the WRAPPER is implemented primarily in sequential Fortran 77. At a practical level the key steps provided by the WRAPPER are

1. specifying how a domain will be decomposed
2. starting a code in either sequential or parallel modes of operations
3. controlling communication between tiles and between concurrently computing CPU's.

This section describes the details of each of these operations. Section 4.3.1 explains how the way in which a domain is decomposed (or composed) is expressed. Section **??** describes practical details of running codes in various different parallel modes on contemporary computer systems. Section **??** explains the internal information that the WRAPPER uses to control how information is communicated between tiles.

4.3.1 Specifying a domain decomposition

At its heart much of the WRAPPER works only in terms of a collection of tiles which are interconnected to each other. This is also true of application code operating within the WRAPPER. Application code is written as a series of compute operations, each of which operates on a single tile. If application code needs to perform operations involving data associated with another tile, it uses a WRAPPER function to obtain that data. The specification of how a global domain is constructed from tiles or alternatively how a global domain is decomposed into tiles is made in the file *SIZE.h*. This file defines the following parameters

Parameters: *sNx, sNy, OLx, OLy, nSx, nSy, nPx, nPy* File: *model/inc/SIZE.h*

Together these parameters define a tiling decomposition of the style shown in figure 4.9. The parameters *sNx* and *sNy* define the size of an individual tile. The parameters *OLx* and *OLy* define the maximum size of the overlap extent. This must be set to the maximum width of the computation stencil that the numerical code finite-difference operations require between overlap region updates. The maximum overlap required by any of the operations in the MITgcm code distributed with this release is three grid points. This is set by the requirements of the ∇^4 dissipation and diffusion operator. Code modifications and enhancements that involve adding wide finite-difference stencils may require increasing *OLx* and *OLy*. Setting *OLx* and *OLy* to a too large value will decrease code performance (because redundant computations will be performed), however it will not cause any other problems.

The parameters *nSx* and *nSy* specify the number of tiles that will be created within a single process. Each of these tiles will have internal dimensions of *sNx* and *sNy*. If, when the code is executed, these tiles are allocated to different threads of a process that are then bound to different physical processors (see the multi-threaded execution discussion in section 4.3.2) then computation will be performed concurrently on each tile. However, it is also possible to run the same decomposition within a process running a single thread on a single processor. In this case the tiles will be computed over sequentially. If the decomposition is run in a single process running multiple threads but attached to a single physical processor, then, in general, the computation for different tiles will be interleaved by system level software. This too is a valid mode of operation.

The parameters *sNx, sNy, OLx, OLy, nSx* and*nSy* are used extensively by numerical code. The settings of *sNx, sNy, OLx* and *OLy* are used to form the loop ranges for many numerical calculations and to provide dimensions for arrays holding numerical state. The *nSx* and*nSy* are used in conjunction with the thread number parameter *myThid*. Much of the numerical code operating within the WRAPPER takes the form

```
DO bj=myByLo(myThid),myByHi(myThid)
 DO bi=myBxLo(myThid),myBxHi(myThid)
```

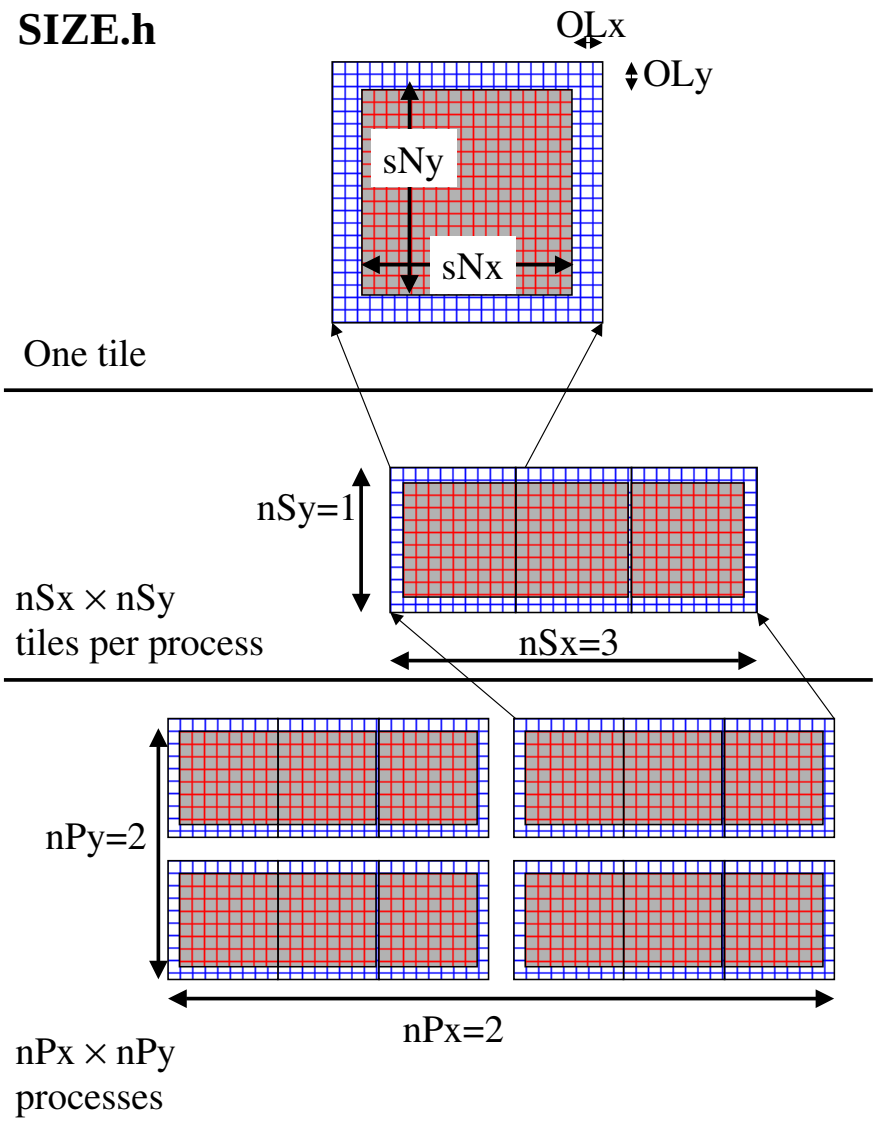


Figure 4.9: The three level domain decomposition hierarchy employed by the WRAPPER. A domain is composed of tiles. Multiple tiles can be allocated to a single process. Multiple processes can exist, each with multiple tiles. Tiles within a process can be spread over multiple compute threads.

```
    :
    a block of computations ranging
    over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
    :
 ENDDO
ENDDO

communication code to sum a number or maybe update
tile overlap regions

DO bj=myByLo(myThid),myByHi(myThid)
 DO bi=myBxLo(myThid),myBxHi(myThid)
    :
    another block of computations ranging
    over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
    :
 ENDDO
ENDDO
```

The variables *myBxLo(myThid), myBxHi(myThid), myByLo(myThid)* and *myByHi(myThid)* set the bounds of the loops in *bi* and *bj* in this schematic. These variables specify the subset of the tiles in the range 1,*nSx* and 1,*nSy* that the logical processor bound to thread number *myThid* owns. The thread number variable *myThid* ranges from 1 to the total number of threads requested at execution time. For each value of *myThid* the loop scheme above will step sequentially through the tiles owned by that thread. However, different threads will have different ranges of tiles assigned to them, so that separate threads can compute iterations of the *bi*, *bj* loop concurrently. Within a *bi*, *bj* loop computation is performed concurrently over as many processes and threads as there are physical processors available to compute.

An exception to the the use of *bi* and *bj* in loops arises in the exchange routines used when the exch2 package is used with the cubed sphere. In this case *bj* is generally set to 1 and the loop runs from 1,*bi*. Within the loop *bi* is used to retrieve the tile number, which is then used to reference exchange parameters.

The amount of computation that can be embedded a single loop over *bi* and *bj* varies for different parts of the MITgcm algorithm. Figure 4.10 shows a code extract from the two-dimensional implicit elliptic solver. This portion of the code computes the l2Norm of a vector whose elements are held in the array *cg2d_r* writing the final result to scalar variable *err*. In this case, because the l2norm requires a global reduction, the *bi*,*bj* loop only contains one statement. This computation phase is then followed by a communication phase in which all threads and processes must participate. However, in other areas of the MITgcm code entries subsections of code are within a single *bi*,*bj* loop. For example the evaluation of all the momentum equation prognostic terms (see *S/R DYNAMICS()*) is within a single *bi*,*bj* loop.

```
      REAL*8  cg2d_r(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
      REAL*8  err
          :
          :
        other computations
          :
          :
      err = 0.
      DO bj=myByLo(myThid),myByHi(myThid)
       DO bi=myBxLo(myThid),myBxHi(myThid)
        DO J=1,sNy
         DO I=1,sNx
           err             = err             +
     &     cg2d_r(I,J,bi,bj)*cg2d_r(I,J,bi,bj)
         ENDDO
        ENDDO
       ENDDO
      ENDDO

      CALL GLOBAL_SUM_R8( err    , myThid )
      err = SQRT(err)
```

Figure 4.10: Example of numerical code for calculating the l2-norm of a vector within the WRAPPER. Notice that under the WRAPPER arrays such as *cg2d_r* have two extra trailing dimensions. These right most indices are tile indexes. Different threads with a single process operate on different ranges of tile index, as controlled by the settings of *myByLo, myByHi, myBxLo* and *myBxHi*.

The final decomposition parameters are *nPx* and *nPy*. These parameters are used to indicate to the WRAPPER level how many processes (each with *nSx*×*nSy* tiles) will be used for this simulation. This information is needed during initialization and during I/O phases. However, unlike the variables *sNx, sNy, OLx, OLy, nSx* and *nSy* the values of *nPx* and *nPy* are absent from the core numerical and support code.

4.3.1.1 Examples of *SIZE.h* specifications

The following different *SIZE.h* parameter setting illustrate how to interpret the values of *sNx, sNy, OLx, OLy, nSx, nSy, nPx* and *nPy*.

1.
```
      PARAMETER (
     &           sNx =  90,
     &           sNy =  40,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1)
```
This sets up a single tile with x-dimension of ninety grid points, y-dimension of forty grid points, and x and y overlaps of three grid points each.

2.
```
      PARAMETER (
     &           sNx =  45,
     &           sNy =  20,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   2,
     &           nPy =   2)
```
This sets up tiles with x-dimension of forty-five grid points, y-dimension of twenty grid points, and x and y overlaps of three grid points each. There are four tiles allocated to four separate processes (*nPx=2,nPy=2*) and arranged so that the global domain size is again ninety grid points in x and forty grid points in y. In general the formula for global grid size (held in model variables *Nx* and *Ny*) is
```
      Nx  = sNx*nSx*nPx
      Ny  = sNy*nSy*nPy
```

3.
```
      PARAMETER (
     &           sNx =  90,
     &           sNy =  10,
```

```
&           OLx =   3,
&           OLy =   3,
&           nSx =   1,
&           nSy =   2,
&           nPx =   1,
&           nPy =   2)
```

This sets up tiles with x-dimension of ninety grid points, y-dimension of ten grid points, and x and y overlaps of three grid points each. There are four tiles allocated to two separate processes (*nPy=2*) each of which has two separate sub-domains *nSy=2*, The global domain size is again ninety grid points in x and forty grid points in y. The two sub-domains in each process will be computed sequentially if they are given to a single thread within a single process. Alternatively if the code is invoked with multiple threads per process the two domains in y may be computed concurrently.

4.
```
 PARAMETER (
&           sNx =  32,
&           sNy =  32,
&           OLx =   3,
&           OLy =   3,
&           nSx =   6,
&           nSy =   1,
&           nPx =   1,
&           nPy =   1)
```

This sets up tiles with x-dimension of thirty-two grid points, y-dimension of thirty-two grid points, and x and y overlaps of three grid points each. There are six tiles allocated to six separate logical processors (*nSx=6*). This set of values can be used for a cube sphere calculation. Each tile of size 32×32 represents a face of the cube. Initializing the tile connectivity correctly (see section 4.3.3.3. allows the rotations associated with moving between the six cube faces to be embedded within the tile-tile communication code.

4.3.2 Starting the code

When code is started under the WRAPPER, execution begins in a main routine *eesupp/src/main.F* that is owned by the WRAPPER. Control is transferred to the application through a routine called *THE_MODEL_MAIN()* once the WRAPPER has initialized correctly and has created the necessary variables to support subsequent calls to communication routines by the application code. The startup calling sequence followed by the WRAPPER is shown in figure 4.11.

```
MAIN
|
|--EEBOOT               :: WRAPPER initialization
|  |
|  |-- EEBOOT_MINMAL    :: Minimal startup. Just enough to
|  |                       allow basic I/O.
|  |-- EEINTRO_MSG      :: Write startup greeting.
|  |
|  |-- EESET_PARMS      :: Set WRAPPER parameters
|  |
|  |-- EEWRITE_EEENV    :: Print WRAPPER parameter settings
|  |
|  |-- INI_PROCS        :: Associate processes with grid regions.
|  |
|  |-- INI_THREADING_ENVIRONMENT   :: Associate threads with grid regions.
|       |
|       |--INI_COMMUNICATION_PATTERNS :: Initialize between tile
|                                     :: communication data structures
|
|
|--CHECK_THREADS    :: Validate multiple thread start up.
|
|--THE_MODEL_MAIN   :: Numerical code top-level driver routine
```

Figure 4.11: Main stages of the WRAPPER startup procedure. This process proceeds transfer of control to application code, which occurs through the procedure *THE_MODEL_MAIN()*.

4.3.2.1 Multi-threaded execution

Prior to transferring control to the procedure *THE_MODEL_MAIN()* the WRAPPER may cause several coarse grain threads to be initialized. The routine *THE_MODEL_MAIN()* is called once for each thread and is passed a single stack argument which is the thread number, stored in the variable *myThid*. In addition to specifying a decomposition with multiple tiles per process (see section 4.3.1) configuring and starting a code to run using multiple threads requires the following steps.

Compilation First the code must be compiled with appropriate multi-threading directives active in the file *main.F* and with appropriate compiler flags to request multi-threading support. The header files *MAIN_PDIRECTIVES1.h* and *MAIN_PDIRECTIVES2.h* contain directives compatible with compilers for Sun, Compaq, SGI, Hewlett-Packard SMP systems and CRAY PVP systems. These directives can be activated by using compile time directives *-DTARGET_SUN*, *-DTARGET_DEC*, *-DTARGET_SGI*, *-DTARGET_HP* or *-DTARGET_CRAY_VECTOR* respectively. Compiler options for invoking multi-threaded compilation vary from system to system and from compiler to compiler. The options will be described in the individual compiler documentation. For the Fortran compiler from Sun the following options are needed to correctly compile multi-threaded code

```
-stackvar -explicitpar -vpara -noautopar
```

These options are specific to the Sun compiler. Other compilers will use different syntax that will be described in their documentation. The effect of these options is as follows

1. **-stackvar** Causes all local variables to be allocated in stack storage. This is necessary for local variables to ensure that they are private to their thread. Note, when using this option it may be necessary to override the default limit on stack-size that the operating system assigns to a process. This can normally be done by changing the settings of the command shells *stack-size* limit variable. However, on some systems changing this limit will require privileged administrator access to modify system parameters.

2. **-explicitpar** Requests that multiple threads be spawned in response to explicit directives in the application code. These directives are inserted with syntax appropriate to the particular target platform when, for example, the *-DTARGET_SUN* flag is selected.

3. **-vpara** This causes the compiler to describe the multi-threaded configuration it is creating. This is not required but it can be useful when trouble shooting.

4. **-noautopar** This inhibits any automatic multi-threaded parallelization the compiler may otherwise generate.

An example of valid settings for the *eedata* file for a domain with two sub-domains in y and running with two threads is shown below

```
nTx=1,nTy=2
```

This set of values will cause computations to stay within a single thread when moving across the *nSx* sub-domains. In the y-direction, however, sub-domains will be split equally between two threads.

Multi-threading files and parameters The following files and variables are used in setting up multi-threaded execution.

File: *eesupp/inc/MAIN_PDIRECTIVES1.h*
File: *eesupp/inc/MAIN_PDIRECTIVES2.h*
File: *model/src/THE_MODEL_MAIN.F*
File: *eesupp/src/MAIN.F*
File: *tools/genmake2*
File: *eedata*
CPP: *TARGET_SUN*
CPP: *TARGET_DEC*
CPP: *TARGET_HP*
CPP: *TARGET_SGI*
CPP: *TARGET_CRAY_VECTOR*
Parameter: *nTx*
Parameter: *nTy*

4.3.2.2 Multi-process execution

Despite its appealing programming model, multi-threaded execution remains less common then multi-process execution. One major reason for this is that many system libraries are still not "thread-safe". This means that for example on some systems it is not safe to call system routines to do I/O when running in multi-threaded mode, except for in a limited set of circumstances. Another reason is that support for multi-threaded programming models varies between systems.

Multi-process execution is more ubiquitous. In order to run code in a multi-process configuration a decomposition specification (see section 4.3.1) is given (in which the at least one of the parameters *nPx* or *nPy* will be greater than one) and then, as for multi-threaded operation, appropriate compile time and run time steps must be taken.

Compilation Multi-process execution under the WRAPPER assumes that the portable, MPI libraries are available for controlling the start-up of multiple processes. The MPI libraries are not required, although they are usually used, for performance critical communication. However, in order to simplify the task of controlling and coordinating the start up of a large number (hundreds and

possibly even thousands) of copies of the same program, MPI is used. The calls to the MPI multi-process startup routines must be activated at compile time. Currently MPI libraries are invoked by specifying the appropriate options file with the `-of` flag when running the *genmake2* script, which generates the Makefile for compiling and linking MITgcm. (Previously this was done by setting the *ALLOW_USE_MPI* and *ALWAYS_USE_MPI* flags in the *CPP_EEOPTIONS.h* file.) More detailed information about the use of *genmake2* for specifying local compiler flags is located in section 3.16.2.

Directory: *tools/build_options*
File: *tools/genmake2*

Execution The mechanics of starting a program in multi-process mode under MPI is not standardized. Documentation associated with the distribution of MPI installed on a system will describe how to start a program using that distribution. For the free, open-source MPICH system the MITgcm program is started using a command such as

```
mpirun -np 64 -machinefile mf ./mitgcmuv
```

In this example the text *-np 64* specifies the number of processes that will be created. The numeric value *64* must be equal to the product of the processor grid settings of *nPx* and *nPy* in the file *SIZE.h*. The parameter *mf* specifies that a text file called "mf" will be read to get a list of processor names on which the sixty-four processes will execute. The syntax of this file is specified by the MPI distribution.

File: *SIZE.h*
Parameter: *nPx*
Parameter: *nPy*

Environment variables On most systems multi-threaded execution also requires the setting of a special environment variable. On many machines this variable is called PARALLEL and its values should be set to the number of parallel threads required. Generally the help pages associated with the multi-threaded compiler on a machine will explain how to set the required environment variables for that machines.

Runtime input parameters Finally the file *eedata* needs to be configured to indicate the number of threads to be used in the x and y directions. The variables nTx and nTy in this file are used to specify the information required. The product of nTx and nTy must be equal to the number of threads spawned i.e. the setting of the environment variable PARALLEL. The value of nTx must subdivide the number of sub-domains in x (nSx) exactly. The value of nTy

must subdivide the number of sub-domains in y (*nSy*) exactly. The multiprocess startup of the MITgcm executable *mitgcmuv* is controlled by the routines *EEBOOT_MINIMAL()* and *INI_PROCS()*. The first routine performs basic steps required to make sure each process is started and has a textual output stream associated with it. By default two output files are opened for each process with names **STDOUT.NNNN** and **STDERR.NNNN**. The **NNNNN** part of the name is filled in with the process number so that process number 0 will create output files **STDOUT.0000** and **STDERR.0000**, process number 1 will create output files **STDOUT.0001** and **STDERR.0001** etc... These files are used for reporting status and configuration information and for reporting error conditions on a process by process basis. The *EEBOOT_MINIMAL()* procedure also sets the variables *myProcId* and *MPI_COMM_MODEL*. These variables are related to processor identification are are used later in the routine *INI_PROCS()* to allocate tiles to processes.

Allocation of processes to tiles in controlled by the routine *INI_PROCS()*. For each process this routine sets the variables *myXGlobalLo* and *myYGlobalLo*. These variables specify in index space the coordinates of the southernmost and westernmost corner of the southernmost and westernmost tile owned by this process. The variables *pidW*, *pidE*, *pidS* and *pidN* are also set in this routine. These are used to identify processes holding tiles to the west, east, south and north of this process. These values are stored in global storage in the header file *EESUPPORT.h* for use by communication routines. The above does not hold when the exch2 package is used – exch2 sets its own parameters to specify the global indices of tiles and their relationships to each other. See the documentation on the exch2 package (6.18) for details.

File: *eesupp/src/eeboot_minimal.F*
File: *eesupp/src/ini_procs.F*
File: *eesupp/inc/EESUPPORT.h*
Parameter: *myProcId*
Parameter: *MPI_COMM_MODEL*
Parameter: *myXGlobalLo*
Parameter: *myYGlobalLo*
Parameter: *pidW*
Parameter: *pidE*
Parameter: *pidS*
Parameter: *pidN*

4.3.3 Controlling communication

The WRAPPER maintains internal information that is used for communication operations and that can be customized for different platforms. This section describes the information that is held and used.

1. **Tile-tile connectivity information** For each tile the WRAPPER sets a flag that sets the tile number to the north, south, east and west of that tile.

This number is unique over all tiles in a configuration. Except when using the cubed sphere and the exch2 package, the number is held in the variables *tileNo* (this holds the tiles own number), *tileNoN*, *tileNoS*, *tileNoE* and *tileNoW*. A parameter is also stored with each tile that specifies the type of communication that is used between tiles. This information is held in the variables *tileCommModeN*, *tileCommModeS*, *tileCommModeE* and *tileCommModeW*. This latter set of variables can take one of the following values *COMM_NONE*, *COMM_MSG*, *COMM_PUT* and *COMM_GET*. A value of *COMM_NONE* is used to indicate that a tile has no neighbor to communicate with on a particular face. A value of *COMM_MSG* is used to indicated that some form of distributed memory communication is required to communicate between these tile faces (see section 4.2.7). A value of *COMM_PUT* or *COMM_GET* is used to indicate forms of shared memory communication (see section 4.2.6). The *COMM_PUT* value indicates that a CPU should communicate by writing to data structures owned by another CPU. A *COMM_GET* value indicates that a CPU should communicate by reading from data structures owned by another CPU. These flags affect the behavior of the WRAPPER exchange primitive (see figure 4.7). The routine *ini_communication_patterns()* is responsible for setting the communication mode values for each tile.

When using the cubed sphere configuration with the exch2 package, the relationships between tiles and their communication methods are set by the package in other variables. See the exch2 package documentation (6.18 for details.

File: *eesupp/src/ini_communication_patterns.F*
File: *eesupp/inc/EESUPPORT.h*
Parameter: *tileNo*
Parameter: *tileNoE*
Parameter: *tileNoW*
Parameter: *tileNoN*
Parameter: *tileNoS*
Parameter: *tileCommModeE*
Parameter: *tileCommModeW*
Parameter: *tileCommModeN*
Parameter: *tileCommModeS*

2. **MP directives** The WRAPPER transfers control to numerical application code through the routine *THE_MODEL_MAIN*. This routine is called in a way that allows for it to be invoked by several threads. Support for this is based on using multi-processing (MP) compiler directives. Most commercially available Fortran compilers support the generation of code to spawn multiple threads through some form of compiler directives. As this is generally much more convenient than writing code to interface to operating system libraries to explicitly spawn threads, and on some sys-

tems this may be the only method available the WRAPPER is distributed with template MP directives for a number of systems.

These directives are inserted into the code just before and after the transfer of control to numerical algorithm code through the routine *THE_MODEL_MAIN*. Figure 4.12 shows an example of the code that performs this process for a Silicon Graphics system. This code is extracted from the files *main.F* and *MAIN_PDIRECTIVES1.h*. The variable *nThreads* specifies how many instances of the routine *THE_MODEL_MAIN* will be created. The value of *nThreads* is set in the routine *INI_THREADING_ENVIRONMENT*. The value is set equal to the the product of the parameters *nTx* and *nTy* that are read from the file *eedata*. If the value of *nThreads* is inconsistent with the number of threads requested from the operating system (for example by using an environment variable as described in section **??**) then usually an error will be reported by the routine *CHECK_THREADS*.

File: *eesupp/src/ini_threading_environment.F*
File: *eesupp/src/check_threads.F*
File: *eesupp/src/main.F*
File: *eesupp/inc/MAIN_PDIRECTIVES1.h*
File: *eedata*
Parameter: *nThreads*
Parameter: *nTx*
Parameter: *nTy*

3. **memsync flags** As discussed in section 4.2.6.1, when using shared memory, a low-level system function may be need to force memory consistency. The routine *MEMSYNC()* is used for this purpose. This routine should not need modifying and the information below is only provided for completeness. A logical parameter *exchNeedsMemSync* set in the routine *INI_COMMUNICATION_PATTERNS()* controls whether the *MEMSYNC()* primitive is called. In general this routine is only used for multi-threaded execution. The code that goes into the *MEMSYNC()* routine is specific to the compiler and processor being used for multi-threaded execution and in general must be written using a short code snippet of assembly language. For an Ultra Sparc system the following code snippet is used

```
asm("membar #LoadStore|#StoreStore");
```

for an Alpha based system the equivalent code reads

```
asm("mb");
```

while on an x86 system the following code is required

```
asm("lock; addl $0,0(%%esp)": : :"memory")
```

4. **Cache line size** As discussed in section 4.2.6.2, milti-threaded codes explicitly avoid penalties associated with excessive coherence traffic on an SMP system. To do this the shared memory data structures used by the *GLOBAL_SUM*, *GLOBAL_MAX* and *BARRIER* routines are padded. The variables that control the padding are set in the header file *EEPARAMS.h*. These variables are called *cacheLineSize*, *lShare1*, *lShare4* and *lShare8*. The default values should not normally need changing.

5. **_BARRIER** This is a CPP macro that is expanded to a call to a routine which synchronizes all the logical processors running under the WRAPPER. Using a macro here preserves flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure *BARRIER()*. The default setting for the _BARRIER macro is given in the file *CPP_EEMACROS.h*.

6. **_GSUM** This is a CPP macro that is expanded to a call to a routine which sums up a floating point number over all the logical processors running under the WRAPPER. Using a macro here provides extra flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure *GLOBAL_SUM_R8()* (for 64-bit floating point operands) or *GLOBAL_SUM_R4()* (for 32-bit floating point operands). The default setting for the _GSUM macro is given in the file *CPP_EEMACROS.h*. The _GSUM macro is a performance critical operation, especially for large processor count, small tile size configurations. The custom communication example discussed in section 4.3.3.2 shows how the macro is used to invoke a custom global sum routine for a specific set of hardware.

7. **_EXCH** The _EXCH CPP macro is used to update tile overlap regions. It is qualified by a suffix indicating whether overlap updates are for two-dimensional (_EXCH_XY) or three dimensional (_EXCH_XYZ) physical fields and whether fields are 32-bit floating point (_EXCH_XY_R4, _EXCH_XYZ_R4) or 64-bit floating point (_EXCH_XY_R8, _EXCH_XYZ_R8). The macro mappings are defined in the header file *CPP_EEMACROS.h*. As with _GSUM, the _EXCH operation plays a crucial role in scaling to small tile, large logical and physical processor count configurations. The example in section 4.3.3.2 discusses defining an optimized and specialized form on the _EXCH operation.

   The _EXCH operation is also central to supporting grids such as the cube-sphere grid. In this class of grid a rotation may be required between tiles. Aligning the coordinate requiring rotation with the tile decomposition, allows the coordinate transformation to be embedded within a custom form of the _EXCH primitive. In these cases _EXCH is mapped to exch2 routines, as detailed in the exch2 package documentation 6.18.

8. **Reverse Mode** The communication primitives _EXCH and _GSUM both employ hand-written adjoint forms (or reverse mode) forms. These reverse mode forms can be found in the source code directory *pkg/autodiff*. For the global sum primitive the reverse mode form calls are to *GLOBAL_ADSUM_R4* and *GLOBAL_ADSUM_R8*. The reverse mode form of the exchange primitives are found in routines prefixed *ADEXCH*. The exchange routines make calls to the same low-level communication primitives as the forward mode operations. However, the routine argument *simulationMode* is set to the value *REVERSE_SIMULATION*. This signifies ti the low-level routines that the adjoint forms of the appropriate communication operation should be performed.

9. **MAX_NO_THREADS** The variable *MAX_NO_THREADS* is used to indicate the maximum number of OS threads that a code will use. This value defaults to thirty-two and is set in the file *EEPARAMS.h*. For single threaded execution it can be reduced to one if required. The value; is largely private to the WRAPPER and application code will nor normally reference the value, except in the following scenario.

   For certain physical parametrization schemes it is necessary to have a substantial number of work arrays. Where these arrays are allocated in heap storage (for example COMMON blocks) multi-threaded execution will require multiple instances of the COMMON block data. This can be achieved using a Fortran 90 module construct, however, if this might be unavailable then the work arrays can be extended with dimensions use the tile dimensioning scheme of *nSx* and *nSy* (as described in section 4.3.1). However, if the configuration being specified involves many more tiles than OS threads then it can save memory resources to reduce the variable *MAX_NO_THREADS* to be equal to the actual number of threads that will be used and to declare the physical parameterization work arrays with a single *MAX_NO_THREADS* extra dimension. An example of this is given in the verification experiment *aim.5l_cs*. Here the default setting of *MAX_NO_THREADS* is altered to

```
INTEGER MAX_NO_THREADS
PARAMETER ( MAX_NO_THREADS =    6 )
```

   and several work arrays for storing intermediate calculations are created with declarations of the form.

```
common /FORCIN/ sst1(ngp,MAX_NO_THREADS)
```

   This declaration scheme is not used widely, because most global data is used for permanent not temporary storage of state information. In the case of permanent state information this approach cannot be used because there has to be enough storage allocated for all tiles. However, the technique can sometimes be a useful scheme for reducing memory requirements in complex physical parameterizations.

```
C--
C--  Parallel directives for MIPS Pro Fortran compiler
C--
C      Parallel compiler directives for SGI with IRIX
C$PAR  PARALLEL DO
C$PAR&  CHUNK=1,MP_SCHEDTYPE=INTERLEAVE,
C$PAR&  SHARE(nThreads),LOCAL(myThid,I)
C
      DO I=1,nThreads
        myThid = I

C--     Invoke nThreads instances of the numerical model
        CALL THE_MODEL_MAIN(myThid)

      ENDDO
```

Figure 4.12: Prior to transferring control to the procedure *THE_MODEL_MAIN()* the WRAPPER may use MP directives to spawn multiple threads.

4.3.3.1 Specializing the Communication Code

The isolation of performance critical communication primitives and the subdivision of the simulation domain into tiles is a powerful tool. Here we show how it can be used to improve application performance and how it can be used to adapt to new griding approaches.

4.3.3.2 JAM example

On some platforms a big performance boost can be obtained by binding the communication routines *_EXCH* and *_GSUM* to specialized native libraries) fro example the shmem library on CRAY T3E systems). The *LETS_MAKE_JAM* CPP flag is used as an illustration of a specialized communication configuration that substitutes for standard, portable forms of *_EXCH* and *_GSUM*. It affects three source files *eeboot.F*, *CPP_EEMACROS.h* and *cg2d.F*. When the flag is defined is has the following effects.

- An extra phase is included at boot time to initialize the custom communications library (see *ini_jam.F*).
- The *_GSUM* and *_EXCH* macro definitions are replaced with calls to custom routines (see *gsum_jam.F* and *exch_jam.F*)
- a highly specialized form of the exchange operator (optimized for overlap regions of width one) is substituted into the elliptic solver routine *cg2d.F*.

Developing specialized code for other libraries follows a similar pattern.

4.3.3.3 Cube sphere communication

Actual *_EXCH* routine code is generated automatically from a series of template files, for example *exch_rx.template*. This is done to allow a large number of variations on the exchange process to be maintained. One set of variations supports the cube sphere grid. Support for a cube sphere grid in MITgcm is based on having each face of the cube as a separate tile or tiles. The exchange routines are then able to absorb much of the detailed rotation and reorientation required when moving around the cube grid. The set of *_EXCH* routines that contain the word cube in their name perform these transformations. They are invoked when the run-time logical parameter *useCubedSphereExchange* is set true. To facilitate the transformations on a staggered C-grid, exchange operations are defined separately for both vector and scalar quantities and for grid-centered and for grid-face and corner quantities. Three sets of exchange routines are defined. Routines with names of the form *exch_rx* are used to exchange cell centered scalar quantities. Routines with names of the form *exch_uv_rx* are used to exchange vector quantities located at the C-grid velocity points. The vector quantities exchanged by the *exch_uv_rx* routines can either be signed (for example velocity components) or un-signed (for example grid-cell separations). Routines with names of the form *exch_z_rx* are used to exchange quantities at the C-grid vorticity point locations.

4.4 MITgcm execution under WRAPPER

Fitting together the WRAPPER elements, package elements and MITgcm core equation elements of the source code produces calling sequence shown in section 4.4.1

4.4.1 Annotated call tree for MITgcm and WRAPPER

WRAPPER layer.

```
MAIN
|
|--EEBOOT               :: WRAPPER initialization
|  |
|  |-- EEBOOT_MINMAL    :: Minimal startup. Just enough to
|  |                       allow basic I/O.
|  |-- EEINTRO_MSG      :: Write startup greeting.
|  |
|  |-- EESET_PARMS      :: Set WRAPPER parameters
|  |
|  |-- EEWRITE_EEENV    :: Print WRAPPER parameter settings
|  |
|  |-- INI_PROCS        :: Associate processes with grid regions.
|  |
|  |-- INI_THREADING_ENVIRONMENT   :: Associate threads with grid regions.
|       |
|       |--INI_COMMUNICATION_PATTERNS :: Initialize between tile
```

```
       |                                  :: communication data structures
       |
       |
       |--CHECK_THREADS    :: Validate multiple thread start up.
       |
       |--THE_MODEL_MAIN   :: Numerical code top-level driver routine
```

Core equations plus packages.

```
C
C
C Invocation from WRAPPER level...
C  :
C  :
C  |
C  |-THE_MODEL_MAIN :: Primary driver for the MITgcm algorithm
C    |              :: Called from WRAPPER level numerical
C    |              :: code invocation routine. On entry
C    |              :: to THE_MODEL_MAIN separate thread and
C    |              :: separate processes will have been established.
C    |              :: Each thread and process will have a unique ID
C    |              :: but as yet it will not be associated with a
C    |              :: specific region in decomposed discrete space.
C    |
C    |-INITIALISE_FIXED :: Set fixed model arrays such as topography,
C    | |                :: grid, solver matrices etc..
C    | |
C    | |-INI_PARMS :: Routine to set kernel model parameters.
C    | |           :: By default kernel parameters are read from file
C    | |           :: "data" in directory in which code executes.
C    | |
C    | |-MON_INIT :: Initializes monitor package ( see pkg/monitor )
C    | |
C    | |-INI_GRID :: Control grid array (vert. and hori.) initialization.
C    | | |        :: Grid arrays are held and described in GRID.h.
C    | | |
C    | | |-INI_VERTICAL_GRID        :: Initialize vertical grid arrays.
C    | | |
C    | | |-INI_CARTESIAN_GRID       :: Cartesian horiz. grid initialization
C    | | |                          :: (calculate grid from kernel parameters).
C    | | |
C    | | |-INI_SPHERICAL_POLAR_GRID :: Spherical polar horiz. grid
C    | | |                          :: initialization (calculate grid from
C    | | |                          :: kernel parameters).
C    | | |
C    | | |-INI_CURVILINEAR_GRID     :: General orthogonal, structured horiz.
C    | | |                          :: grid initializations. ( input from raw
C    | | |                          :: grid files, LONC.bin, DXF.bin etc... )
C    | | |
C    | |-INI_DEPTHS    :: Read (from "bathyFile") or set bathymetry/orgography.
C    | |
C    | |-INI_MASKS_ETC :: Derive horizontal and vertical cell fractions and
C    | |               :: land masking for solid-fluid boundaries.
C    | |
C    | |-INI_LINEAR_PHSURF :: Set ref. surface Bo_surf
C    | |
C    | |-INI_CORI          :: Set coriolis term. zero, f-plane, beta-plane,
```

```
C    | |                  :: sphere options are coded.
C    | |
C    | |-PACAKGES_BOOT      :: Start up the optional package environment.
C    | |                    :: Runtime selection of active packages.
C    | |
C    | |-PACKAGES_READPARMS :: Call active package internal parameter load.
C    | | |
C    | | |-GMREDI_READPARMS    :: GM Package. see pkg/gmredi
C    | | |-KPP_READPARMS       :: KPP Package. see pkg/kpp
C    | | |-SHAP_FILT_READPARMS :: Shapiro filter package. see pkg/shap_filt
C    | | |-OBCS_READPARMS      :: Open bndy package. see pkg/obcs
C    | | |-AIM_READPARMS       :: Intermediate Atmos. pacakage. see pkg/aim
C    | | |-COST_READPARMS      :: Cost function package. see pkg/cost
C    | | |-CTRL_INIT           :: Control vector support package. see pkg/ctrl
C    | | |-OPTIM_READPARMS     :: Optimisation support package. see pkg/ctrl
C    | | |-GRDCHK_READPARMS    :: Gradient check package. see pkg/grdchk
C    | | |-ECCO_READPARMS      :: ECCO Support Package. see pkg/ecco
C    | |
C    | |-PACKAGES_CHECK
C    | | |
C    | | |-KPP_CHECK           :: KPP Package. pkg/kpp
C    | | |-OBCS_CHECK          :: Open bndy Package. pkg/obcs
C    | | |-GMREDI_CHECK        :: GM Package. pkg/gmredi
C    | |
C    | |-PACKAGES_INIT_FIXED
C    | | |-OBCS_INIT_FIXED     :: Open bndy Package. see pkg/obcs
C    | | |-FLT_INIT            :: Floats Package. see pkg/flt
C    | |
C    | |-ZONAL_FILT_INIT       :: FFT filter Package. see pkg/zonal_filt
C    | |
C    | |-INI_CG2D              :: 2d con. grad solver initialisation.
C    | |
C    | |-INI_CG3D              :: 3d con. grad solver initialisation.
C    | |
C    | |-CONFIG_SUMMARY        :: Provide synopsis of kernel setup.
C    |                         :: Includes annotated table of kernel
C    |                         :: parameter settings.
C    |
C    |-CTRL_UNPACK :: Control vector support package. see pkg/ctrl
C    |
C    |-ADTHE_MAIN_LOOP :: Derivative evaluating form of main time stepping loop
C    !                 :: Auotmatically generated by TAMC/TAF.
C    |
C    |-CTRL_PACK   :: Control vector support package. see pkg/ctrl
C    |
C    |-GRDCHK_MAIN :: Gradient check package. see pkg/grdchk
C    |
C    |-THE_MAIN_LOOP :: Main timestepping loop routine.
C    | |
C    | |-INITIALISE_VARIA :: Set the initial conditions for time evolving
C    | | |                :: variables
C    | | |
C    | | |-INI_LINEAR_PHISURF :: Set ref. surface Bo_surf
C    | | |
C    | | |-INI_CORI     :: Set coriolis term. zero, f-plane, beta-plane,
C    | | |              :: sphere options are coded.
C    | | |
```

```
C    | | |-INI_CG2D     :: 2d con. grad solver initialisation.
C    | | |-INI_CG3D     :: 3d con. grad solver initialisation.
C    | | |-INI_MIXING   :: Initialise diapycnal diffusivity.
C    | | |-INI_DYNVARS  :: Initialise to zero all DYNVARS.h arrays (dynamical
C    | | |              :: fields).
C    | | |
C    | | |-INI_FIELDS   :: Control initializing model fields to non-zero
C    | | | |-INI_VEL    :: Initialize 3D flow field.
C    | | | |-INI_THETA  :: Set model initial temperature field.
C    | | | |-INI_SALT   :: Set model initial salinity field.
C    | | | |-INI_PSURF  :: Set model initial free-surface height/pressure.
C    | | |
C    | | |-INI_TR1      :: Set initial tracer 1 distribution.
C    | | |
C    | | |-THE_CORRECTION_STEP :: Step forward to next time step.
C    | | | |                   :: Here applied to move restart conditions
C    | | | |                   :: (saved in mid timestep) to correct level in
C    | | | |                   :: time (only used for pre-c35).
C    | | | |
C    | | | |-CALC_GRAD_PHI_SURF :: Return DDx and DDy of surface pressure
C    | | | |-CORRECTION_STEP    :: Pressure correction to momentum
C    | | | |-CYCLE_TRACER       :: Move tracers forward in time.
C    | | | |-OBCS_APPLY         :: Open bndy package. see pkg/obcs
C    | | | |-SHAP_FILT_APPLY    :: Shapiro filter package. see pkg/shap_filt
C    | | | |-ZONAL_FILT_APPLY   :: FFT filter package. see pkg/zonal_filt
C    | | | |-CONVECTIVE_ADJUSTMENT :: Control static instability mixing.
C    | | | | |-FIND_RHO  :: Find adjacent densities.
C    | | | | |-CONVECT   :: Mix static instability.
C    | | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C    | | | |
C    | | | |-CALC_EXACT_ETA        :: Change SSH to flow divergence.
C    | | |
C    | | |-CONVECTIVE_ADJUSTMENT_INI :: Control static instability mixing
C    | | | |                         :: Extra time history interactions.
C    | | | |
C    | | | |-FIND_RHO  :: Find adjacent densities.
C    | | | |-CONVECT   :: Mix static instability.
C    | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C    | | |
C    | | |-PACKAGES_INIT_VARIABLES :: Does initialisation of time evolving
C    | | | |                       :: package data.
C    | | | |
C    | | | |-GMREDI_INIT          :: GM package. ( see pkg/gmredi )
C    | | | |-KPP_INIT             :: KPP package. ( see pkg/kpp )
C    | | | |-KPP_OPEN_DIAGS
C    | | | |-OBCS_INIT_VARIABLES  :: Open bndy. package. ( see pkg/obcs )
C    | | | |-AIM_INIT             :: Interm. atmos package. ( see pkg/aim )
C    | | | |-CTRL_MAP_INI         :: Control vector package.( see pkg/ctrl )
C    | | | |-COST_INIT            :: Cost function package. ( see pkg/cost )
C    | | | |-ECCO_INIT            :: ECCO support package. ( see pkg/ecco )
C    | | | |-INI_FORCING          :: Set model initial forcing fields.
C    | | | | |                    :: Either set in-line or from file as shown.
C    | | | |  |-READ_FLD_XY_RS(zonalWindFile)
C    | | | |  |-READ_FLD_XY_RS(meridWindFile)
C    | | | |  |-READ_FLD_XY_RS(surfQFile)
C    | | | |  |-READ_FLD_XY_RS(EmPmRfile)
C    | | | |  |-READ_FLD_XY_RS(thetaClimFile)
```

```
C    | | |   |-READ_FLD_XY_RS(saltClimFile)
C    | | |   |-READ_FLD_XY_RS(surfQswFile)
C    | | |
C    | | |-CALC_SURF_DR   :: Calculate the new surface level thickness.
C    | | |-UPDATE_SURF_DR :: Update the surface-level thickness fraction.
C    | | |-UPDATE_CG2D    :: Update 2d conjugate grad. for Free-Surf.
C    | | |-STATE_SUMMARY    :: Summarize model prognostic variables.
C    | | |-TIMEAVE_STATVARS :: Time averaging package ( see pkg/timeave ).
C    | |
C    | |-WRITE_STATE      :: Controlling routine for IO to dump model state.
C    | | |-WRITE_REC_XYZ_RL :: Single file I/O
C    | | |-WRITE_FLD_XYZ_RL :: Multi-file I/O
C    | |
C    | |-MONITOR          :: Monitor state ( see pkg/monitor )
C    | |-CTRL_MAP_FORCING :: Control vector support package. ( see pkg/ctrl )
C====|>|
C====|>| ***************************
C====|>| BEGIN MAIN TIMESTEPPING LOOP
C====|>| ***************************
C/\  | |-FORWARD_STEP     :: Step forward a time-step ( AT LAST !!! )
C/\  | | |
C/\  | | |-DUMMY_IN_STEPPING :: autodiff package ( pkg/autoduff ).
C/\  | | |-CALC_EXACT_ETA :: Change SSH to flow divergence.
C/\  | | |-CALC_SURF_DR   :: Calculate the new surface level thickness.
C/\  | | |-EXF_GETFORCING :: External forcing package. ( pkg/exf )
C/\  | | |-EXTERNAL_FIELDS_LOAD :: Control loading time dep. external data.
C/\  | | | |                    :: Simple interpolation between end-points
C/\  | | | |                    :: for forcing datasets.
C/\  | | | |
C/\  | | | |-EXCH :: Sync forcing. in overlap regions.
C/\  | | |
C/\  | | |-THERMODYNAMICS :: theta, salt + tracer equations driver.
C/\  | | | |
C/\  | | | |-INTEGRATE_FOR_W :: Integrate for vertical velocity.
C/\  | | | |-OBCS_APPLY_W    :: Open bndy. package ( see pkg/obcs ).
C/\  | | | |-FIND_RHO        :: Calculates [rho(S,T,z)-Rhonil] of a slice
C/\  | | | |-GRAD_SIGMA      :: Calculate isoneutral gradients
C/\  | | | |-CALC_IVDC       :: Set Implicit Vertical Diffusivity for Convection
C/\  | | | |
C/\  | | | |-OBCS_CALC            :: Open bndy. package ( see pkg/obcs ).
C/\  | | | |-EXTERNAL_FORCING_SURF:: Accumulates appropriately dimensioned
C/\  | | | |                      :: forcing terms.
C/\  | | | |
C/\  | | | |-GMREDI_CALC_TENSOR   :: GM package ( see pkg/gmredi ).
C/\  | | | |-GMREDI_CALC_TENSOR_DUMMY :: GM package ( see pkg/gmredi ).
C/\  | | | |-KPP_CALC             :: KPP package ( see pkg/kpp ).
C/\  | | | |-KPP_CALC_DUMMY       :: KPP package ( see pkg/kpp ).
C/\  | | | |-AIM_DO_ATMOS_PHYSICS :: Intermed. atmos package ( see pkg/aim ).
C/\  | | | |-GAD_ADVECTION        :: Generalised advection driver (multi-dim
C/\  | | | |                         advection case) (see pkg/gad).
C/\  | | | |-CALC_COMMON_FACTORS  :: Calculate common data (such as volume flux)
C/\  | | | |-CALC_DIFFUSIVITY     :: Calculate net vertical diffusivity
C/\  | | | | |
C/\  | | | | |-GMREDI_CALC_DIFF   :: GM package ( see pkg/gmredi ).
C/\  | | | | |-KPP_CALC_DIFF      :: KPP package ( see pkg/kpp ).
C/\  | | | |
```

```
C/\  | | | |-CALC_GT              :: Calculate the temperature tendency terms
C/\  | | | | |
C/\  | | | | |-GAD_CALC_RHS       :: Generalised advection package
C/\  | | | | |                    :: ( see pkg/gad )
C/\  | | | | |-EXTERNAL_FORCING_T :: Problem specific forcing for temperature.
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tendencies forward in time.
C/\  | | | | |-FREESURF_RESCALE_G :: Re-scale Gt for free-surface height.
C/\  | | | |
C/\  | | | |-TIMESTEP_TRACER      :: Step tracer field forward in time
C/\  | | | |
C/\  | | | |-CALC_GS              :: Calculate the salinity tendency terms
C/\  | | | | |
C/\  | | | | |-GAD_CALC_RHS       :: Generalised advection package
C/\  | | | | |                    :: ( see pkg/gad )
C/\  | | | | |-EXTERNAL_FORCING_S :: Problem specific forcing for salt.
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tendencies forward in time.
C/\  | | | | |-FREESURF_RESCALE_G :: Re-scale Gs for free-surface height.
C/\  | | | |
C/\  | | | |-TIMESTEP_TRACER      :: Step tracer field forward in time
C/\  | | | |
C/\  | | | |-CALC_GTR1            :: Calculate other tracer(s) tendency terms
C/\  | | | | |
C/\  | | | | |-GAD_CALC_RHS       :: Generalised advection package
C/\  | | | | |                    :: ( see pkg/gad )
C/\  | | | | |-EXTERNAL_FORCING_TR:: Problem specific forcing for tracer.
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tendencies forward in time.
C/\  | | | | |-FREESURF_RESCALE_G :: Re-scale Gs for free-surface height.
C/\  | | | |
C/\  | | | |-TIMESTEP_TRACER      :: Step tracer field forward in time
C/\  | | | |-OBCS_APPLY_TS        :: Open bndy. package (see pkg/obcs ).
C/\  | | | |-FREEZE               :: Limit range of temperature.
C/\  | | | |
C/\  | | | |-IMPLDIFF             :: Solve vertical implicit diffusion equation.
C/\  | | | |-OBCS_APPLY_TS        :: Open bndy. package (see pkg/obcs ).
C/\  | | | |
C/\  | | | |-AIM_AIM2DYN_EXCHANGES :: Inetermed. atmos (see pkg/aim).
C/\  | | | |-EXCH                 :: Update overlaps
C/\  | | |
C/\  | | |-DYNAMICS       :: Momentum equations driver.
C/\  | | | |
C/\  | | | |-CALC_GRAD_PHI_SURF :: Calculate the gradient of the surface
C/\  | | | |                       Potential anomaly.
C/\  | | | |-CALC_VISCOSITY     :: Calculate net vertical viscosity
C/\  | | | | |-KPP_CALC_VISC    :: KPP package ( see pkg/kpp ).
C/\  | | | |
C/\  | | | |-CALC_PHI_HYD       :: Integrate the hydrostatic relation.
C/\  | | | |-MOM_FLUXFORM       :: Flux form mom eqn. package ( see
C/\  | | | |                       pkg/mom_fluxform ).
C/\  | | | |-MOM_VECINV         :: Vector invariant form mom eqn. package ( see
C/\  | | | |                       pkg/mom_vecinv   ).
C/\  | | | |-TIMESTEP           :: Step momentum fields forward in time
C/\  | | | |-OBCS_APPLY_UV      :: Open bndy. package (see pkg/obcs ).
C/\  | | | |
C/\  | | | |-IMPLDIFF           :: Solve vertical implicit diffusion equation.
C/\  | | | |-OBCS_APPLY_UV      :: Open bndy. package (see pkg/obcs ).
C/\  | | | |
C/\  | | | |-TIMEAVE_CUMUL_1T   :: Time averaging package ( see pkg/timeave ).
```

```
C/\  | | | |-TIMEAVE_CUMUATE    :: Time averaging package ( see pkg/timeave ).
C/\  | | | |-DEBUG_STATS_RL     :: Quick debug package ( see pkg/debug ).
C/\  | | |
C/\  | | |-CALC_GW        :: vert. momentum tendency terms ( NH, QH only ).
C/\  | | |
C/\  | | |-UPDATE_SURF_DR :: Update the surface-level thickness fraction.
C/\  | | |
C/\  | | |-UPDATE_CG2D    :: Update 2d conjugate grad. for Free-Surf.
C/\  | | |
C/\  | | |-SOLVE_FOR_PRESSURE           :: Find surface pressure.
C/\  | | | |-CALC_DIV_GHAT     :: Form the RHS of the surface pressure eqn.
C/\  | | | |-CG2D              :: Two-dim pre-con. conjugate-gradient.
C/\  | | | |-CG3D              :: Three-dim pre-con. conjugate-gradient solver.
C/\  | | |
C/\  | | |-THE_CORRECTION_STEP          :: Step forward to next time step.
C/\  | | | |
C/\  | | | |-CALC_GRAD_PHI_SURF :: Return DDx and DDy of surface pressure
C/\  | | | |-CORRECTION_STEP    :: Pressure correction to momentum
C/\  | | | |-CYCLE_TRACER       :: Move tracers forward in time.
C/\  | | | |-OBCS_APPLY         :: Open bndy package. see pkg/obcs
C/\  | | | |-SHAP_FILT_APPLY    :: Shapiro filter package. see pkg/shap_filt
C/\  | | | |-ZONAL_FILT_APPLY   :: FFT filter package. see pkg/zonal_filt
C/\  | | | |-CONVECTIVE_ADJUSTMENT :: Control static instability mixing.
C/\  | | | | |-FIND_RHO  :: Find adjacent densities.
C/\  | | | | |-CONVECT   :: Mix static instability.
C/\  | | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C/\  | | | |
C/\  | | | |-CALC_EXACT_ETA        :: Change SSH to flow divergence.
C/\  | | |
C/\  | | |-DO_FIELDS_BLOCKING_EXCHANGES :: Sync up overlap regions.
C/\  | | | |-EXCH
C/\  | | |
C/\  | | |-FLT_MAIN         :: Float package ( pkg/flt ).
C/\  | | |
C/\  | | |-MONITOR          :: Monitor package ( pkg/monitor ).
C/\  | | |
C/\  | | |-DO_THE_MODEL_IO  :: Standard diagnostic I/O.
C/\  | | | |-WRITE_STATE    :: Core state I/O
C/\  | | | |-TIMEAVE_STATV_WRITE :: Time averages. see pkg/timeave
C/\  | | | |-AIM_WRITE_DIAGS     :: Intermed. atmos diags. see pkg/aim
C/\  | | | |-GMREDI_DIAGS        :: GM diags. see pkg/gmredi
C/\  | | | |-KPP_DO_DIAGS        :: KPP diags. see pkg/kpp
C/\  | | |
C/\  | | |-WRITE_CHECKPOINT :: Do I/O for restart files.
C/\  | |
C/\  | |-COST_TILE        :: Cost function package. ( see pkg/cost )
C<===|=|
C<===|=| *************************
C<===|=| END MAIN TIMESTEPPING LOOP
C<===|=| *************************
C<===|=|
C    | |-COST_FINAL       :: Cost function package. ( see pkg/cost )
C    |
C    |-WRITE_CHECKPOINT :: Final state storage, for restart.
C    |
C    |-TIMER_PRINTALL :: Computational timing summary
C    |
```

```
C    |-COMM_STATS     :: Summarise inter-proc and inter-thread communication
C                     :: events.
C
```

4.4.2 Measuring and Characterizing Performance

TO BE DONE (CNH)

4.4.3 Estimating Resource Requirements

TO BE DONE (CNH)

4.4.3.1 Atlantic 1/6 degree example

4.4.3.2 Dry Run testing

4.4.3.3 Adjoint Resource Requirements

4.4.3.4 State Estimation Environment Resources

Chapter 5

Automatic Differentiation

Automatic differentiation (AD), also referred to as algorithmic (or, more loosely, computational) differentiation, involves automatically deriving code to calculate partial derivatives from an existing fully non-linear prognostic code. (see [23]). A software tool is used that parses and transforms source files according to a set of linguistic and mathematical rules. AD tools are like source-to-source translators in that they parse a program code as input and produce a new program code as output. However, unlike a pure source-to-source translation, the output program represents a new algorithm, such as the evaluation of the Jacobian, the Hessian, or higher derivative operators. In principle, a variety of derived algorithms can be generated automatically in this way.

The MITGCM has been adapted for use with the Tangent linear and Adjoint Model Compiler (TAMC) and its successor TAF (Transformation of Algorithms in Fortran), developed by Ralf Giering ([20], [18, 19]). The first application of the adjoint of the MITGCM for sensitivity studies has been published by [37]. [48, 47] use the MITGCM and its adjoint for ocean state estimation studies. In the following we shall refer to TAMC and TAF synonymously, except were explicitly stated otherwise.

TAMC exploits the chain rule for computing the first derivative of a function with respect to a set of input variables. Treating a given forward code as a composition of operations – each line representing a compositional element, the chain rule is rigorously applied to the code, line by line. The resulting tangent linear or adjoint code, then, may be thought of as the composition in forward or reverse order, respectively, of the Jacobian matrices of the forward code's compositional elements.

5.1 Some basic algebra

Let $\mathcal{M}$ be a general nonlinear, model, i.e. a mapping from the m-dimensional space $U \subset \mathbb{R}^m$ of input variables $\vec{u} = (u_1, \ldots, u_m)$ (model parameters, initial conditions, boundary conditions such as forcing functions) to the n-dimensional

space $V \subset \mathbb{R}^n$ of model output variable $\vec{v} = (v_1, \ldots, v_n)$ (model state, model diagnostics, objective function, ...) under consideration,

$$\begin{aligned} \mathcal{M} : U &\longrightarrow V \\ \vec{u} &\longmapsto \vec{v} = \mathcal{M}(\vec{u}) \end{aligned} \tag{5.1}$$

The vectors $\vec{u} \in U$ and $v \in V$ may be represented w.r.t. some given basis vectors $\text{span}(U) = \{\vec{e}_i\}_{i=1,\ldots,m}$ and $\text{span}(V) = \{\vec{f}_j\}_{j=1,\ldots,n}$ as

$$\vec{u} = \sum_{i=1}^{m} u_i \, \vec{e}_i, \qquad \vec{v} = \sum_{j=1}^{n} v_j \, \vec{f}_j$$

Two routes may be followed to determine the sensitivity of the output variable $\vec{v}$ to its input $\vec{u}$.

5.1.1 Forward or direct sensitivity

Consider a perturbation to the input variables $\delta\vec{u}$ (typically a single component $\delta\vec{u} = \delta u_i \, \vec{e}_i$). Their effect on the output may be obtained via the linear approximation of the model $\mathcal{M}$ in terms of its Jacobian matrix M, evaluated in the point $u^{(0)}$ according to

$$\delta\vec{v} \; = \; M|_{\vec{u}^{(0)}} \, \delta\vec{u} \tag{5.2}$$

with resulting output perturbation $\delta\vec{v}$. In components $M_{ji} = \partial\mathcal{M}_j/\partial u_i$, it reads

$$\delta v_j \; = \; \sum_i \left. \frac{\partial \mathcal{M}_j}{\partial u_i} \right|_{u^{(0)}} \delta u_i \tag{5.3}$$

Eq. (5.2) is the tangent linear model (TLM). In contrast to the full nonlinear model $\mathcal{M}$, the operator M is just a matrix which can readily be used to find the forward sensitivity of $\vec{v}$ to perturbations in u, but if there are very many input variables ($\gg O(10^6)$ for large-scale oceanographic application), it quickly becomes prohibitive to proceed directly as in (5.2), if the impact of each component $\mathbf{e_i}$ is to be assessed.

5.1.2 Reverse or adjoint sensitivity

Let us consider the special case of a scalar objective function $\mathcal{J}(\vec{v})$ of the model output (e.g. the total meridional heat transport, the total uptake of CO_2 in the Southern Ocean over a time interval, or a measure of some model-to-data misfit)

$$\begin{array}{rccccc} \mathcal{J} : & U & \longrightarrow & V & \longrightarrow & \mathbb{R} \\ & \vec{u} & \longmapsto & \vec{v} = \mathcal{M}(\vec{u}) & \longmapsto & \mathcal{J}(\vec{u}) = \mathcal{J}(\mathcal{M}(\vec{u})) \end{array} \tag{5.4}$$

The perturbation of $\mathcal{J}$ around a fixed point $\mathcal{J}_0$,

$$\mathcal{J} \; = \; \mathcal{J}_0 \; + \; \delta\mathcal{J}$$

can be expressed in both bases of $\vec{u}$ and $\vec{v}$ w.r.t. their corresponding inner product $\langle\ ,\ \rangle$

$$\begin{aligned}\mathcal{J} &= \mathcal{J}|_{\vec{u}^{(0)}} + \left\langle \nabla_u \mathcal{J}^T|_{\vec{u}^{(0)}}\ ,\ \delta\vec{u} \right\rangle + O(\delta\vec{u}^2) \\ &= \mathcal{J}|_{\vec{v}^{(0)}} + \left\langle \nabla_v \mathcal{J}^T|_{\vec{v}^{(0)}}\ ,\ \delta\vec{v} \right\rangle + O(\delta\vec{v}^2)\end{aligned} \tag{5.5}$$

(note, that the gradient ∇f is a co-vector, therefore its transpose is required in the above inner product). Then, using the representation of $\delta\mathcal{J} = \left\langle \nabla_v \mathcal{J}^T\ ,\ \delta\vec{v} \right\rangle$, the definition of an adjoint operator A^* of a given operator A,

$$\langle\, A^*\vec{x}\,,\, \vec{y}\,\rangle = \langle\, \vec{x}\,,\, A\vec{y}\,\rangle$$

which for finite-dimensional vector spaces is just the transpose of A,

$$A^* \;=\; A^T$$

and from eq. (5.2), (5.5), we note that (omitting |'s):

$$\delta\mathcal{J} \;=\; \left\langle \nabla_v \mathcal{J}^T\ ,\ \delta\vec{v} \right\rangle \;=\; \left\langle \nabla_v \mathcal{J}^T\ ,\ M\,\delta\vec{u} \right\rangle \;=\; \left\langle M^T\,\nabla_v \mathcal{J}^T\ ,\ \delta\vec{u} \right\rangle \tag{5.6}$$

With the identity (5.5), we then find that the gradient $\nabla_u\mathcal{J}$ can be readily inferred by invoking the adjoint M^* of the tangent linear model M

$$\begin{aligned}\nabla_u \mathcal{J}^T|_{\vec{u}} &= M^T|_{\vec{u}} \cdot \nabla_v \mathcal{J}^T|_{\vec{v}} \\ &= M^T|_{\vec{u}} \cdot \delta\vec{v}^* \\ &= \delta\vec{u}^*\end{aligned} \tag{5.7}$$

Eq. (5.7) is the adjoint model (ADM), in which M^T is the adjoint (here, the transpose) of the tangent linear operator M, $\delta\vec{v}^*$ the adjoint variable of the model state $\vec{v}$, and $\delta\vec{u}^*$ the adjoint variable of the control variable $\vec{u}$.

The reverse nature of the adjoint calculation can be readily seen as follows. Consider a model integration which consists of Λ consecutive operations $\mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(.....(\mathcal{M}_\lambda(.....(\mathcal{M}_1(\mathcal{M}_0(\vec{u})))))$, where the $\mathcal{M}$'s could be the elementary steps, i.e. single lines in the code of the model, or successive time steps of the model integration, starting at step 0 and moving up to step Λ, with intermediate $\mathcal{M}_\lambda(\vec{u}) = \vec{v}^{(\lambda+1)}$ and final $\mathcal{M}_\Lambda(\vec{u}) = \vec{v}^{(\Lambda+1)} = \vec{v}$. Let $\mathcal{J}$ be a cost function which explicitly depends on the final state $\vec{v}$ only (this restriction is for clarity reasons only). $\mathcal{J}(u)$ may be decomposed according to:

$$\mathcal{J}(\mathcal{M}(\vec{u})) \;=\; \mathcal{J}(\mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(.....(\mathcal{M}_\lambda(.....(\mathcal{M}_1(\mathcal{M}_0(\vec{u}))))))) \tag{5.8}$$

Then, according to the chain rule, the forward calculation reads, in terms of the Jacobi matrices (we've omitted the |'s which, nevertheless are important to the aspect of *tangent* linearity; note also that by definition $\langle \nabla_v \mathcal{J}^T\ ,\ \delta\vec{v} \rangle = \nabla_v\mathcal{J}\cdot\delta\vec{v}$)

$$\begin{aligned}\nabla_v \mathcal{J}(M(\delta\vec{u})) &= \nabla_v\mathcal{J}\cdot M_\Lambda \cdot ..... \cdot M_\lambda \cdot ..... \cdot M_1 \cdot M_0 \cdot \delta\vec{u} \\ &= \nabla_v\mathcal{J}\cdot\delta\vec{v}\end{aligned} \tag{5.9}$$

whereas in reverse mode we have

$$\boxed{\begin{aligned} M^T(\nabla_v \mathcal{J}^T) &= M_0^T \cdot M_1^T \cdot ..... \cdot M_\lambda^T \cdot ..... \cdot M_\Lambda^T \cdot \nabla_v \mathcal{J}^T \\ &= M_0^T \cdot M_1^T \cdot ..... \cdot \nabla_{v^{(\lambda)}} \mathcal{J}^T \\ &= \nabla_u \mathcal{J}^T \end{aligned}} \tag{5.10}$$

clearly expressing the reverse nature of the calculation. Eq. (5.10) is at the heart of automatic adjoint compilers. If the intermediate steps λ in eqn. (5.8) – (5.10) represent the model state (forward or adjoint) at each intermediate time step as noted above, then correspondingly, $M^T(\delta\vec{v}^{(\lambda)\,*}) = \delta\vec{v}^{(\lambda-1)\,*}$ for the adjoint variables. It thus becomes evident that the adjoint calculation also yields the adjoint of each model state component $\vec{v}^{(\lambda)}$ at each intermediate step λ, namely

$$\boxed{\begin{aligned} \nabla_{v^{(\lambda)}} \mathcal{J}^T|_{\vec{v}^{(\lambda)}} &= M_\lambda^T|_{\vec{v}^{(\lambda)}} \cdot ..... \cdot M_\Lambda^T|_{\vec{v}^{(\lambda)}} \cdot \delta\vec{v}^* \\ &= \delta\vec{v}^{(\lambda)\,*} \end{aligned}} \tag{5.11}$$

in close analogy to eq. (5.7) We note in passing that that the $\delta\vec{v}^{(\lambda)\,*}$ are the Lagrange multipliers of the model equations which determine $\vec{v}^{(\lambda)}$.

In components, eq. (5.7) reads as follows. Let

$$\begin{aligned} \delta\vec{u} &= (\delta u_1, \ldots, \delta u_m)^T, & \delta\vec{u}^* &= \nabla_u \mathcal{J}^T = \left(\tfrac{\partial \mathcal{J}}{\partial u_1}, \ldots, \tfrac{\partial \mathcal{J}}{\partial u_m}\right)^T \\ \delta\vec{v} &= (\delta v_1, \ldots, \delta u_n)^T, & \delta\vec{v}^* &= \nabla_v \mathcal{J}^T = \left(\tfrac{\partial \mathcal{J}}{\partial v_1}, \ldots, \tfrac{\partial \mathcal{J}}{\partial v_n}\right)^T \end{aligned}$$

denote the perturbations in $\vec{u}$ and $\vec{v}$, respectively, and their adjoint variables; further

$$M = \begin{pmatrix} \frac{\partial \mathcal{M}_1}{\partial u_1} & \cdots & \frac{\partial \mathcal{M}_1}{\partial u_m} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{M}_n}{\partial u_1} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_m} \end{pmatrix}$$

is the Jacobi matrix of $\mathcal{M}$ (an $n \times m$ matrix) such that $\delta\vec{v} = M \cdot \delta\vec{u}$, or

$$\delta v_j = \sum_{i=1}^{m} M_{ji}\, \delta u_i = \sum_{i=1}^{m} \frac{\partial \mathcal{M}_j}{\partial u_i} \delta u_i$$

Then eq. (5.7) takes the form

$$\delta u_i^* = \sum_{j=1}^{n} M_{ji}\, \delta v_j^* = \sum_{j=1}^{n} \frac{\partial \mathcal{M}_j}{\partial u_i} \delta v_j^*$$

or

$$\begin{pmatrix} \frac{\partial}{\partial u_1}\mathcal{J}\Big|_{\vec{u}^{(0)}} \\ \vdots \\ \frac{\partial}{\partial u_m}\mathcal{J}\Big|_{\vec{u}^{(0)}} \end{pmatrix} = \begin{pmatrix} \frac{\partial \mathcal{M}_1}{\partial u_1}\Big|_{\vec{u}^{(0)}} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_1}\Big|_{\vec{u}^{(0)}} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{M}_1}{\partial u_m}\Big|_{\vec{u}^{(0)}} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_m}\Big|_{\vec{u}^{(0)}} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial}{\partial v_1}\mathcal{J}\Big|_{\vec{v}} \\ \vdots \\ \frac{\partial}{\partial v_n}\mathcal{J}\Big|_{\vec{v}} \end{pmatrix}$$

Furthermore, the adjoint $\delta v^{(\lambda)\,*}$ of any intermediate state $v^{(\lambda)}$ may be obtained, using the intermediate Jacobian (an $n_{\lambda+1} \times n_\lambda$ matrix)

$$M_\lambda \;=\; \begin{pmatrix} \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_{n_\lambda}^{(\lambda)}} \\ \vdots & & \vdots \\ \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_{n_\lambda}^{(\lambda)}} \end{pmatrix}$$

and the shorthand notation for the adjoint variables $\delta v_j^{(\lambda)\,*} = \frac{\partial}{\partial v_j^{(\lambda)}} \mathcal{J}^T$, $j = 1, \ldots, n_\lambda$, for intermediate components, yielding

$$\begin{pmatrix} \delta v_1^{(\lambda)\,*} \\ \vdots \\ \delta v_{n_\lambda}^{(\lambda)\,*} \end{pmatrix} = \begin{pmatrix} \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_1^{(\lambda)}} & \cdots\ \cdots & \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_1^{(\lambda)}} \\ \vdots & & \vdots \\ \frac{\partial (\mathcal{M}_\lambda)_1}{\partial v_{n_\lambda}^{(\lambda)}} & \cdots\ \cdots & \frac{\partial (\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_{n_\lambda}^{(\lambda)}} \end{pmatrix} \cdot$$

$$\begin{pmatrix} \frac{\partial (\mathcal{M}_{\lambda+1})_1}{\partial v_1^{(\lambda+1)}} & \cdots & \frac{\partial (\mathcal{M}_{\lambda+1})_{n_{\lambda+2}}}{\partial v_1^{(\lambda+1)}} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \frac{\partial (\mathcal{M}_{\lambda+1})_1}{\partial v_{n_{\lambda+1}}^{(\lambda+1)}} & \cdots & \frac{\partial (\mathcal{M}_{\lambda+1})_{n_{\lambda+2}}}{\partial v_{n_{\lambda+1}}^{(\lambda+1)}} \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} \delta v_1^* \\ \vdots \\ \delta v_n^* \end{pmatrix} \tag{5.12}$$

Eq. (5.9) and (5.10) are perhaps clearest in showing the advantage of the reverse over the forward mode if the gradient $\nabla_u \mathcal{J}$, i.e. the sensitivity of the cost function $\mathcal{J}$ with respect to *all* input variables u (or the sensitivity of the cost function with respect to *all* intermediate states $\vec{v}^{(\lambda)}$) are sought. In order to be able to solve for each component of the gradient $\partial \mathcal{J} / \partial u_i$ in (5.9) a forward calculation has to be performed for each component separately, i.e. $\delta \vec{u} = \delta u_i \vec{e}_i$ for the i-th forward calculation. Then, (5.9) represents the projection of $\nabla_u \mathcal{J}$ onto the i-th component. The full gradient is retrieved from the m forward calculations. In contrast, eq. (5.10) yields the full gradient $\nabla_u \mathcal{J}$ (and all intermediate gradients $\nabla_{v^{(\lambda)}} \mathcal{J}$) within a single reverse calculation.

Note, that if $\mathcal{J}$ is a vector-valued function of dimension $l > 1$, eq. (5.10) has to be modified according to

$$M^T \left(\nabla_v \mathcal{J}^T \left(\delta \vec{J} \right) \right) \;=\; \nabla_u \mathcal{J}^T \cdot \delta \vec{J}$$

where now $\delta \vec{J} \in I\!R^l$ is a vector of dimension l. In this case l reverse simulations have to be performed for each δJ_k, $k = 1, \ldots, l$. Then, the reverse mode is more efficient as long as $l < n$, otherwise the forward mode is preferable. Strictly, the reverse mode is called adjoint mode only for $l = 1$.

A detailed analysis of the underlying numerical operations shows that the computation of $\nabla_u \mathcal{J}$ in this way requires about 2 to 5 times the computation

of the cost function. Alternatively, the gradient vector could be approximated by finite differences, requiring m computations of the perturbed cost function.

To conclude we give two examples of commonly used types of cost functions:

Example 1: $\mathcal{J} = v_j(T)$
The cost function consists of the j-th component of the model state $\vec{v}$ at time T. Then $\nabla_v \mathcal{J}^T = \vec{f_j}$ is just the j-th unit vector. The $\nabla_u \mathcal{J}^T$ is the projection of the adjoint operator onto the j-th component $\mathbf{f_j}$,

$$\nabla_u \mathcal{J}^T \;=\; M^T \cdot \nabla_v \mathcal{J}^T \;=\; \sum_i M^T_{ji}\, \vec{e}_i$$

Example 2: $\mathcal{J} = \langle\, \mathcal{H}(\vec{v}) - \vec{d}\,,\, \mathcal{H}(\vec{v}) - \vec{d} \,\rangle$
The cost function represents the quadratic model vs. data misfit. Here, $\vec{d}$ is the data vector and $\mathcal{H}$ represents the operator which maps the model state space onto the data space. Then, $\nabla_v \mathcal{J}$ takes the form

$$\begin{aligned} \nabla_v \mathcal{J}^T &= 2\; H \cdot \left(\mathcal{H}(\vec{v}) - \vec{d} \right) \\ &= 2 \sum_j \left\{ \sum_k \frac{\partial \mathcal{H}_k}{\partial v_j} \left(\mathcal{H}_k(\vec{v}) - d_k \right) \right\} \vec{f_j} \end{aligned}$$

where $H_{kj} = \partial \mathcal{H}_k / \partial v_j$ is the Jacobi matrix of the data projection operator. Thus, the gradient $\nabla_u \mathcal{J}$ is given by the adjoint operator, driven by the model vs. data misfit:

$$\nabla_u \mathcal{J}^T \;=\; 2\, M^T \cdot H \cdot \left(\mathcal{H}(\vec{v}) - \vec{d} \right)$$

5.1.3 Storing vs. recomputation in reverse mode

We note an important aspect of the forward vs. reverse mode calculation. Because of the local character of the derivative (a derivative is defined w.r.t. a point along the trajectory), the intermediate results of the model trajectory $\vec{v}^{(\lambda+1)} = \mathcal{M}_\lambda(v^{(\lambda)})$ may be required to evaluate the intermediate Jacobian $M_\lambda|_{\vec{v}^{(\lambda)}}\, \delta\vec{v}^{(\lambda)}$. This is the case e.g. for nonlinear expressions (momentum advection, nonlinear equation of state), state-dependent conditional statements (parameterization schemes). In the forward mode, the intermediate results are required in the same order as computed by the full forward model $\mathcal{M}$, but in the reverse mode they are required in the reverse order. Thus, in the reverse mode the trajectory of the forward model integration $\mathcal{M}$ has to be stored to be available in the reverse calculation. Alternatively, the complete model state up to the point of evaluation has to be recomputed whenever its value is required.

A method to balance the amount of recomputations vs. storage requirements is called checkpointing (e.g. [22], [43]). It is depicted in 5.1 for a 3-level checkpointing [as an example, we give explicit numbers for a 3-day integration with a 1-hourly timestep in square brackets].

$lev3$ In a first step, the model trajectory is subdivided into n^{lev3} subsections [n^{lev3}=3 1-day intervals], with the label $lev3$ for this outermost loop. The model is then integrated along the full trajectory, and the model state stored to disk only at every k_i^{lev3}-th timestep [i.e. 3 times, at $i = 0, 1, 2$ corresponding to $k_i^{lev3} = 0, 24, 48$]. In addition, the cost function is computed, if needed.

$lev2$ In a second step each subsection itself is divided into n^{lev2} subsections [n^{lev2}=4 6-hour intervals per subsection]. The model picks up at the last outermost dumped state $v_{k_n^{lev3}}$ and is integrated forward in time along the last subsection, with the label $lev2$ for this intermediate loop. The model state is now stored to disk at every k_i^{lev2}-th timestep [i.e. 4 times, at $i = 0, 1, 2, 3$ corresponding to $k_i^{lev2} = 48, 54, 60, 66$].

$lev1$ Finally, the model picks up at the last intermediate dump state $v_{k_n^{lev2}}$ and is integrated forward in time along the last subsection, with the label $lev1$ for this intermediate loop. Within this sub-subsection only, parts of the model state is stored to memory at every timestep [i.e. every hour $i = 0, ..., 5$ corresponding to $k_i^{lev1} = 66, 67, \ldots, 71$]. The final state $v_n = v_{k_n^{lev1}}$ is reached and the model state of all preceding timesteps along the last innermost subsection are available, enabling integration backwards in time along the last subsection. The adjoint can thus be computed along this last subsection k_n^{lev2}.

This procedure is repeated consecutively for each previous subsection $k_{n-1}^{lev2}, \ldots, k_1^{lev2}$ carrying the adjoint computation to the initial time of the subsection k_n^{lev3}. Then, the procedure is repeated for the previous subsection k_{n-1}^{lev3} carrying the adjoint computation to the initial time k_1^{lev3}.

For the full model trajectory of $n^{lev3} \cdot n^{lev2} \cdot n^{lev1}$ timesteps the required storing of the model state was significantly reduced to $n^{lev2} + n^{lev3}$ to disk and roughly n^{lev1} to memory [i.e. for the 3-day integration with a total oof 72 timesteps the model state was stored 7 times to disk and roughly 6 times to memory]. This saving in memory comes at a cost of a required 3 full forward integrations of the model (one for each checkpointing level). The optimal balance of storage vs. recomputation certainly depends on the computing resources available and may be adjusted by adjusting the partitioning among the n^{lev3}, n^{lev2}, n^{lev1}.

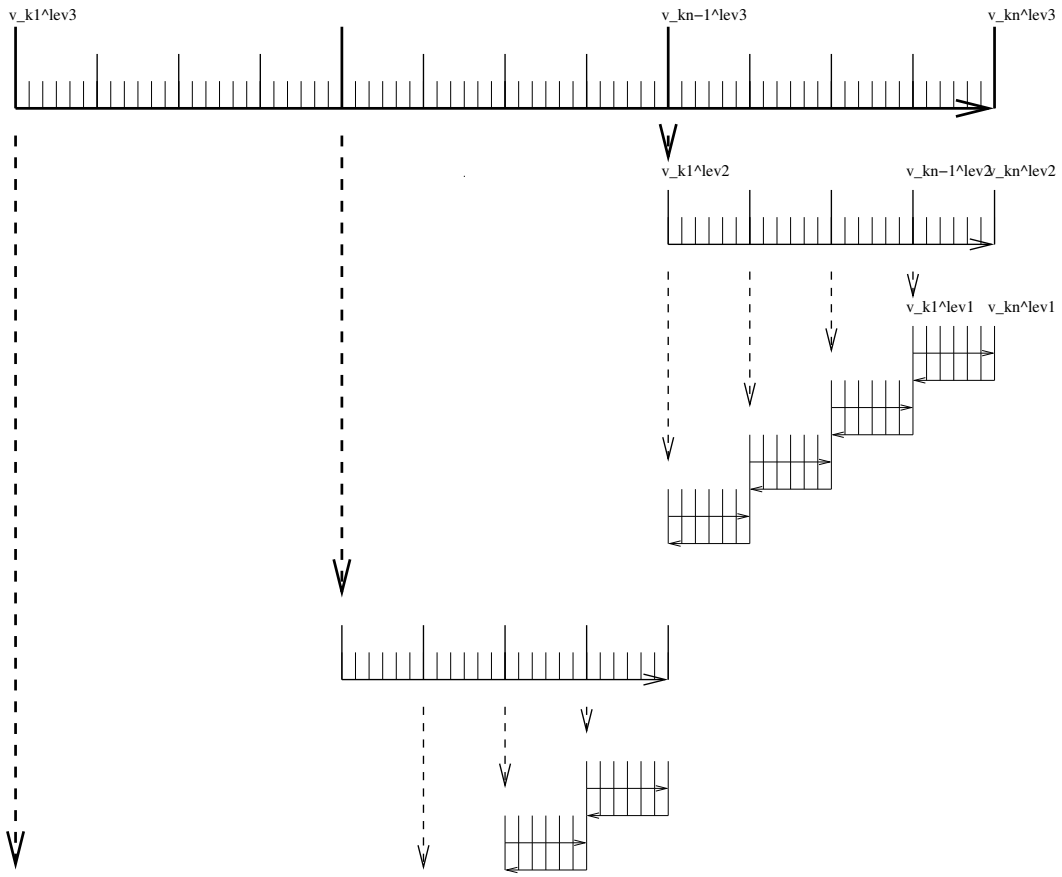


Figure 5.1: Schematic view of intermediate dump and restart for 3-level checkpointing.

5.2 TLM and ADM generation in general

In this section we describe in a general fashion the parts of the code that are relevant for automatic differentiation using the software tool TAF.

The basic flow is depicted in 5.2. If CPP option `ALLOW_AUTODIFF_TAMC` is defined, the driver routine *the_model_main*, instead of calling *the_main_loop*, invokes the adjoint of this routine, *adthe_main_loop*, which is the toplevel routine in terms of automatic differentiation. The routine *adthe_main_loop* has been generated by TAF. It contains both the forward integration of the full model, the cost function calculation, any additional storing that is required for efficient checkpointing, and the reverse integration of the adjoint model.

[DESCRIBE IN A SEPARATE SECTION THE WORKING OF THE TLM]

In Fig. 5.2 the structure of *adthe_main_loop* has been strongly simplified to

focus on the essentials; in particular, no checkpointing procedures are shown here. Prior to the call of *adthe_main_loop*, the routine *ctrl_unpack* is invoked to unpack the control vector or initialise the control variables. Following the call of *adthe_main_loop*, the routine *ctrl_pack* is invoked to pack the control vector (cf. Section 5.2.5). If gradient checks are to be performed, the option `ALLOW_GRADIENT_CHECK` is defined. In this case the driver routine *grdchk_main* is called after the gradient has been computed via the adjoint (cf. Section **??**).

5.2.1 General setup

In order to configure AD-related setups the following packages need to be enabled: The packages are enabled by adding them to your experiment-specific configuration file *packages.conf* (see Section ???).

The following AD-specific CPP option files need to be customized:

```
the_model_main
|
|--- initialise_fixed
|
|--- #ifdef ALLOW_ADJOINT_RUN
|           |
|           |--- ctrl_unpack
|           |
|           |--- adthe_main_loop
|           |    |
|           |    |--- initialise_varia
|           |    |--- ctrl_map_forcing
|           |    |--- do iloop = 1, nTimeSteps
|           |    |       |--- forward_step
|           |    |       |--- cost_tile
|           |    |    end do
|           |    |--- cost_final
|           |    |
|           |    |--- adcost_final
|           |    |--- do iloop = nTimeSteps, 1, -1
|           |    |       |--- adcost_tile
|           |    |       |--- adforward_step
|           |    |    end do
|           |    |--- adctrl_map_forcing
|           |    |--- adinitialise_varia
|           |    o
|           |
|           |--- ctrl_pack
|           |
|--- #else
|           |
|           |--- the_main_loop
|           |
|    #endif
|
|--- #ifdef ALLOW_GRADIENT_CHECK
|           |
|           |--- grdchk_main
|           o
|    #endif
o
```

Figure 5.2:

autodiff
ctrl
cost
grdchk

- *ECCO_CPPOPTIONS.h*
  This header file collects CPP options for the packages *autodiff, cost, ctrl* as well as AD-unrelated options for the external forcing package *exf.* [1]
- *tamc.h*
  This header configures the splitting of the time stepping loop w.r.t. the 3-level checkpointing (see section ???).

5.2.2 Building the AD code

The build process of an AD code is very similar to building the forward model. However, depending on which AD code one wishes to generate, and on which AD tool is available (TAF or TAMC), the following `make` targets are available:

	AD-target	*output*	*description*
(1)	`<MODE><TOOL>only`	`<MODE>_<TOOL>_output.f`	generates code for <MODE> using <TOOL> no `make` dependencies on `.F .h` useful for compiling on remote platforms
(2)	`<MODE><TOOL>`	`<MODE>_<TOOL>_output.f`	generates code for <MODE> using <TOOL> includes `make` dependencies on `.F .h` i.e. input for <TOOL> may be re-generated
(3)	`<MODE>all`	`mitgcmuv_<MODE>`	generates code for <MODE> using <TOOL> and compiles all code (use of TAF is set as default)

Here, the following placeholders are used

<TOOL>
- `TAF`
- `TAMC`

<MODE>
- `ad` generates the adjoint model (ADM)
- `ftl` generates the tangent linear model (TLM)
- `svd` generates both ADM and TLM for singular value decomposition (SVD) type calculations

For example, to generate the adjoint model using TAF after routines (`.F`) or headers (`.h`) have been modified, but without compilation, type `make adtaf`; or, to generate the tangent linear model using TAMC without re-generating the input code, type `make ftltamconly`.

[1] NOTE: These options are not set in their package-specific headers such as *COST_CPPOPTIONS.h*, but are instead collected in the single header file *ECCO_CPPOPTIONS.h*. The package-specific header files serve as simple placeholders at this point.

A typical full build process to generate the ADM via TAF would look like follows:

```
% mkdir build
% cd build
% ../../../tools/genmake2 -mods=../code_ad
% make depend
% make adall
```

5.2.3 The AD build process in detail

The `make <MODE>all` target consists of the following procedures:

1. A header file `AD_CONFIG.h` is generated which contains a CPP option on which code ought to be generated. Depending on the `make` target, the contents is
   - `#define ALLOW_ADJOINT_RUN`
   - `#define ALLOW_TANGENTLINEAR_RUN`
   - `#define ALLOW_ECCO_OPTIMIZATION`
2. A single file `<MODE>_input_code.f` is concatenated consisting of all `.f` files that are part of the list **AD_FILES** and all `.flow` files that are part of the list **AD_FLOW_FILES**.
3. The AD tool is invoked with the **¡MODE¿_¡TOOL¿_FLAGS**. The default AD tool flags in `genmake2` can be overrwritten by an `adjoint_options` file (similar to the platform-specific `build_options`, see Section ???. The AD tool writes the resulting AD code into the file `<MODE>_input_code_ad.f`
4. A short sed script `adjoint_sed` is applied to `<MODE>_input_code_ad.f` to reinstate **myThid** into the CALL argument list of active file I/O. The result is written to file `<MODE>_<TOOL>_output.f`.
5. All routines are compiled and an executable is generated (see Table ???).

5.2.3.1 The list AD_FILES and `.list` files

Not all routines are presented to the AD tool. Routines typically hidden are diagnostics routines which do not influence the cost function, but may create artificial flow dependencies such as I/O of active variables.

`genmake2` generates a list (or variable) **AD_FILES** which contains all routines that are shown to the AD tool. This list is put together from all files with suffix `.list` that `genmake2` finds in its search directories. The list file for the core MITgcm routines is in `model/src/` is called `model_ad_diff.list`. Note that no wrapper routine is shown to TAF. These are either not visible at all to the AD code, or hand-written AD code is available (see next section).

Each package directory contains its package-specific list file `<PKG>_ad_diff.list`. For example, `pkg/ptracers/` contains the file `ptracers_ad_diff.list`. Thus, enabling a package will automatically extend the **AD_FILES** list of `genmake2` to incorporate the package-specific routines. Note that you will need to regenerate the `Makefile` if you enable a package (e.g. by adding it to `packages.conf`) and a `Makefile` already exists.

5.2.3.2 The list AD_FLOW_FILES and `.flow` files

TAMC and TAF can evaluate user-specified directives that start with a specific syntax (`CADJ`, `C$TAF`, `!$TAF`). The main categories of directives are STORE directives and FLOW directives. Here, we are concerned with flow directives, store directives are treated elsewhere.

Flow directives enable the AD tool to evaluate how it should treat routines that are 'hidden' by the user, i.e. routines which are not contained in the **AD_FILES** list (see previous section), but which are called in part of the code that the AD tool does see. The flow directive tell the AD tool

- which subroutine arguments are input/output
- which subroutine arguments are active
- which subroutine arguments are required to compute the cost
- which subroutine arguments are dependent

The syntax for the flow directives can be found in the AD tool manuals.

`genmake2` generates a list (or variable) **AD_FLOW_FILES** which contains all files with suffix `.flow` that it finds in its search directories. The flow directives for the core MITgcm routines of `eesupp/src/` and `model/src/` reside in `pkg/autodiff/`. This directory also contains hand-written adjoint code for the MITgcm WRAPPER (see Section ???).

Flow directives for package-specific routines are contained in the corresponding package directories in the file `<PKG>_ad.flow`, e.g. ptracers-specific directives are in `ptracers_ad.flow`.

5.2.3.3 Store directives for 3-level checkpointing

The storing that is required at each period of the 3-level checkpointing is controled by three top-level headers.

```
do ilev_3 = 1, nchklev_3
#  include ''checkpoint_lev3.h''
   do ilev_2 = 1, nchklev_2
#     include ''checkpoint_lev2.h''
      do ilev_1 = 1, nchklev_1
#        include ''checkpoint_lev1.h''
```

```
...

      end do
   end do
end do
```

All files `checkpoint_lev?.h` are contained in directory `pkg/autodiff/`.

5.2.3.4 Changing the default AD tool flags: ad_options files

5.2.3.5 Hand-written adjoint code

5.2.4 The cost function (dependent variable)

The cost function $\mathcal{J}$ is referred to as the dependent variable. It is a function of the input variables $\vec{u}$ via the composition $\mathcal{J}(\vec{u}) = \mathcal{J}(M(\vec{u}))$. The input are referred to as the independent variables or control variables. All aspects relevant to the treatment of the cost function $\mathcal{J}$ (parameter setting, initialization, accumulation, final evaluation), are controlled by the package *pkg/cost*. The aspects relevant to the treatment of the independent variables are controlled by the package *pkg/ctrl* and will be treated in the next section.

```
 the_model_main
 |
 |-- initialise_fixed
 |   |
 |   |-- packages_readparms
 |       |
 |       |-- cost_readparms
 |       o
 |
 |-- the_main_loop
...  |
      |-- initialise_varia
      |   |
      |   |-- packages_init_variables
      |       |
      |       |-- cost_init
      |       o
      |
      |-- do iloop = 1,nTimeSteps
      |      |-- forward_step
      |      |-- cost_tile
      |      |   |
      |      |   |-- cost_tracer
      |   end do
      |
      |-- cost_final
      o
```

Figure 5.3:

5.2.4.1 Enabling the package

packages.conf, *ECCO_CPPOPTIONS.h*

- The package is enabled by adding *cost* to your file *packages.conf* (see Section ???)
-

N.B.: In general the following packages ought to be enabled simultaneously: *autodiff, cost, ctrl*. The basic CPP option to enable the cost function is **ALLOW_COST**. Each specific cost function contribution has its own option. For the present example the option is **ALLOW_COST_TRACER**. All cost-specific options are set in *ECCO_CPPOPTIONS.h* Since the cost function is usually used in conjunction with automatic differentiation, the CPP option **ALLOW_ADJOINT_RUN** (file *CPP_OPTIONS.h*) and **ALLOW_AUTODIFF_TAMC** (file *ECCO_CPPOPTIONS.h*) should be defined.

5.2.4.2 Initialization

The initialization of the *cost* package is readily enabled as soon as the CPP option **ALLOW_COST** is defined.

- Parameters: *cost_readparms*

  This S/R reads runtime flags and parameters from file *data.cost*. For the present example the only relevant parameter read is **mult_tracer**. This multiplier enables different cost function contributions to be switched on (= 1.) or off (= 0.) at runtime. For more complex cost functions which involve model vs. data misfits, the corresponding data filenames and data specifications (start date and time, period, ...) are read in this S/R.

- Variables: *cost_init*

  This S/R initializes the different cost function contributions. The contribution for the present example is **objf_tracer** which is defined on each tile (bi,bj).

5.2.4.3 Accumulation

- *cost_tile*, *cost_tracer*

The 'driver' routine *cost_tile* is called at the end of each time step. Within this 'driver' routine, S/R are called for each of the chosen cost function contributions. In the present example (**ALLOW_COST_TRACER**), S/R *cost_tracer* is called. It accumulates **objf_tracer** according to eqn. (**??**).

5.2.4.4 Finalize all contributions

- *cost_final*

At the end of the forward integration S/R *cost_final* is called. It accumulates the total cost function **fc** from each contribution and sums over all tiles:

$$\mathcal{J} = \text{fc} = \text{mult_tracer} \sum_{\text{global sum}} \sum_{bi,\,bj}^{nSx,\,nSy} \text{objf_tracer}(bi,bj) + ... \qquad (5.13)$$

The total cost function **fc** will be the 'dependent' variable in the argument list for TAMC, i.e.

```
tamc -output 'fc' ...
```

5.2.5 The control variables (independent variables)

The control variables are a subset of the model input (initial conditions, boundary conditions, model parameters). Here we identify them with the variable $\vec{u}$. All intermediate variables whose derivative w.r.t. control variables do not vanish are called active variables. All subroutines whose derivative w.r.t. the control variables don't vanish are called active routines. Read and write operations from and to file can be viewed as variable assignments. Therefore, files to which active variables are written and from which active variables are read are called active files. All aspects relevant to the treatment of the control variables (parameter setting, initialization, perturbation) are controlled by the package *pkg/ctrl*.

5.2.5.1 genmake and CPP options

- *genmake*, *CPP_OPTIONS.h*, *ECCO_CPPOPTIONS.h*

To enable the directory to be included to the compile list, **ctrl** has to be added to the **enable** list in *.genmakerc* or in *genmake* itself (analogous to *cost* package, cf. previous section). Each control variable is enabled via its own CPP option in *ECCO_CPPOPTIONS.h*.

5.2.5.2 Initialization

- Parameters: *ctrl_readparms*

  This S/R reads runtime flags and parameters from file *data.ctrl*. For the present example the file contains the file names of each control variable that is used. In addition, the number of wet points for each control variable and the net dimension of the space of control variables (counting wet points only) **nvarlength** is determined. Masks for wet points for each tile **(bi, bj)** and vertical layer **k** are generated for the three relevant categories on the C-grid: **nWetCtile** for tracer fields, **nWetWtile** for zonal velocity fields, **nWetStile** for meridional velocity fields.

- Control variables, control vector, and their gradients: *ctrl_unpack*

  Two important issues related to the handling of the control variables in the MITGCM need to be addressed. First, in order to save memory, the control variable arrays are not kept in memory, but rather read from file and added to the initial fields during the model initialization phase. Similarly, the corresponding adjoint fields which represent the gradient of the cost function w.r.t. the control variables are written to file at the end of

```
 *************
 the_main_loop
 *************
 |
 |--- initialise_varia
 |    |
 |   ...
 |    |--- packages_init_varia
 |    |    |
 |    |   ...
 |    |    |--- #ifdef ALLOW_ADJOINT_RUN
 |    |    |          call ctrl_map_ini
 |    |    |          call cost_ini
 |    |    |    #endif
 |    |   ...
 |    |    o
 |   ...
 |    o
...
 |--- #ifdef ALLOW_ADJOINT_RUN
 |          call ctrl_map_forcing
 |    #endif
...
 |--- #ifdef ALLOW_TAMC_CHECKPOINTING
 |           do ilev_3 = 1,nchklev_3
 |             do ilev_2 = 1,nchklev_2
 |               do ilev_1 = 1,nchklev_1
 |                 iloop = (ilev_3-1)*nchklev_2*nchklev_1 +
 |                         (ilev_2-1)*nchklev_1           + ilev_1
 |    #else
 |          do iloop = 1, nTimeSteps
 |    #endif
 |    |
 |    |---       call forward_step
 |    |
 |    |--- #ifdef ALLOW_COST
 |    |          call cost_tile
 |    |    #endif
 |    |
 |    |    enddo
 |    o
 |
 |--- #ifdef ALLOW_COST
 |          call cost_final
 |    #endif
 o
```

Figure 5.4:

```
the_model_main
|
|-- initialise_fixed
|   |
|   |-- packages_readparms
|       |
|       |-- ctrl_init               - initialise control
|       o                             package
|
|-- ctrl_unpack                     - unpack control vector
|
|-- adthe_main_loop                 - forward/adjoint run
|   |
|   |-- initialise_variables
|   |   |
|   |   |-- packages_init_variables
|   |       |
|   |       |-- ctrl_map_ini        - link init. state and
|   |       o                         parameters to control
|   |                                 variables
|   |-- ctrl_map_forcing            - link forcing fields to
|  ...                                control variables
|
|-- ctrl_pack                       - pack control vector
```

Figure 5.5:

the adjoint integration. Second, in addition to the files holding the 2-dim. and 3-dim. control variables and the corresponding cost gradients, a 1-dim. control vector and gradient vector are written to file. They contain only the wet points of the control variables and the corresponding gradient. This leads to a significant data compression. Furthermore, an option is available (`ALLOW_NONDIMENSIONAL_CONTROL_IO`) to non-dimensionalise the control and gradient vector, which otherwise would contain different pieces of different magnitudes and units. Finally, the control and gradient vector can be passed to a minimization routine if an update of the control variables is sought as part of a minimization exercise.

The files holding fields and vectors of the control variables and gradient are generated and initialised in S/R *ctrl_unpack*.

5.2.5.3 Perturbation of the independent variables

The dependency flow for differentiation w.r.t. the controls starts with adding a perturbation onto the input variable, thus defining the independent or control variables for TAMC. Three types of controls may be considered:

- *ctrl_map_ini* (initial value sensitivity):
  Consider as an example the initial tracer distribution **tr1** as control variable. After **tr1** has been initialised in *ini_tr1* (dynamical variables such as temperature and salinity are initialised in *ini_fields*), a perturbation

anomaly is added to the field in S/R *ctrl_map_ini*

$$\begin{aligned} u &= u_{[0]} + \Delta u \\ \mathbf{tr1}(...) &= \mathbf{tr1_{ini}}(...) + \mathbf{xx_tr1}(...) \end{aligned} \tag{5.14}$$

xx_tr1 is a 3-dim. global array holding the perturbation. In the case of a simple sensitivity study this array is identical to zero. However, it's specification is essential in the context of automatic differentiation since TAMC treats the corresponding line in the code symbolically when determining the differentiation chain and its origin. Thus, the variable names are part of the argument list when calling TAMC:

```
tamc -input 'xx_tr1 ...' ...
```

Now, as mentioned above, the MITGCM avoids maintaining an array for each control variable by reading the perturbation to a temporary array from file. To ensure the symbolic link to be recognized by TAMC, a scalar dummy variable **xx_tr1_dummy** is introduced and an 'active read' routine of the adjoint support package *pkg/autodiff* is invoked. The read-procedure is tagged with the variable **xx_tr1_dummy** enabling TAMC to recognize the initialization of the perturbation. The modified call of TAMC thus reads

```
tamc -input 'xx_tr1_dummy ...' ...
```

and the modified operation to (5.14) in the code takes on the form

```
      call active_read_xyz(
     &      ..., tmpfld3d, ..., xx_tr1_dummy, ... )

      tr1(...) = tr1(...) + tmpfld3d(...)
```

Note, that reading an active variable corresponds to a variable assignment. Its derivative corresponds to a write statement of the adjoint variable, followed by a reset. The 'active file' routines have been designed to support active read and corresponding adjoint active write operations (and vice versa).

- *ctrl_map_forcing* (boundary value sensitivity):

The handling of boundary values as control variables proceeds exactly analogous to the initial values with the symbolic perturbation taking place in S/R *ctrl_map_forcing*. Note however an important difference: Since the boundary values are time dependent with a new forcing field applied at each time steps, the general problem may be thought of as a new control variable at each time step (or, if the perturbation is averaged over a certain period, at each N timesteps), i.e.

$$u_{\text{forcing}} = \{\, u_{\text{forcing}}(t_n) \,\}_{n=1,\ldots,\text{nTimeSteps}}$$

In the current example an equilibrium state is considered, and only an initial perturbation to surface forcing is applied with respect to the equilibrium state. A time dependent treatment of the surface forcing is implemented in the ECCO environment, involving the calendar (*cal*) and external forcing (*exf*) packages.

- *ctrl_map_params* (parameter sensitivity):

This routine is not yet implemented, but would proceed proceed along the same lines as the initial value sensitivity. The mixing parameters **diffkr** and **kapgm** are currently added as controls in *ctrl_map_ini.F*.

5.2.5.4 Output of adjoint variables and gradient

Several ways exist to generate output of adjoint fields.

- *ctrl_map_ini, ctrl_map_forcing*:

  - **xx_...**: the control variable fields
    Before the forward integration, the control variables are read from file **xx_ ...** and added to the model field.
  - **adxx_...**: the adjoint variable fields, i.e. the gradient $\nabla_u \mathcal{J}$ for each control variable
    After the adjoint integration the corresponding adjoint variables are written to **adxx_ ...**.

- *ctrl_unpack, ctrl_pack*:

  - **vector_ctrl**: the control vector
    At the very beginning of the model initialization, the updated compressed control vector is read (or initialised) and distributed to 2-dim. and 3-dim. control variable fields.
  - **vector_grad**: the gradient vector
    At the very end of the adjoint integration, the 2-dim. and 3-dim. adjoint variables are read, compressed to a single vector and written to file.

- *addummy_in_stepping*:

In addition to writing the gradient at the end of the forward/adjoint integration, many more adjoint variables of the model state at intermediate times can be written using S/R *addummy_in_stepping*. This routine is part of the adjoint support package *pkg/autodiff* (cf.f. below). The procedure is enabled using via the CPP-option **ALLOW_AUTODIFF_MONITOR** (file *ECCO_CPPOPTIONS.h*). To be part of the adjoint code, the corresponding S/R *dummy_in_stepping* has to be called in the forward model

(S/R *the_main_loop*) at the appropriate place. The adjoint common blocks are extracted from the adjoint code via the header file *adcommon.h*.

dummy_in_stepping is essentially empty, the corresponding adjoint routine is hand-written rather than generated automatically. Appropriate flow directives (*dummy_in_stepping.flow*) ensure that TAMC does not automatically generate *addummy_in_stepping* by trying to differentiate *dummy_in_stepping*, but instead refers to the hand-written routine.

dummy_in_stepping is called in the forward code at the beginning of each timestep, before the call to *dynamics*, thus ensuring that *addummy_in_stepping* is called at the end of each timestep in the adjoint calculation, after the call to *addynamics*.

addummy_in_stepping includes the header files *adcommon.h*. This header file is also hand-written. It contains the common blocks **/addynvars_r/**, **/addynvars_cd/**, **/addynvars_diffkr/**, **/addynvars_kapgm/**, **/adtr1_r/**, **/adffields/**, which have been extracted from the adjoint code to enable access to the adjoint variables.

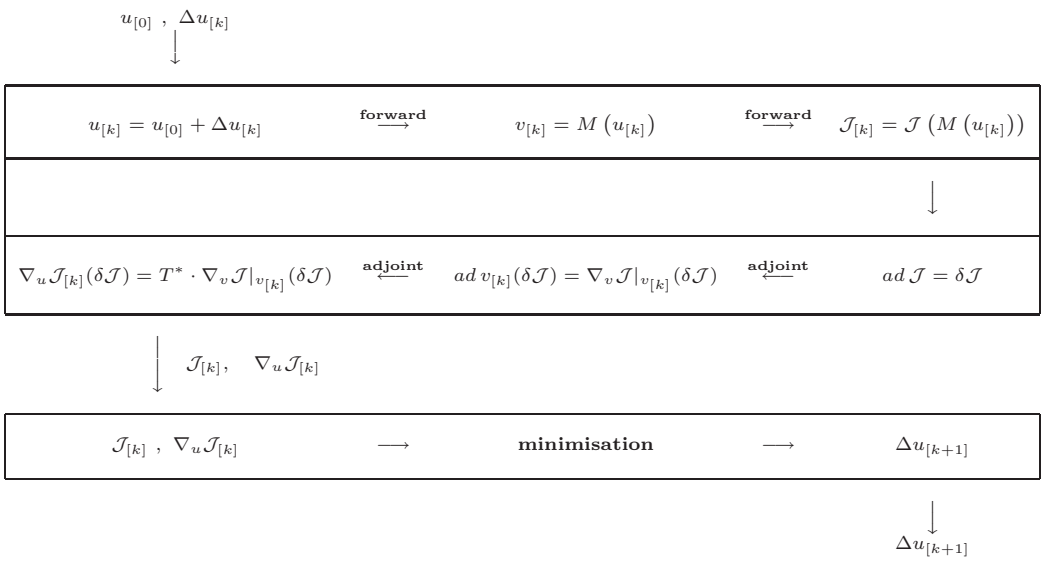
WARNING: If the structure of the common blocks **/dynvars_r/**, **/dynvars_cd/**, etc., changes similar changes will occur in the adjoint common blocks. Therefore, consistency between the TAMC-generated common blocks and those in *adcommon.h* have to be checked.

5.2.5.5 Control variable handling for optimization applications

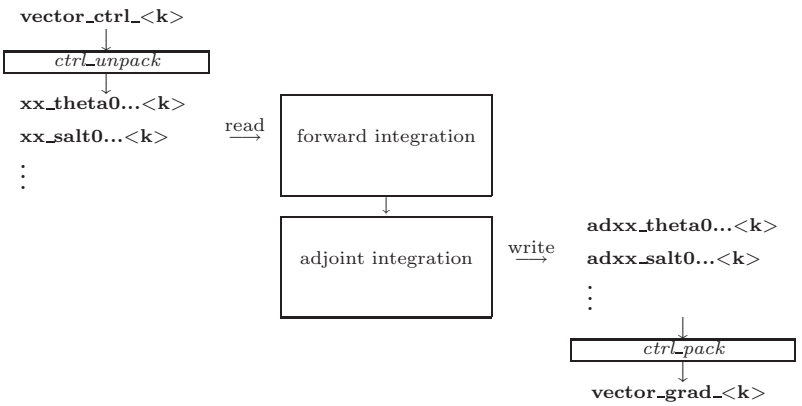
In optimization mode the cost function $\mathcal{J}(u)$ is sought to be minimized with respect to a set of control variables $\delta\mathcal{J} = 0$, in an iterative manner. The gradient $\nabla_u \mathcal{J}|_{u_{[k]}}$ together with the value of the cost function itself $\mathcal{J}(u_{[k]})$ at iteration step k serve as input to a minimization routine (e.g. quasi-Newton method, conjugate gradient, ... [21]) to compute an update in the control variable for iteration step $k+1$

$$u_{[k+1]} = u_{[0]} + \Delta u_{[k+1]} \quad \text{satisfying} \quad \mathcal{J}\left(u_{[k+1]}\right) < \mathcal{J}\left(u_{[k]}\right)$$

$u_{[k+1]}$ then serves as input for a forward/adjoint run to determine $\mathcal{J}$ and $\nabla_u \mathcal{J}$ at iteration step $k+1$. Tab. **??** sketches the flow between forward/adjoint model and the minimization routine.



The routines *ctrl_unpack* and *ctrl_pack* provide the link between the model and the minimization routine. As described in Section **??** the *unpack* and *pack* routines read and write control and gradient *vectors* which are compressed to contain only wet points, in addition to the full 2-dim. and 3-dim. fields. The corresponding I/O flow looks as follows:



ctrl_unpack reads the updated control vector **vector_ctrl_<k>**. It distributes the different control variables to 2-dim. and 3-dim. files *xx_...<k>*. At the start of the forward integration the control variables are read from *xx_...<k>* and added to the field. Correspondingly, at the end of the adjoint integration the adjoint fields are written to *adxx_...<k>*, again via the active file routines. Finally, *ctrl_pack* collects all adjoint files and writes them to the compressed vector file **vector_grad_<k>**.

5.3 Sensitivity of Air-Sea Exchange to Tracer Injection Site

The MITGCM has been adapted to enable AD using TAMC or TAF. The present description, therefore, is specific to the use of TAMC or TAF as AD tool. The following sections describe the steps which are necessary to generate a tangent linear or adjoint model of the MITGCM. We take as an example the sensitivity of carbon sequestration in the ocean. The AD-relevant hooks in the code are sketched in 5.2, 5.4.

5.3.1 Overview of the experiment

We describe an adjoint sensitivity analysis of out-gassing from the ocean into the atmosphere of a carbon-like tracer injected into the ocean interior (see [29]).

5.3.1.1 Passive tracer equation

For this work the MITGCM was augmented with a thermodynamically inactive tracer, C. Tracer residing in the ocean model surface layer is out-gassed according to a relaxation time scale, μ. Within the ocean interior, the tracer is passively advected by the ocean model currents. The full equation for the time evolution

$$\frac{\partial C}{\partial t} = -U \cdot \nabla C - \mu C + \Gamma(C) + S \tag{5.15}$$

also includes a source term S. This term represents interior sources of C such as would arise due to direct injection. The velocity term, U, is the sum of the model Eulerian circulation and an eddy-induced velocity, the latter parameterized according to Gent/McWilliams ([16, 17]). The convection function, Γ, mixes C vertically wherever the fluid is locally statically unstable.

The out-gassing time scale, μ, in eqn. (5.15) is set so that $1/\mu \sim 1$ year for the surface ocean and $\mu = 0$ elsewhere. With this value, eqn. (5.15) is valid as a prognostic equation for small perturbations in oceanic carbon concentrations. This configuration provides a powerful tool for examining the impact of large-scale ocean circulation on CO_2 out-gassing due to interior injections. As source we choose a constant in time injection of $S = 1$ mol/s.

5.3.1.2 Model configuration

The model configuration employed has a constant $4^\circ \times 4^\circ$ resolution horizontal grid and realistic geography and bathymetry. Twenty vertical layers are used with vertical spacing ranging from 50 m near the surface to 815 m at depth. Driven to steady-state by climatological wind-stress, heat and fresh-water forcing the model reproduces well known large-scale features of the ocean general circulation.

5.3.1.3 Out-gassing cost function

To quantify and understand out-gassing due to injections of C in eqn. (5.15), we define a cost function $\mathcal{J}$ that measures the total amount of tracer out-gassed at each timestep:

$$\mathcal{J}(t=T) = \int_{t=0}^{t=T} \int_A \mu C \, dA \, dt \tag{5.16}$$

Equation(5.16) integrates the out-gassing term, μC, from (5.15) over the entire ocean surface area, A, and accumulates it up to time T. Physically, $\mathcal{J}$ can be thought of as representing the amount of CO_2 that our model predicts would be out-gassed following an injection at rate S. The sensitivity of $\mathcal{J}$ to the spatial location of S, $\frac{\partial \mathcal{J}}{\partial S}$, can be used to identify regions from which circulation would cause CO_2 to rapidly out-gas following injection and regions in which CO_2 injections would remain effectively sequestered within the ocean.

5.3.2 Code configuration

The model configuration for this experiment resides under the directory *verification/carbon/*. The code customization routines are in *verification/carbon/code/*:

- *.genmakerc*
- *COST_CPPOPTIONS.h*
- *CPP_EEOPTIONS.h*
- *CPP_OPTIONS.h*
- *CTRL_OPTIONS.h*
- *ECCO_OPTIONS.h*
- *SIZE.h*
- *adcommon.h*
- *tamc.h*

The runtime flag and parameters settings are contained in *verification/carbon/input/*, together with the forcing fields and and restart files:

- *data*
- *data.cost*
- *data.ctrl*
- *data.gmredi*
- *data.grdchk*
- *data.optim*

- *data.pkg*
- *eedata*
- *topog.bin*
- *windx.bin, windy.bin*
- *salt.bin, theta.bin*
- *SSS.bin, SST.bin*
- *pickup**

Finally, the file to generate the adjoint code resides in *adjoint/*:

- *makefile*

Below we describe the customizations of this files which are specific to this experiment.

5.3.2.1 File *.genmakerc*

This file overwrites default settings of *genmake*. In the present example it is used to switch on the following packages which are related to automatic differentiation and are disabled by default:

```
set ENABLE=( autodiff cost ctrl ecco gmredi grdchk kpp )
```

Other packages which are not needed are switched off:

```
set DISABLE=( aim obcs zonal_filt shap_filt cal exf )
```

5.3.2.2 File *COST_CPPOPTIONS.h, CTRL_OPTIONS.h*

These files used to contain package-specific CPP-options (see Section **??**). For technical reasons those options have been grouped together in the file *ECCO_OPTIONS.h*. To retain the modularity, the files have been kept and contain the standard include of the *CPP_OPTIONS.h* file.

5.3.2.3 File *CPP_EEOPTIONS.h*

This file contains 'wrapper'-specific CPP options. It only needs to be changed if the code is to be run in a parallel environment (see Section **??**).

5.3.2.4 File *CPP_OPTIONS.h*

This file contains model-specific CPP options (see Section **??**). Most options are related to the forward model setup. They are identical to the global steady circulation setup of *verification/global_ocean.90x40x15/*. The three options specific to this experiment are

```
#define ALLOW_PASSIVE_TRACER
```

This flag enables the code to carry through the advection/diffusion of a passive

tracer along the model integration.

```
#define ALLOW_MIT_ADJOINT_RUN
```

This flag enables the inclusion of some AD-related fields concerning initialization, link between control variables and forward model variables, and the call to the top-level forward/adjoint subroutine *adthe_main_loop* instead of *the_main_loop*.

```
#define ALLOW_GRADIENT_CHECK
```

This flag enables the gradient check package. After computing the unperturbed cost function and its gradient, a series of computations are performed for which
• an element of the control vector is perturbed
• the cost function w.r.t. the perturbed element is computed
• the difference between the perturbed and unperturbed cost function is computed to compute the finite difference gradient
• the finite difference gradient is compared with the adjoint-generated gradient.
The gradient check package is further described in Section ???.

5.3.2.5 File *ECCO_OPTIONS.h*

The CPP options of several AD-related packages are grouped in this file:

- Overall ECCO-related execution modus:
  These determine whether a pure forward run, a sensitivity run or an iteration of optimization is performed. These options are not needed in the present context.

- Adjoint support package: *pkg/autodiff/*
  This package contains hand-written adjoint code such as active file handling, flow directives for files which must not be differentiated, and TAMC-specific header files.
  ```
  #define ALLOW_AUTODIFF_TAMC
  ```
  defines TAMC-related features in the code.
  ```
  #define ALLOW_TAMC_CHECKPOINTING
  ```
  enables the checkpointing feature of TAMC (see Section **??**). In the present example a 3-level checkpointing is implemented. The code contains the relevant store directives, common block and tape initializations, storing key computation, and loop index handling. The checkpointing length at each level is defined in file *tamc.h*, cf. below. The out and intermediate loop directivs are contained in the files *checkpoint_lev3_directives.h*, *checkpoint_lev2_directives.h* (package *pkg/autodiff*).
  ```
  #define ALLOW_AUTODIFF_MONITOOR
  ```
  enables the monitoring of intermediate adjoint variables (see Section **??**).
  ```
  #define ALLOW_DIVIDED_ADJOINT
  ```
  enables adjoint dump and restart (see Section **??**).

- Cost function package: *pkg/cost/*
  This package contains all relevant routines for initializing, accumulating and finalizing the cost function (see Section **??**).
  ```
  #define ALLOW_COST
  ```

enables all general aspects of the cost function handling, in particular the hooks in the forward code for initializing, accumulating and finalizing the cost function.

```
#define ALLOW_COST_TRACER
```

includes the call to the cost function for this particular experiment, eqn. (5.16).

- Control variable package: *pkg/ctrl/*
  This package contains all relevant routines for the handling of the control vector. Each control variable can be enabled/disabled with its own flag:

`#define ALLOW_THETA0_CONTROL`	initial temperature
`#define ALLOW_SALT0_CONTROL`	initial salinity
`#define ALLOW_TR0_CONTROL`	initial passive tracer concentration
`#define ALLOW_TAUU0_CONTROL`	zonal wind stress
`#define ALLOW_TAUV0_CONTROL`	meridional wind stress
`#define ALLOW_SFLUX0_CONTROL`	freshwater flux
`#define ALLOW_HFLUX0_CONTROL`	heat flux
`#define ALLOW_DIFFKR_CONTROL`	diapycnal diffusivity
`#undef ALLOW_KAPPAGM_CONTROL`	isopycnal diffusivity

5.3.2.6 File *SIZE.h*

The file contains the grid point dimensions of the forward model. It is identical to the *verification/exp2/*:

```
sNx = 90
sNy = 40
Nr = 20
```

It corresponds to a single-tile/single-processor setup: `nSx = nSy = 1, nPx = nPy = 1`, with standard overlap dimensioning `OLx = OLy = 3`.

5.3.2.7 File *adcommon.h*

This file contains common blocks of some adjoint variables that are generated by TAMC. The common blocks are used by the adjoint support routine *addummy_in_stepping* which needs to access those variables:

`common /addynvars_r/`	is related to *DYNVARS.h*
`common /addynvars_cd/`	is related to *DYNVARS.h*
`common /addynvars_diffkr/`	is related to *DYNVARS.h*
`common /addynvars_kapgm/`	is related to *DYNVARS.h*
`common /adtr1_r/`	is related to *TR1.h*
`common /adffields/`	is related to *FFIELDS.h*

Note that if the structure of the common block changes in the above header files of the forward code, the structure of the adjoint common blocks will change accordingly. Thus, it has to be made sure that the structure of the adjoint common block in the hand-written file *adcommon.h* complies with the automatically generated adjoint common blocks in *adjoint_model.F*. The header file is enabled via the CPP-option **ALLOW_AUTODIFF_MONITOR**.

5.3.2.8 File *tamc.h*

This routine contains the dimensions for TAMC checkpointing and some indices relevant for storing ky computations.

- `#ifdef ALLOW_TAMC_CHECKPOINTING`
  3-level checkpointing is enabled, i.e. the timestepping is divided into three different levels (see Section **??**). The model state of the outermost (`nchklev_3`) and the intermediate (`nchklev_2`) timestepping loop are stored to file (handled in *the_main_loop*). The innermost loop (`nchklev_1`) avoids I/O by storing all required variables to common blocks. This storing may also be necessary if no checkpointing is chosen (nonlinear functions, if-statements, iterative loops, ...). In the present example the dimensions are chosen as follows:
  ```
  nchklev_1 = 36
  nchklev_2 = 30
  nchklev_3 = 60
  ```
  To guarantee that the checkpointing intervals span the entire integration period the following relation must be satisfied:
  `nchklev_1*nchklev_2*nchklev_3` $\geq$ `nTimeSteps`
  where `nTimeSteps` is either specified in *data* or computed via
  `nTimeSteps = (endTime-startTime)/deltaTClock` .

- `#undef ALLOW_TAMC_CHECKPOINTING`
  No checkpointing is enabled. In this case the relevant counter is `nchklev_0`. Similar to above, the following relation has to be satisfied
  `nchklev_0` $\geq$ `nTimeSteps`.

The following parameters may be worth describing:
`isbyte`
`maxpass`

5.3.2.9 File *makefile*

This file contains all relevant parameter flags and lists to run TAMC or TAF. It is assumed that TAMC is available to you, either locally, being installed on your network, or remotely through the 'TAMC Utility'. TAMC is called with the command `tamc` followed by a number of options. They are described in detail in the TAMC manual [18]. Here we briefly discuss the main flags used in the *makefile*. The standard output for TAF is written to file *taf.log*.

```
tamc -input <variable names> -output <variable name> -i4 -r4 ...
     -toplevel <S/R name> -reverse <file names>

 taf -input <variable names> -output <variable name> -i4 -r4 ...
     -toplevel <S/R name> -reverse <file names>
     -flow taf_flow.log -nonew_arg
```

- `-toplevel <S/R name>`
  Name of the toplevel routine, with respect to which the control flow analysis is performed.

- `-input <variable names>`
  List of independent variables u with respect to which the dependent variable J is differentiated.

- `-output <variable name>`
  Dependent variable J which is to be differentiated.

- `-reverse <file names>`
  Adjoint code is generated to compute the sensitivity of an independent variable w.r.t. many dependent variables. In the discussion of Section ??? the generated adjoint top-level routine computes the product of the transposed Jacobian matrix M^T times the gradient vector $\nabla_v J$.
  `<file names>` refers to the list of files *.f* which are to be analyzed by TAMC. This list is generally smaller than the full list of code to be compiled. The files not contained are either above the top-level routine (some initializations), or are deliberately hidden from TAMC, either because hand-written adjoint routines exist, or the routines must not (or don't have to) be differentiated. For each routine which is part of the flow tree of the top-level routine, but deliberately hidden from TAMC (or for each package which contains such routines), a corresponding file *.flow* exists containing flow directives for TAMC.

- `-i4 -r4`

- `-flow taf_flow.log`
  Will cause TAF to produce a flow listing file named *taf_flow.log* in which the set of active and passive variables are identified for each subroutine.

- `-nonew_arg`
  The default in the order of the parameter list of adjoint routines has changed. Before TAF 1.3 the default was compatible with the TAMC-generated list. As of TAF 1.3 the order of adjoint routine parameter lists is no longer copatible with TAMC. To restore compatibility when using TAF 1.3 and higher, this argument is needed. It is currently crucial to use since all hand-written adjoint routines refer to the TAMC default.

5.3.2.10 The input parameter files

File *data*

File *data.cost*

File *data.ctrl*

File *data.gmredi*

File *data.grdchk*

File *data.optim*

File *data.pkg*

File *eedata*

File *topog.bin*
Contains two-dimendional bathymetry information

File *windx.bin, windy.bin, salt.bin, theta.bin, SSS.bin, SST.bin*
These contain the initial values (salinity, temperature, *salt.bin, theta.bin*), surface boundary values (surface wind stresses, (*windx.bin, windy.bin*), and surface restoring fields (*SSS.bin, SST.bin*).

File *pickup**
Contains model state after model spinup.

5.3.3 Compiling the model and its adjoint

The built process of the adjoint model is slightly more complex than that of compiling the forward code. The main reason is that the adjoint code generation requires a specific list of routines that are to be differentiated (as opposed to the automatic generation of a list of files to be compiled by genmake). This list excludes routines that don't have to be or must not be differentiated. For some of the latter routines flow directives may be necessary, a list of which has to be given as well. For this reason, a separate *makefile* is currently maintained in the directory `adjoint/`. This makefile is responsible for the adjoint code generation.

In the following we describe the build process step by step, assuming you are in the directory `bin/`. A summary of steps to follow is given at the end.

Adjoint code generation and compilation – step by step

1. `ln -s ../verification/???/code/.genmakerc .`
   `ln -s ../verification/???/code/*.[Fh] .`
   Link your customized genmake options, header files, and modified code to the compile directory.

2. `../tools/genmake -makefile`
   Generate your Makefile (cf. Section ???).

3. `make depend`
   Dependency analysis for the CPP pre-compiler (cf. Section ???).

4. `cd ../adjoint`
   `make adtaf` or `make adtamc`
   Depending on whether you have TAF or TAMC at your disposal, you'll choose `adtaf` or `adtamc` as your make target for the *makefile* in the directory `adjoint/`. Several things happen at this stage.

   (a) `make adrestore, make ftlrestore`
   The initial template files *adjoint_model.F* and *tangentlinear_model.F* in *pkg/autodiff* which are part of the compiling list created by *genmake* are restored.

   (b) `make depend, make small_f`
   The `bin/` directory is brought up to date, i.e. for recent changes in header or source code *.[Fh]*, corresponding *.f* routines are generated or re-generated. Note that here, only CPP precompiling is performed; no object code *.o* is generated as yet. Precompiling is necessary for TAMC to see the full code.

   (c) `make allcode`
   All Fortran routines `*.f` in `bin/` are concatenated into a single file called *tamc_code.f*.

   (d) `make admodeltaf/admodeltamc`
   Adjoint code is generated by TAMC or TAF. The adjoint code is written to the file *tamc_code_ad.f*. It contains all adjoint routines of the forward routines concatenated in *tamc_code.f*. For a given forward routines `subroutine routinename` the adjoint routine is named `adsubroutine routinename` by default (that default can be changed via the flag `-admark <markname>`). Furthermore, it may contain modified code which incorporates the translation of adjoint store directives into specific Fortran code. For a given forward routines `subroutine routinename` the modified routine is named `mdsubroutine routinename`. TAMC or TAF info is written to file *tamc_code.prot* or *taf.log*, respectively.

   (e) `make adchange`
   The multi-threading capability of the MITGCM requires a slight change in the parameter list of some routines that are related to to active file handling. This post-processing invokes the sed script *adjoint_ecco_sed.com* to insert the threading counter **myThId** into the parameter list of those subroutines. The resulting code is written to file *tamc_code_sed_ad.f* and appended to the file *adjoint_model.F*. This concludes the adjoint code generation.

5. `cd ../bin`
   `make`

The file *adjoint_model.F* now contains the full adjoint code. All routines are now compiled.

N.B.: The targets `make adtaf/adtamc` now comprise a series of targets that in previous versions had to be invoked separarely. This was probably preferable at a more experimental stage, but has now been dropped in favour of a more straightforward build process.

Adjoint code generation and compilation – summary

```
cd bin
ln -s ../verification/my_experiment/code/.genmakerc .
ln -s ../verification/my_experiment/code/*.[Fh] .
../tools/genmake -makefile
make depend
cd ../adjoint
make adtaf <OR: make adtamc>
     contains the targets:
     adrestore small_f allcode admodeltaf/admodeltamc adchange
cd ../bin
make
```

5.4 The gradient check package

An indispensable test to validate the gradient computed via the adjoint is a comparison against finite difference gradients. The gradient check package *pkg/grdchk* enables such tests in a straightforward and easy manner. The driver routine *grdchk_main* is called from *the_model_main* after the gradient has been computed via the adjoint model (cf. flow chart ???).

The gradient check proceeds as follows: The $i-$th component of the gradient $(\nabla_u \mathcal{J}^T)_i$ is compared with the following finite-difference gradient:

$$\left(\nabla_u \mathcal{J}^T\right)_i \quad \text{vs.} \quad \frac{\partial \mathcal{J}}{\partial u_i} = \frac{\mathcal{J}(u_i + \epsilon) - \mathcal{J}(u_i)}{\epsilon}$$

A gradient check at point u_i may generally considered to be successful if the deviation of the ratio between the adjoint and the finite difference gradient from unity is less than 1 percent,

$$1 - \frac{(\text{grad}\mathcal{J})_i(\text{adjoint})}{(\text{grad}\mathcal{J})_i(\text{finite difference})} < 1\%$$

5.4.1 Code description

5.4.2 Code configuration

The relevant CPP precompile options are set in the following files:

- *.genmakerc*
  option `grdchk` is added to the **enable list** (alternatively, *genmake* may be invoked with the option `-enable=grdchk`).
- *CPP_OPTIONS.h*
  Together with the flag **ALLOW_ADJOINT_RUN**, define the flag **ALLOW_GRADIENT_CHECK**.

The relevant runtime flags are set in the files

- *data.pkg*
  Set **useGrdchk = .TRUE.**
- *data.grdchk*
  - **grdchk_eps**
  - **nbeg**
  - **nstep**
  - **nend**
  - **grdchkvarindex**

```
the_model_main
|
|-- ctrl_unpack
|-- adthe_main_loop            - unperturbed cost function and
|-- ctrl_pack                    adjoint gradient are computed here
|
|-- grdchk_main
    |
    |-- grdchk_init
    |-- do icomp=...           - loop over control vector elements
        |
        |-- grdchk_loc         - determine location of icomp on grid
        |
        |-- grdchk_getxx       - get control vector component from file
        |                        perturb it and write back to file
        |-- grdchk_getadxx     - get gradient component calculated
        |                        via adjoint
        |-- the_main_loop      - forward run and cost evaluation
        |                        with perturbed control vector element
        |-- calculate ratio of adj. vs. finite difference gradient
        |
        |-- grdchk_setxx       - Reset control vector element
        |
        |-- grdchk_print       - print results
```

Figure 5.6:

5.5 Adjoint dump & restart – divided adjoint (DIVA)

Patrick Heimbach & Geoffrey Gebbie, MIT/EAPS, 07-Mar-2003

NOTE:
THIS SECTION IS SUBJECT TO CHANGE. IT REFERS TO TAF-1.4.26.

Previous TAF versions are incomplete and have problems with both TAF options '-pure' and '-mpi'.

The code which is tuned to the DIVA implementation of this TAF version is *checkpoint50* **(MITgcm) and** *ecco_c50_e28* **(ECCO).**

5.5.1 Introduction

Most high performance computing (HPC) centres require the use of batch jobs for code execution. Limits in maximum available CPU time and memory may prevent the adjoint code execution from fitting into any of the available queues. This presents a serious limit for large scale / long time adjoint ocean and climate model integrations. The MITgcm itself enables the split of the total model integration into sub-intervals through standard dump/restart of/from the full model state. For a similar procedure to run in reverse mode, the adjoint model requires, in addition to the model state, the adjoint model state, i.e. all variables with derivative information which are needed in an adjoint restart. This adjoint dump & restart is also termed 'divided adjoint (DIVA).

For this to work in conjunction with automatic differentiation, an AD tool needs to perform the following tasks:

1. identify an adjoint state, i.e. those sensitivities whose accumulation is interrupted by a dump/restart and which influence the outcome of the gradient. Ideally, this state consists of
   - the adjoint of the model state,
   - the adjoint of other intermediate results (such as control variables, cost function contributions, etc.)
   - bookkeeping indices (such as loop indices, etc.)
2. generate code for storing and reading adjoint state variables
3. generate code for bookkeeping , i.e. maintaining a file with index information
4. generate a suitable adjoint loop to propagate adjoint values for dump/restart with a minimum overhad of adjoint intermediate values.

TAF (but not TAMC!) generates adjoint code which performs the above specified tasks. It is closely tied to the adjoint multi-level checkpointing. The adjoint state is dumped (and restarted) at each step of the outermost checkpointing level and adjoint intergration is performed over one outermost checkpointing interval. Prior to the adjoint computations, a full foward sweep is performed to generate the outermost (forward state) tapes and to calculate the cost function. In the current implementation, the forward sweep is immediately followed by the first adjoint leg. Thus, in theory, the following steps are performed (automatically)

- **1st model call:**
  This is the case if file `costfinal` does *not* exist. S/R `mdthe_main_loop` is called.
  1. calculate forward trajectory and dump model state after each outermost checkpointing interval to files `tapelev3`
  2. calculate cost function `fc` and write it to file `costfinal`
- **2nd and all remaining model call:**
  This is the case if file `costfinal` *does* exist. S/R `adthe_main_loop` is called.
  1. (forward run and cost function call is avoided since all values are known)
     - if 1st adjoint leg:
       create index file `divided.ctrl` which contains info on current checkpointing index $ilev3$

   - if not i-th adjoint leg:
     adjoint picks up at $ilev3 = nlev3 - i + 1$ and runs to $nlev3 - i$
2. perform adjoint leg from $nlev3 - i + 1$ to $nlev3 - i$
3. dump adjoint state to file `snapshot`
4. dump index file `divided.ctrl` for next adjoint leg
5. in the last step the gradient is written.

A few modifications were performed in the forward code, obvious ones such as adding the corresponding TAF-directive at the appropriate place, and less obvious ones (avoid some re-initializations, when in an intermediate adjoint integration interval).

[For TAF-1.4.20 a number of hand-modifications were necessary to compensate for TAF bugs. Since we refer to TAF-1.4.26 onwards, these modifications are not documented here].

5.5.2 Recipe 1: single processor

1. In `ECCO_CPPOPTIONS.h` set:

```
#define ALLOW_DIVIDED_ADJOINT
#undef  ALLOW_DIVIDED_ADJOINT_MPI
```

2. Generate adjoint code. Using the TAF option '`-pure`', two codes are generated:

   - `mdthe_main_loop`:
     Is responsible for the forward trajectory, storing of outermost checkpoint levels to file, computation of cost function, and storing of cost function to file (1st step).
   - `adthe_main_loop`:
     Is responsible for computing one adjoint leg, dump adjoint state to file and write index info to file (2nd and consecutive steps).
     for adjoint code generation, e.g. add '`-pure`' to TAF option list

     ```
     make adtaf
     ```

   - One modification needs to be made to adjoint codes in S/R adecco_the_main_loop:
     There's a remaining issue with the '`-pure`' option. The '`call ad...`' between '`call ad...`' and the read of the `snapshot` file should be called only in the firt adjoint leg between $nlev3$ and $nlev3 - 1$. In the ecco-branch, the following lines should be bracketed by an `if (idivbeg .GE. nchklev_3) then`, thus:

```
...
      xx_psbar_mean_dummy = onetape_xx_psbar_mean_dummy_3h(1)
      xx_tbar_mean_dummy = onetape_xx_tbar_mean_dummy_4h(1)
      xx_sbar_mean_dummy = onetape_xx_sbar_mean_dummy_5h(1)
```

```
      call barrier( mythid )
cAdd(
      if (idivbeg .GE. nchklev_3) then
cAdd)

      call adcost_final( mythid )
      call barrier( mythid )
      call adcost_sst( mythid )
      call adcost_ssh( mythid )
      call adcost_hyd( mythid )
      call adcost_averagesfields( mytime,myiter,mythid )
      call barrier( mythid )
cAdd(
      endif
cAdd)

C-----------------------------------------------
C read snapshot
C-----------------------------------------------
      if (idivbeg .lt. nchklev_3) then
        open(unit=77,file='snapshot',status='old',form='unformatted',
     $iostat=iers)
...
```

For the main code, in all likelihood the block which needs to be bracketed consists of `adcost_final` only.

- Now the code can be copied as usual to `adjoint_model.F` and then be compiled:

```
make adchange
then compile
```

5.5.3 Recipe 2: multi processor (MPI)

1. On the machine where you execute the code (most likely not the machine where you run TAF) find the includes directory for MPI containing `mpif.h`. Either copy `mpif.h` to the machine where you generate the `.f` files before TAF-ing, or add the path to the includes directory to you genmake `platform` setup, TAF needs some MPI parameter settings (essentially `mpi_comm_world` and `mpi_integer`) to incorporate those in the adjoint code.

2. In `ECCO_CPPOPTIONS.h` set

```
#define ALLOW_DIVIDED_ADJOINT
#define ALLOW_DIVIDED_ADJOINT_MPI
```

   This will include the header file `mpif.h` into the top level routine for TAF.

3. Add the TAF option '`-mpi`' to the TAF argument list in the makefile.

4. Follow the same steps as in **Recipe 1** (previous section).

That's it. Good luck & have fun.

Chapter 6

Physical Parameterization and Packages

Within this chapter, the MITgcm "packages" are described. Initially, "packages" were conceived to group source code files together based upon their functionality. Each package was assigned a separate subdirectory (within `pkg`) and, usually, contained source code for implimenting different physical parametizations. This was a convenient method for both segregating and rapidly including or excluding parameterizations during the software build process.

Over time, package use has increased. The number of packages has grown and they have evolved to contain much of the model functionality including momentum schemes, I/O utilities, diagnostics, "exchange" algorithms for domain decomposition, and numerous physical parameterizations. The following sections describe how to use the existing packages and how to modify them and create new ones.

6.1 Using MITgcm Packages

The set of packages that will be used within a partiucular model can be configured using a combination of both "compile–time" and "run–time" options. Compile–time options are those used to select which packages will be "compiled in" or implemented within the program. Packages excluded at compile time are completely absent from the executable program(s) and thus cannot be later activated by any set of subsequent run–time options.

6.1.1 Package Inclusion/Exclusion

There are numerous ways that one can specify compile–time package inclusion or exclusion and they are all implemented by the `genmake2` program which was previously described in Section 3.5. The options are as follows:

1. Setting the `genamake2` options `--enable PKG` and/or `--disable PKG` specifies inclusion or exclusion. This method is intended as a convenient way to perform a single (perhaps for a quick test) compilation.

2. By creating a text file with the name `packages.conf` in either the local build directory or the `-mods=DIR` directory, one can specify a list of packages (one package per line, with '`#`' as the comment character) to be included. Since the `packages.conf` file can be saved, this is the preferred method for setting and recording (for future reference) the package configuration.

3. For convenience, a list of "standard" package groups is contained in the `pkg/pkg_groups` file. By selecting one of the package group names in the `packages.conf` file, one automatically obtains all packages in that group.

4. By default (that is, if a `packages.conf` file is not found), the `genmake2` program will use the contents of the `pkg/pkg_default` file to obtain a list of packages.

5. To help prevent users from creating unusable package groups, the `genmake2` program will parse the contents of the `pkg/pkg_depend` file to determine:

   - whether any two requested packages cannot be simultaneously included (*eg. seaice* and *thsice* are mutually exclusive),
   - whether additional packages must be included in order to satisfy package dependencies (*eg. rw* depends upon functionality within the *mdsio* package), and
   - whether the set of all requested packages is compatible with the dependencies (and producing an error if they aren't).

   Thus, as a result of the dependencies, additional packages may be added to those originally requested.

6.1.2 Package Activation

For run–time package control, MITgcm uses flags set through a `data.pkg` file. While some packages (*eg.* `debug`, `mnc`, `exch2`) may have their own usage conventions, most follow a simple flag naming convention of the form:

```
usePackageName=.TRUE.
```

where the `usePackageName` variable can activate or disable the package at runtime. As mentioned previously, packages must be included in order to be activated. Generally, such mistakes will be detected and reported as errors by the code. However, users should still be aware of the dependency.

6.2 Package Coding Standards

The following sections describe how to modify and/or create new MITgcm packages.

6.2.1 Packages are Not Libraries

To a beginner, the MITgcm packages may resemble libraries as used in myriad software projects. While future versions are likely to implement packages as libraries (perhaps using FORTRAN90/95 syntax) the current packages (FORTRAN77) are **not** based upon any concept of libraries.

6.2.1.1 File Inclusion Rules

Instead, packages should be viewed only as directories containing "sets of source files" that are built using some simple mechanisms provided by `genmake2`. Conceptually, the build process adds files as they are found and proceeds according to the following rules:

1. `genmake2` locates a "core" or main set of source files (the `-standarddirs` option sets these locations and the default value contains the directories `eesupp` and `model`).

2. `genmake2` then finds additional source files by inspecting the contents of each of the package directories:

   (a) As the new files are found, they are added to a list of source files.

   (b) If there is a file name "collision" (that is, if one of the files in a package has the same name as one of the files previously encountered) then the file within the newer (more recently visited) package will superseed (or "hide") any previous file(s) with the same name.

   (c) Packages are visited (and thus files discovered) *in the order that the packages are enabled* within `genmake2`. Thus, the files in `PackB` may superseed the files in `PackA` if `PackA` is enabled before `PackB`. Thus,

package ordering can be significant! For this reason, `genmake2` honors the order in which packages are specified.

These rules were adopted since they provide a relatively simple means for rapidly including (or "hiding") existing files with modified versions.

6.2.1.2 Conditional Compilation and `PACKAGES_CONFIG.h`

Given that packages are simply groups of files that may be added or removed to form a whole, one may wonder how linking (that is, FORTRAN symbol resolution) is handled. This is the second way that `genmake2` supports the concept of packages. Basically, `genmake2` creates a `Makefile` that, in turn, is able to create a file called `PACKAGES_CONFIG.h` that contains a set of C preprocessor (or "CPP") directives such as:

```
#undef  ALLOW_KPP
#undef  ALLOW_LAND
...
#define ALLOW_GENERIC_ADVDIFF
#define ALLOW_MDSIO
...
```

These CPP symbols are then used throughout the code to conditionally isolate variable definitions, function calls, or any other code that depends upon the presence or absence of any particular package.

An example illustrating the use of these defines is:

```
#ifdef ALLOW_GMREDI
      IF (useGMRedi) CALL GMREDI_CALC_DIFF(
     I        bi,bj,iMin,iMax,jMin,jMax,K,
     I        maskUp,
     O        KappaRT,KappaRS,
     I        myThid)
#endif
```

which is included from the file `calc_diffusivity.F` and shows how both the compile–time `ALLOW_GMREDI` flag and the run–time `useGMRedi` are nested.

There are some benefits to using the technique described here. The first is that code snippets or subroutines associated with packages can be placed or called from almost anywhere else within the code. The second benefit is related to memory footprint and performance. Since unused code can be removed, there is no performance penalty due to unnecessary memory allocation, unused function calls, or extra run-time `IF (...)` conditions. The major problems with this approach are the potentially difficult-to-read and difficult-to-debug code caused by an overuse of CPP statements. So while it can be done, developers should exerecise some discipline and avoid unnecesarily "smearing" their package implementation details across numerous files.

6.2.1.3 Package Startup or Boot Sequence

Calls to package routines within the core code timestepping loop can vary. However, all packages should follow a required "boot" sequence outlined here:

```
1. S/R PACKAGES_BOOT()
        :
    CALL OPEN_COPY_DATA_FILE( 'data.pkg', 'PACKAGES_BOOT', ... )


2. S/R PACKAGES_READPARMS()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_READPARMS( retCode )
    #endif

3. S/R PACKAGES_INIT_FIXED()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_INIT_FIXED( retCode )
    #endif

4. S/R PACKAGES_CHECK()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_CHECK( retCode )
    #else
      if ( use${Pkg} )
 &       CALL PACKAGES_CHECK_ERROR('${PKG}')
    #endif

5. S/R PACKAGES_INIT_VARIABLES()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_INIT_VARIA( )
    #endif

 6. S/R DO_THE_MODEL_IO

    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_DIAGS( )
    #endif

 7. S/R PACKAGES_WRITE_PICKUP()

    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_WRITE_PICKUP( )
    #endif
```

6.2.1.4 Adding a package to PARAMS.h and packages_boot()

An MITgcm package directory contains all the code needed for that package apart from one variable for each package. This variable is the `use${Pkg}` flag. This flag, which is of type logical, **must** be declared in the shared header file *PARAMS.h* in the *PARM_PACKAGES* block. This convention is used to support a single runtime control file *data.pkg* which is read by the startup routine *packages_boot()* and that sets a flag controlling the runtime use of a package. This routine needs to be able to read the flags for packages that were not built at compile time. Therefore when adding a new package, in addition to creating the per-package directory in the *pkg/* subdirectory a developer should add a `use${Pkg}` flag to *PARAMS.h* and a `use${Pkg}` entry to the *packages_boot() PACKAGES* namelist. The only other package specific code that should appear outside the individual package directory are calls to the specific package API.

6.3 Gent/McWiliams/Redi SGS Eddy parameterization

There are two parts to the Redi/GM parameterization of geostrophic eddies. The first aims to mix tracer properties along isentropes (neutral surfaces) by means of a diffusion operator oriented along the local isentropic surface (Redi). The second part, adiabatically re-arranges tracers through an advective flux where the advecting flow is a function of slope of the isentropic surfaces (GM).

The first GCM implementation of the Redi scheme was by Cox 1987 in the GFDL ocean circulation model. The original approach failed to distinguish between isopycnals and surfaces of locally referenced potential density (now called neutral surfaces) which are proper isentropes for the ocean. As will be discussed later, it also appears that the Cox implementation is susceptible to a computational mode. Due to this mode, the Cox scheme requires a background lateral diffusion to be present to conserve the integrity of the model fields.

The GM parameterization was then added to the GFDL code in the form of a non-divergent bolus velocity. The method defines two stream-functions expressed in terms of the isoneutral slopes subject to the boundary condition of zero value on upper and lower boundaries. The horizontal bolus velocities are then the vertical derivative of these functions. Here in lies a problem highlighted by Griffies et al., 1997: the bolus velocities involve multiple derivatives on the potential density field, which can consequently give rise to noise. Griffies et al. point out that the GM bolus fluxes can be identically written as a skew flux which involves fewer differential operators. Further, combining the skew flux formulation and Redi scheme, substantial cancellations take place to the point that the horizontal fluxes are unmodified from the lateral diffusion parameterization.

6.3.1 Redi scheme: Isopycnal diffusion

The Redi scheme diffuses tracers along isopycnals and introduces a term in the tendency (rhs) of such a tracer (here τ) of the form:

$$\nabla \cdot \kappa_\rho \mathbf{K_{Redi}} \nabla \tau \tag{6.1}$$

where κ_ρ is the along isopycnal diffusivity and $\mathbf{K_{Redi}}$ is a rank 2 tensor that projects the gradient of τ onto the isopycnal surface. The unapproximated projection tensor is:

$$\mathbf{K_{Redi}} = \begin{pmatrix} 1+S_x & S_x S_y & S_x \\ S_x S_y & 1+S_y & S_y \\ S_x & S_y & |S|^2 \end{pmatrix} \tag{6.2}$$

Here, $S_x = -\partial_x \sigma / \partial_z \sigma$ and $S_y = -\partial_y \sigma / \partial_z \sigma$ are the components of the isoneutral slope.

The first point to note is that a typical slope in the ocean interior is small, say of the order 10^{-4}. A maximum slope might be of order 10^{-2} and only

exceeds such in unstratified regions where the slope is ill defined. It is therefore justifiable, and customary, to make the small slope approximation, $|S| << 1$. The Redi projection tensor then becomes:

$$\mathbf{K_{Redi}} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ S_x & S_y & |S|^2 \end{pmatrix} \tag{6.3}$$

6.3.2 GM parameterization

The GM parameterization aims to parameterise the "advective" or "transport" effect of geostrophic eddies by means of a "bolus" velocity, $\mathbf{u}^*$. The divergence of this advective flux is added to the tracer tendency equation (on the rhs):

$$-\nabla \cdot \tau \mathbf{u}^* \tag{6.4}$$

The bolus velocity is defined as:

$$\begin{aligned} u^* &= -\partial_z F_x & (6.5)\\ v^* &= -\partial_z F_y & (6.6)\\ w^* &= \partial_x F_x + \partial_y F_y & (6.7) \end{aligned}$$

where F_x and F_y are stream-functions with boundary conditions $F_x = F_y = 0$ on upper and lower boundaries. The virtue of casting the bolus velocity in terms of these stream-functions is that they are automatically non-divergent ($\partial_x u^* + \partial_y v^* = -\partial_{xz} F_x - \partial_{yz} F_y = -\partial_z w^*$). F_x and F_y are specified in terms of the isoneutral slopes S_x and S_y:

$$\begin{aligned} F_x &= \kappa_{GM} S_x & (6.8)\\ F_y &= \kappa_{GM} S_y & (6.9) \end{aligned}$$

This is the form of the GM parameterization as applied by Donabasaglu, 1997, in MOM versions 1 and 2.

6.3.3 Griffies Skew Flux

Griffies notes that the discretisation of bolus velocities involves multiple layers of differencing and interpolation that potentially lead to noisy fields and computational modes. He pointed out that the bolus flux can be re-written in terms of a non-divergent flux and a skew-flux:

$$\begin{aligned} \mathbf{u}^*\tau &= \begin{pmatrix} -\partial_z(\kappa_{GM} S_x)\tau \\ -\partial_z(\kappa_{GM} S_y)\tau \\ (\partial_x \kappa_{GM} S_x + \partial_y \kappa_{GM} S_y)\tau \end{pmatrix} \\ &= \begin{pmatrix} -\partial_z(\kappa_{GM} S_x \tau) \\ -\partial_z(\kappa_{GM} S_y \tau) \\ \partial_x(\kappa_{GM} S_x \tau) + \partial_y(\kappa_{GM} S_y)\tau) \end{pmatrix} + \begin{pmatrix} \kappa_{GM} S_x \partial_z \tau \\ \kappa_{GM} S_y \partial_z \tau \\ -\kappa_{GM} S_x \partial_x \tau - \kappa_{GM} S_y)\partial_y \tau \end{pmatrix} \end{aligned}$$

The first vector is non-divergent and thus has no effect on the tracer field and can be dropped. The remaining flux can be written:

$$\mathbf{u}^*\tau = -\kappa_{\mathbf{GM}}\mathbf{K}_{\mathbf{GM}}\nabla\tau \tag{6.10}$$

where

$$\mathbf{K}_{\mathbf{GM}} = \begin{pmatrix} 0 & 0 & -S_x \\ 0 & 0 & -S_y \\ S_x & S_y & 0 \end{pmatrix} \tag{6.11}$$

is an anti-symmetric tensor.

This formulation of the GM parameterization involves fewer derivatives than the original and also involves only terms that already appear in the Redi mixing scheme. Indeed, a somewhat fortunate cancellation becomes apparent when we use the GM parameterization in conjunction with the Redi isoneutral mixing scheme:

$$\kappa_\rho\mathbf{K}_{\mathbf{Redi}}\nabla\tau - \mathbf{u}^*\tau = (\kappa_\rho\mathbf{K}_{\mathbf{Redi}} + \kappa_{\mathbf{GM}}\mathbf{K}_{\mathbf{GM}})\nabla\tau \tag{6.12}$$

In the instance that $\kappa_{GM} = \kappa_\rho$ then

$$\kappa_\rho\mathbf{K}_{\mathbf{Redi}} + \kappa_{\mathbf{GM}}\mathbf{K}_{\mathbf{GM}} = \kappa_\rho \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2S_x & 2S_y & |S|^2 \end{pmatrix} \tag{6.13}$$

which differs from the variable Laplacian diffusion tensor by only two non-zero elements in the z-row.

S/R GMREDI_CALC_TENSOR (*pkg/gmredi/gmredi_calc_tensor.F*)
σ_x: **SlopeX** (argument on entry)
σ_y: **SlopeY** (argument on entry)
σ_z: **SlopeY** (argument)
S_x: **SlopeX** (argument on exit)
S_y: **SlopeY** (argument on exit)

6.3.4 Variable κ_{GM}

Visbeck et al., 1996, suggest making the eddy coefficient, κ_{GM}, a function of the Eady growth rate, $|f|/\sqrt{Ri}$. The formula involves a non-dimensional constant, α, and a length-scale L:

$$\kappa_{GM} = \alpha L^2 \overline{\frac{|f|}{\sqrt{Ri}}}^z$$

where the Eady growth rate has been depth averaged (indicated by the over-line). A local Richardson number is defined $Ri = N^2/(\partial u/\partial z)^2$ which, when combined with thermal wind gives:

$$\frac{1}{Ri} = \frac{(\frac{\partial u}{\partial z})^2}{N^2} = \frac{(\frac{g}{f\rho_o}|\nabla\sigma|)^2}{N^2} = \frac{M^4}{|f|^2N^2}$$

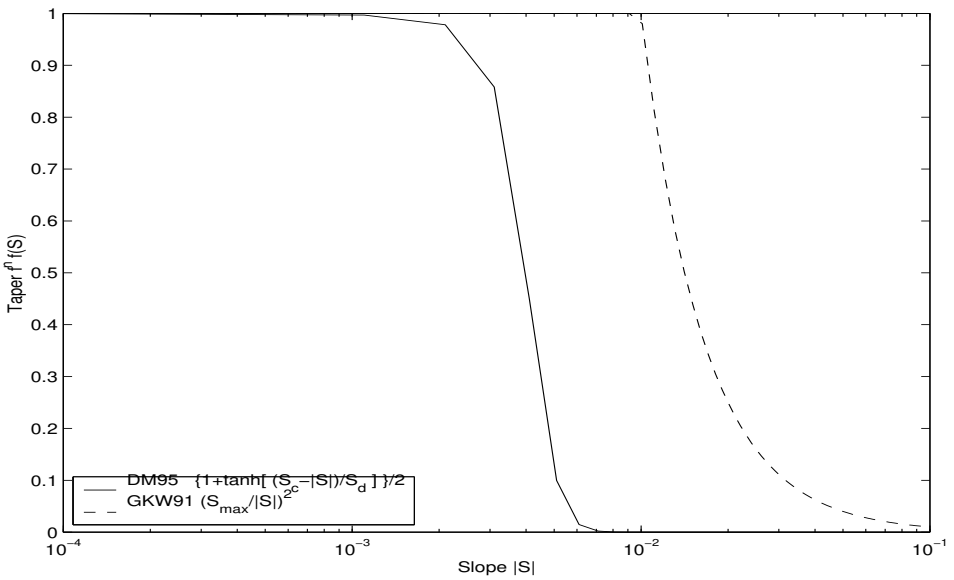


Figure 6.1: Taper functions used in GKW99 and DM95.

where M^2 is defined $M^2 = \frac{g}{\rho_o}|\nabla\sigma|$. Substituting into the formula for κ_{GM} gives:

$$\kappa_{GM} = \alpha L^2 \overline{\frac{M^2}{N}}^z = \alpha L^2 \overline{\frac{M^2}{N^2} N}^z = \alpha L^2 \overline{|S|N}^z$$

6.3.5 Tapering and stability

Experience with the GFDL model showed that the GM scheme has to be matched to the convective parameterization. This was originally expressed in connection with the introduction of the KPP boundary layer scheme (Large et al., 97) but in fact, as subsequent experience with the MIT model has found, is necessary for any convective parameterization.

S/R GMREDI_SLOPE_LIMIT (*pkg/gmredi/gmredi_slope_limit.F*)
σ_x, s_x: **SlopeX** (argument)
σ_y, s_y: **SlopeY** (argument)
σ_z: **dSigmadRReal** (argument)
z_σ^*: **dRdSigmaLtd** (argument)

6.3.5.1 Slope clipping

Deep convection sites and the mixed layer are indicated by homogenized, unstable or nearly unstable stratification. The slopes in such regions can be either infinite, very large with a sign reversal or simply very large. From a numerical

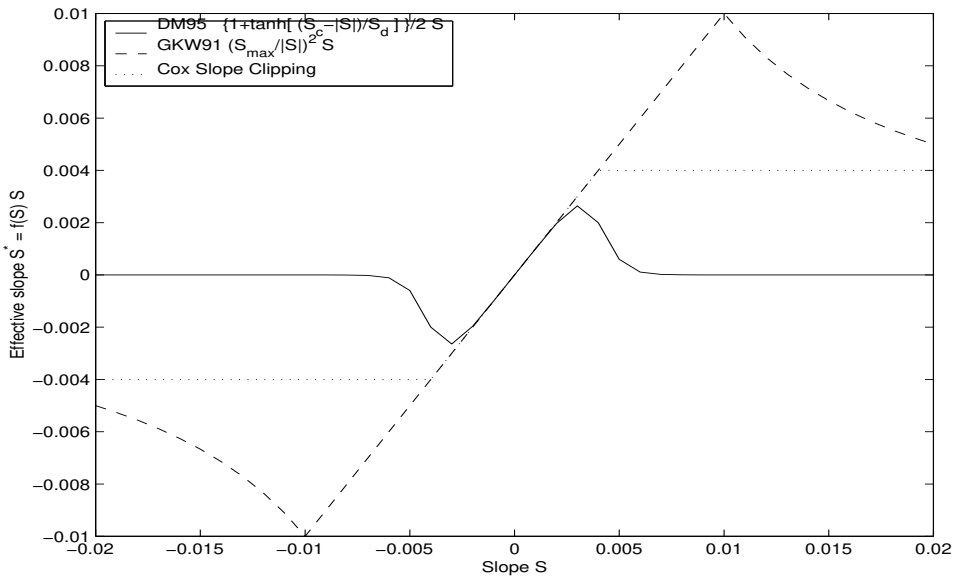


Figure 6.2: Effective slope as a function of "true" slope using Cox slope clipping, GKW91 limiting and DM95 limiting.

point of view, large slopes lead to large variations in the tensor elements (implying large bolus flow) and can be numerically unstable. This was first recognized by Cox, 1987, who implemented "slope clipping" in the isopycnal mixing tensor. Here, the slope magnitude is simply restricted by an upper limit:

$$|\nabla\sigma| = \sqrt{\sigma_x^2 + \sigma_y^2} \tag{6.14}$$

$$S_{lim} = -\frac{|\nabla\sigma|}{S_{max}} \qquad \text{where } S_{max} \text{ is a parameter} \tag{6.15}$$

$$\sigma_z^\star = \min(\sigma_z, S_{lim}) \tag{6.16}$$

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma_z^\star} \tag{6.17}$$

Notice that this algorithm assumes stable stratification through the "min" function. In the case where the fluid is well stratified ($\sigma_z < S_{lim}$) then the slopes evaluate to:

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma_z} \tag{6.18}$$

while in the limited regions ($\sigma_z > S_{lim}$) the slopes become:

$$[s_x, s_y] = \frac{[\sigma_x, \sigma_y]}{|\nabla\sigma|/S_{max}} \tag{6.19}$$

so that the slope magnitude is limited $\sqrt{s_x^2 + s_y^2} = S_{max}$.

The slope clipping scheme is activated in the model by setting **GM_taper_scheme = 'clipping'** in *data.gmredi*.

Even using slope clipping, it is normally the case that the vertical diffusion term (with coefficient $\kappa_\rho \mathbf{K}_{33} = \kappa_\rho S_{max}^2$) is large and must be time-stepped using an implicit procedure (see section on discretisation and code later). Fig. 6.3 shows the mixed layer depth resulting from a) using the GM scheme with clipping and b) no GM scheme (horizontal diffusion). The classic result of dramatically reduced mixed layers is evident. Indeed, the deep convection sites to just one or two points each and are much shallower than we might prefer. This, it turns out, is due to the over zealous re-stratification due to the bolus transport parameterization. Limiting the slopes also breaks the adiabatic nature of the GM/Redi parameterization, re-introducing diabatic fluxes in regions where the limiting is in effect.

6.3.5.2 Tapering: Gerdes, Koberle and Willebrand, Clim. Dyn. 1991

The tapering scheme used in Gerdes et al., 1999, ([42]) addressed two issues with the clipping method: the introduction of large vertical fluxes in addition to convective adjustment fluxes is avoided by tapering the GM/Redi slopes back to zero in low-stratification regions; the adjustment of slopes is replaced by a tapering of the entire GM/Redi tensor. This means the direction of fluxes is unaffected as the amplitude is scaled.

The scheme inserts a tapering function, $f_1(S)$, in front of the GM/Redi tensor:

$$f_1(S) = \min\left[1, \left(\frac{S_{max}}{|S|}\right)^2\right] \tag{6.20}$$

where S_{max} is the maximum slope you want allowed. Where the slopes, $|S| < S_{max}$ then $f_1(S) = 1$ and the tensor is un-tapered but where $|S| \geq S_{max}$ then $f_1(S)$ scales down the tensor so that the effective vertical diffusivity term $\kappa f_1(S)|S|^2 = \kappa S_{max}^2$.

The GKW tapering scheme is activated in the model by setting **GM_taper_scheme = 'gkw91'** in *data.gmredi*.

6.3.6 Tapering: Danabasoglu and McWilliams, J. Clim. 1995

The tapering scheme used by Danabasoglu and McWilliams, 1995, [11], followed a similar procedure but used a different tapering function, $f_1(S)$:

$$f_1(S) = \frac{1}{2}\left(1 + \tanh\left[\frac{S_c - |S|}{S_d}\right]\right) \tag{6.21}$$

where $S_c = 0.004$ is a cut-off slope and $S_d = 0.001$ is a scale over which the slopes are smoothly tapered. Functionally, the operates in the same way as the

Figure missing.

Figure 6.3: Mixed layer depth using GM parameterization with a) Cox slope clipping and for comparison b) using horizontal constant diffusion.

GKW91 scheme but has a substantially lower cut-off, turning off the GM/Redi SGS parameterization for weaker slopes.

The DM tapering scheme is activated in the model by setting **GM_taper_scheme = 'dm95'** in *data.gmredi*.

6.3.7 Tapering: Large, Danabasoglu and Doney, JPO 1997

The tapering used in Large et al., 1997, [35], is based on the DM95 tapering scheme, but also tapers the scheme with an additional function of height, $f_2(z)$, so that the GM/Redi SGS fluxes are reduced near the surface:

$$f_2(S) = \frac{1}{2}\left(1 + \sin(\pi\frac{z}{D} - \pi/2)\right) \tag{6.22}$$

where $D = L_\rho|S|$ is a depth-scale and $L_\rho = c/f$ with $c = 2$ m s^{-1}. This tapering with height was introduced to fix some spurious interaction with the mixed-layer KPP parameterization.

The LDD tapering scheme is activated in the model by setting **GM_taper_scheme = 'ldd97'** in *data.gmredi*.

6.3.8 Package Reference

6.4 DIC Package

6.4.1 Introduction

This is one of the biogeochemical packages handled from the pkg gchem. The main purpose of this package is to consider the cycling of carbon in the ocean. It also looks at the cycling of phosphorous and oxygen. There are five tracers DIC, ALK, $PO4$, DOP and $O2$. The air-sea exchange of CO_2 and O_2 are handled as in the OCMIP experiments (reference). The export of biological matter is computed as a function of available light and PO_4. This export is remineralized at depth according to a Martin curve (again, this is the same as in the OCMIP experiments). There is also a representation of the carbonate flux handled as in the OCMIP experiments. The air-sea exchange on CO_2 is affected by temperature, salinity and the pH of the surface waters. The pH is determined following the method of Follows et al.

6.4.2 Key subroutines and parameters

INITIALIZATION
DIC_ABIOTIC.h contains the common block for the parameters and fields needed to calculate the air-sea flux of CO_2 and O_2. The fixed parameters are set in *dic_abiotic_param* which is called from *gchem_init_fixed.F*. The parameters needed for the biotic part of the calculations are initialized in *dic_biotic_param* and are stored in *DIC_BIOTIC.h*. The first guess of pH is calculated in *dic_surfforcing_init.F*.

LOADING FIELDS
The air-sea exchange of CO_2 and O_2 need wind, atmospheric pressure (although the current version has this hardwired to 1), and sea-ice coverage. The calculation of pH needs silica fields. These fields are read in in *dic_fields_load.F*. These fields are initialized to zero in *dic_ini_forcing.F*. The fields for interpolating are in common block in *DIC_LOAD.h*.

FORCING
The tracers are advected-diffused in *ptracers_integrate.F*. The updated tracers are passed to *dic_biotic_forcing.F* where the effects of the air-sea exchange and biological activity and remineralization are calculated and the tracers are updated for a second time. Below we discuss the subroutines called from *dic_biotic_forcing.F*.

Air-sea exchange of CO_2 is calculated in *dic_surfforcing*. Air-Sea Exchange of CO_2 depends on T,S and pH. The determination of pH is done in *carbon_chem.F*. There are three subroutines in this file: *carbon_coeffs* which determines the coefficients for the carbon chemistry equations; *calc_pco2* which calculates the pH using a Newton-Raphson method; and *calc_pco2_approx* which uses the much more efficient method of Follows et al. The latter is hard-wired into this package, the former is kept here for completeness.

Biological productivity is determined following McKinely et al. and is calculated in *bio_export.F* The light in each latitude band is calculate in *insol.F*. The formation of hard tissue (carbonate) is linked to the biological productivity and has an effect on the alkalinity - the flux of carbonate is calculated in *car_flux.F*. The flux of phosphate to depth where it instantly remineralized is calculated in *phos_flux.F*.

Alkalinity tendency comes from changes to the salinity from addition/subtraction of freshwater in the surface. This is handled in *alk_surfforcing.F*.

Oxygen air-sea exchange is calculated in *o2_surfforcing.F*.

DIAGNOSTICS
Averages of air-sea exchanges, biological productivity, carbonate activity and pH are calculated. These are initialized to zero in *dic_biotic_init* and are stored in common block in *DIC_BIOTIC.h*.

6.4.3 Do's and Don'ts

This package must be run with both ptracers and gchem enabled. It is set up for 5 tracers, but there is the provision of a 6th tracer (iron) that is not discussed here.

6.4.4 Reference Material

6.5 Ocean vertical mixing – the nonlocal K-profile parameterization scheme KPP

6.6 Thermodynamic Sea Ice Package: "thsice"

Important note: This document has been written by Stephanie Dutkiewicz and describes an earlier implementation of the sea-ice package. This needs to be updated to reflect the recent changes (JMC).
This thermodynamic ice model is based on the 3-layer model by Winton (2000). and the energy-conserving LANL CICE model (Bitz and Lipscomb, 1999). The model considers two equally thick ice layers; the upper layer has a variable specific heat resulting from brine pockets, the lower layer has a fixed heat capacity. A zero heat capacity snow layer lies above the ice. Heat fluxes at the top and bottom surfaces are used to calculate the change in ice and snow layer thickness. Grid cells of the ocean model are either fully covered in ice or are open water. There is a provision to parametrize ice fraction (and leads) in this package. Modifications are discussed in small font following the subroutine descriptions.

The ice model is called from *thermodynamics.F*, subroutine *ice_forcing.F* is called in place of *external_forcing_surf.F*.

subroutine ICE_FORCING
In *ice_forcing.F*, we calculate the freezing potential of the ocean model surface layer of water:

$$\mathbf{frzmlt} = (T_f - SST)\frac{c_{sw}\rho_{sw}\Delta z}{\Delta t}$$

where c_{sw} is seawater heat capacity, ρ_{sw} is the seawater density, Δz is the ocean model upper layer thickness and Δt is the model (tracer) timestep. The freezing temperature, $T_f = \mu S$ is a function of the salinity.

1) Provided there is no ice present and **frzmlt** is less than 0, the surface tendencies of wind, heat and freshwater are calculated as usual (ie. as in *external_forcing_surf.F*).

2) If there is ice present in the grid cell we call the main ice model routine *ice_therm.F* (see below). Output from this routine gives net heat and freshwater flux affecting the top of the ocean.

Subroutine *ice_forcing.F* uses these values to find the sea surface tendencies in grid cells. When there is ice present, the surface stress tendencies are set to zero; the ice model is purely thermodynamic and the effect of ice motion on the sea-surface is not examined.

Relaxation of surface T and S is only allowed equatorward of **relaxlat** (see **DATA.ICE below**), and no relaxation is allowed under the ice at any latitude.

(Note that there is provision for allowing grid cells to have both open water and seaice; if **compact** is between 0 and 1)

subroutine ICE_FREEZE

This routine is called from *thermodynamics.F* after the new temperature calculation, *calc_gt.F*, but before *calc_gs.F*. In *ice_freeze.F*, any ocean upper layer grid cell with no ice cover, but with temperature below freezing, $T_f = \mu S$ has ice initialized. We calculate **frzmlt** from all the grid cells in the water column that have a temperature less than freezing. In this routine, any water below the surface that is below freezing is set to T_f. A call to *ice_start.F* is made if **frzmlt** > 0, and salinity tendancy is updated for brine release.

(There is a provision for fractional ice: In the case where the grid cell has less ice coverage than **icemaskmax** we allow *ice_start.F* to be called).

subroutine ICE_START

The energy available from freezing the sea surface is brought into this routine as **esurp**. The enthalpy of the 2 layers of any new ice is calculated as:

$$\begin{aligned} q_1 &= -c_i * T_f + L_i \\ q_2 &= -c_f T_{mlt} + c_i(T_{mlt} - Tf) + L_i(1 - \frac{T_{mlt}}{T_f} \end{aligned} \tag{6.23}$$

where c_f is specific heat of liquid fresh water, c_i is the specific heat of fresh ice, L_i is latent heat of freezing, ρ_i is density of ice and T_{mlt} is melting temperature of ice with salinity of 1. The height of a new layer of ice is

$$h_{inew} = \frac{\mathbf{esurp}\Delta t}{qi_{0av}}$$

where $qi_{0av} = -\frac{\rho_i}{2}(q_1 + q_2)$.

The surface skin temperature T_s and ice temperatures T_1, T_2 and the sea surface temperature are set at T_f.

(There is provision for fractional ice: new ice is formed over open water; the first freezing in the cell must have a height of **himin0**; this determines the ice fraction **compact**. If there is already ice in the grid cell, the new ice must have the same height and the new ice fraction is

$$i_f = (1 - \hat{i_f})\frac{h_{inew}}{h_i}$$

where $\hat{i_f}$ is ice fraction from previous timestep and h_i is current ice height. Snow is redistributed over the new ice fraction. The ice fraction is not allowed to become larger than **iceMaskmax** and if the ice height is above **hihig** then freezing energy comes from the full grid cell, ice growth does not occur under orginal ice due to freezing water.

subroutine ICE_THERM

The main subroutine of this package is *ice_therm.F* where the ice temperatures are calculated and the changes in ice and snow thicknesses are determined. Output provides the net heat and fresh water fluxes that force the top layer of the ocean model.

If the current ice height is less than **himin** then the ice layer is set to zero and the ocean model upper layer temperature is allowed to drop lower than

its freezing temperature; and atmospheric fluxes are allowed to effect the grid cell. If the ice height is greater than **himin** we proceed with the ice model calculation.

We follow the procedure of Winton (1999) – see equations 3 to 21 – to calculate the surface and internal ice temperatures. The surface temperature is found from the balance of the flux at the surface F_s, the shortwave heat flux absorbed by the ice, **fswint**, and the upward conduction of heat through the snow and/or ice, F_u. We linearize F_s about the surface temperature, $\hat{T_s}$, at the previous timestep (where $\hat{}$ indicates the value at the previous timestep):

$$F_s(T_s) = F_s(\hat{T_s}) + \frac{\partial F_s(\hat{T_s})}{\partial T_s}(T_s - \hat{T_s})$$

where,

$$F_s = F_{sensible} + F_{latent} + F^{down}_{longwave} + F^{up}_{longwave} + (1-\alpha)F_{shortwave}$$

and

$$\frac{dF_s}{dT} = \frac{dF_{sensible}}{dT} + \frac{dF_{latent}}{dT} + \frac{dF^{up}_{longwave}}{dT}.$$

F_s and $\frac{dF_s}{dT}$ are currently calculated from the **BULKF** package described separately, but could also be provided by an atmospheric model. The surface albedo is calculated from the ice height and/or surface temperature (see below, *srf_albedo.F*) and the shortwave flux absorbed in the ice is

$$\textbf{fswint} = (1 - e^{\kappa_i h_i})(1-\alpha)F_{shortwave}$$

where κ_i is bulk extinction coefficient.

The conductive flux to the surface is

$$F_u = K_{1/2}(T_1 - T_s)$$

where $K_{1/2}$ is the effective conductive coupling of the snow-ice layer between the surface and the mid-point of the upper layer of ice $K_{1/2} = \frac{4K_iK_s}{K_sh_i+4K_ih_s}$. K_i and K_s are constant thermal conductivities of seaice and snow.

From the above equations we can develop a system of equations to find the skin surface temperature, T_s and the two ice layer temperatures (see Winton, 1999, for details). We solve these equations iteratively until the change in T_s is small. When the surface temperature is greater then the melting temperature of the surface, the temperatures are recalculated setting T_s to 0. The enthalpy of the ice layers are calculated in order to keep track of the energy in the ice model. Enthalpy is defined, here, as the energy required to melt a unit mass of seaice with temperature T. For the upper layer (1) with brine pockets and the lower fresh layer (2):

$$\begin{aligned} q_1 &= -c_f T_f + c_i(T_f - T) + L_i(1 - \frac{T_f}{T}) \\ q_2 &= -c_i T + L_i \end{aligned}$$

where c_f is specific heat of liquid fresh water, c_i is the specific heat of fresh ice, and L_i is latent heat of melting fresh ice.

From the new ice temperatures, we can calculate the energy flux at the surface available for melting (if T_s=0) and the energy at the ocean-ice interface for either melting or freezing.

$$\begin{aligned} E_{top} &= (F_s - K_{1/2}(T_s - T_1))\Delta t \\ E_{bot} &= (\frac{4K_i(T_2 - T_f)}{h_i} - F_b)\Delta t \end{aligned}$$

where F_b is the heat flux at the ice bottom due to the sea surface temperature variations from freezing. If T_{sst} is above freezing, $F_b = c_{sw}\rho_{sw}\gamma(T_{sst} - T_f)u^*$, γ is the heat transfer coefficient and $u^* = QQ$ is frictional velocity between ice and water. If T_{sst} is below freezing, $F_b = (T_f - T_{sst})c_f\rho_f\Delta z/\Delta t$ and set T_{sst} to T_f. We also include the energy from lower layers that drop below freezing, and set those layers to T_f.

If $E_{top} > 0$ we melt snow from the surface, if all the snow is melted and there is energy left, we melt the ice. If the ice is all gone and there is still energy left, we apply the left over energy to heating the ocean model upper layer (See Winton, 1999, equations 27-29). Similarly if $E_{bot} > 0$ we melt ice from the bottom. If all the ice is melted, the snow is melted (with energy from the ocean model upper layer if necessary). If $E_{bot} < 0$ we grow ice at the bottom

$$\Delta h_i = \frac{-E_{bot}}{(q_{bot}\rho_i)}$$

where $q_{bot} = -c_i T_f + L_i$ is the enthalpy of the new ice, The enthalpy of the second ice layer, q_2 needs to be modified:

$$q_2 = \frac{\hat{h_i}/2\hat{q_2} + \Delta h_i q_{bot}}{\hat{h_i}/2 + \Delta h_i}$$

If there is a ice layer and the overlying air temperature is below 0^oC then any precipitation, P joins the snow layer:

$$\Delta h_s = -P\frac{\rho_f}{\rho_s}\Delta t,$$

ρ_f and ρ_s are the fresh water and snow densities. Any evaporation, similarly, removes snow or ice from the surface. We also calculate the snow age here, in case it is needed for the surface albedo calculation (see *srf_albedo.F* below).

For practical reasons we limit the ice growth to **hilim** and snow is limited to **hslim**. We converts any ice and/or snow above these limits back to water, maintaining the salt balance. Note however, that heat is not conserved in this conversion; sea surface temperatures below the ice are not recalculated.

If the snow/ice interface is below the waterline, snow is converted to ice (see Winton, 1999, equations 35 and 36). The subroutine *new_layers_winton.F*,

described below, repartitions the ice into equal thickness layers while conserving energy.

The subroutine *ice_therm.F* now calculates the heat and fresh water fluxes affecting the ocean model surface layer. The heat flux:

$$q_{net} = \mathbf{fswocn} - F_b - \frac{\mathbf{esurp}}{\Delta t}$$

is composed of the shortwave flux that has passed through the ice layer and is absorbed by the water, **fswocn**= QQ, the ocean flux to the ice F_b, and the surplus energy left over from the melting, **esurp**. The fresh water flux is determined from the amount of fresh water and salt in the ice/snow system before and after the timestep.

(There is a provision for fractional ice: If ice height is above **hihig** then all energy from freezing at sea surface is used only in the open water aparts of the cell (ie. F_b will only have the conduction term). The melt energy is partitioned by **frac_energy** between melting ice height and ice extent. However, once ice height drops below **himon0** then all energy melts ice extent.

subroutine SFC_ALBEDO

The routine *ice_therm.F* calls this routine to determine the surface albedo. There are two calculations provided here:
1) from LANL CICE model

$$\alpha = f_s\alpha_s + (1 - f_s)(\alpha_{i_{min}} + (\alpha_{i_{max}} - \alpha_{i_{min}})(1 - e^{-h_i/h_\alpha}))$$

where f_s is 1 if there is snow, 0 if not; the snow albedo, α_s has two values depending on whether $T_s < 0$ or not; $\alpha_{i_{min}}$ and $\alpha_{i_{max}}$ are ice albedos for thin melting ice, and thick bare ice respectively, and h_α is a scale height.
2) From GISS model (Hansen et al 1983)

$$\alpha = \alpha_i e^{-h_s/h_a} + \alpha_s(1 - e^{-h_s/h_a})$$

where α_i is a constant albedo for bare ice, h_a is a scale height and α_s is a variable snow albedo.

$$\alpha_s = \alpha_1 + \alpha_2 e^{-\lambda_a a_s}$$

where α_1 is a constant, α_2 depends on T_s, a_s is the snow age, and λ_a is a scale frequency. The snow age is calculated in *ice_therm.F* and is given in equation 41 in Hansen et al (1983).

subroutine NEW_LAYERS_WINTON

The subroutine *new_layers_winton.F* repartitions the ice into equal thickness layers while conserving energy. We pass to this subroutine, the ice layer enthalpies after melting/growth and the new height of the ice layers. The ending layer height should be half the sum of the new ice heights from *ice_therm.F*. The

enthalpies of the ice layers are adjusted accordingly to maintain total energy in the ice model. If layer 2 height is greater than layer 1 height then layer 2 gives ice to layer 1 and:

$$q_1 = f_1 \hat{q_1} + (1 - f1)\hat{q_2}$$

where f_1 is the fraction of the new to old upper layer heights. T_1 will therefore also have changed. Similarly for when ice layer height 2 is less than layer 1 height, except here we need to to be careful that the new T_2 does not fall below the melting temperature.

Initializing subroutines

ice_init.F: Set ice variables to zero, or reads in pickup information from **pickup.ic** (which was written out in *checkpoint.F*)
ice_readparms.F: Reads **data.ice**

Diagnostic subroutines

ice_ave.F: Keeps track of means of the ice variables
ice_diags.F: Finds averages and writes out diagnostics

Common Blocks

ICE.h: Ice Varibles, also **relaxlat** and **startIceModel**
ICE_DIAGS.h: matrices for diagnostics: averages of fields from *ice_diags.F*
BULKF_ICE_CONSTANTS.h (in **BULKF** package): all the parameters need by the ice model

Input file DATA.ICE

Here we need to set **StartIceModel**: which is 1 if the model starts from no ice; and 0 if there is a pickup file with the ice matrices (**pickup.ic**) which is read in *ice_init.F* and written out in *checkpoint.F*. The parameter **relaxlat** defines the latitude poleward of which there is no relaxing of surface T or S to observations. This avoids the relaxation forcing the ice model at these high latitudes.
(Note: **hicemin** is set to 0 here. If the provision for allowing grid cells to have both open water and seaice is ever implemented, this would be greater than 0)

Important Notes

1) heat fluxes have different signs in the ocean and ice models.
2) StartIceModel must be changed in **data.ice**: 1 (if starting from no ice), 0 (if using pickup.ic file).

References

Bitz, C.M. and W.H. Lipscombe, 1999: An Energy-Conserving Thermodynamic Model of Sea Ice. *Journal of Geophysical Research*, 104, 15,669 – 15,677.

Hansen, J., G. Russell, D. Rind, P. Stone, A. Lacis, S. Lebedeff, R. Ruedy and L.Travis, 1983: Efficient Three-Dimensional Global Models for Climate Studies: Models I and II. *Monthly Weather Review*, 111, 609 – 662.

Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual. LACC-98-16v.2.
(note: this documentation is no longer available as CICE has progressed to a very different version 3)

Winton, M, 2000: A reformulated Three-layer Sea Ice Model. *Journal of Atmospheric and Ocean Technology*, 17, 525 – 531.

6.7 Sea Ice Package: "seaice"

Package "seaice" provides a dynamic and thermodynamic interactive sea-ice model. Sea-ice model thermodynamics are based on Hibler [28], that is, a 2-category model that simulates ice thickness and concentration. Snow is simulated as per Zhang et al. [54]. Although recent years have seen an increased use of multi-category thickness distribution sea-ice models for climate studies, the Hibler 2-category ice model is still the most widely used model and has resulted in realistic simulation of sea-ice variability on regional and global scales. Being less complicated, compared to multi-category models, the 2-category model permits easier application of adjoint model optimization methods.

Note, however, that the Hibler 2-category model and its variants use a so-called zero-layer thermodynamic model to estimate ice growth and decay. The zero-layer thermodynamic model assumes that ice does not store heat and, therefore, tends to exaggerate the seasonal variability in ice thickness. This exaggeration can be significantly reduced by using Semtner's [46] three-layer thermodynamic model that permits heat storage in ice. Recently, the three-layer thermodynamic model has been reformulated by Winton [53]. The reformulation improves model physics by representing the brine content of the upper ice with a variable heat capacity. It also improves model numerics and consumes less computer time and memory. The Winton sea-ice thermodynamics have been ported to the MIT GCM; they currently reside under pkg/thsice. At present pkg/thsice is not fully compatible with pkg/seaice and with pkg/exf. But the eventual objective is to have fully compatible and interchangeable thermodynamic packages for sea-ice, so that it becomes possible to use Hibler dynamics with Winton thermodyanmics.

The ice dynamics models that are most widely used for large-scale climate studies are the viscous-plastic (VP) model [27], the cavitating fluid (CF) model [14], and the elastic-viscous-plastic (EVP) model [32]. Compared to the VP model, the CF model does not allow ice shear in calculating ice motion, stress, and deformation. EVP models approximate VP by adding an elastic term to the equations for easier adaptation to parallel computers. Because of its higher accuracy in plastic solution and relatively simpler formulation, compared to the EVP model, we decided to use the VP model as the dynamic component of our ice model. To do this we extended the alternating-direction-implicit (ADI) method of Zhang and Rothrock [55] for use in a parallel configuration.

The sea ice model requires the following input fields: 10-m winds, 2-m air temperature and specific humidity, downward longwave and shortwave radiations, precipitation, evaporation, and river and glacier runoff. The sea ice model also requires surface temperature from the ocean model and third level horizontal velocity which is used as a proxy for surface geostrophic velocity. Output fields are surface wind stress, evaporation minus precipitation minus runoff, net surface heat flux, and net shortwave flux. The sea-ice model is global: in ice-free regions bulk formulae are used to estimate oceanic forcing from the atmospheric fields.

6.8 Bulk Formula Package

author: Stephanie Dutkiewicz

Instead of forcing the model with heat and fresh water flux data, this package calculates these fluxes using the changing sea surface temperature. We need to read in some atmospheric data: **air temperature, air humidity, down shortwave radiation, down longwave radiation, precipitation, wind speed**. The current setup also reads in **wind stress**, but this can be changed so that the stresses are calculated from the wind speed.

The current setup requires that there is the thermodynamic-seaice package (*pkg/thsice*, also refered below as seaice) is also used. It would be useful though to have it also setup to run with some very simple parametrization of the sea ice.

The heat and fresh water fluxes are calculated in *bulkf_forcing.F* called from *forward_step.F*. These fluxes are used over open water, fluxes over seaice are recalculated in the sea-ice package. Before the call to *bulkf_forcing.F* we call *bulkf_fields_load.F* to find the current atmospheric conditions. The only other changes to the model code come from the initializing and writing diagnostics of these fluxes.

subroutine BULKF_FIELDS_LOAD

Here we find the atmospheric data needed for the bulk formula calculations. These are read in at periodic intervals and values are interpolated to the current time. The data file names come from **data.blk**. The values that can be read in are: air temperature, air humidity, precipitation, down solar radiation, down long wave radiation, zonal and meridional wind speeds, total wind speed, net heat flux, net freshwater forcing, cloud cover, snow fall, zonal and meridional wind stresses, and SST and SSS used for relaxation terms. Not all these files are necessary or used. For instance cloud cover and snow fall are not used in the current bulk formula calculation. If total wind speed is not supplied, wind speed is calculate from the zonal and meridional components. If wind stresses are not read in, then the stresses are calculated from the wind speed. Net heat flux and net freshwater can be read in and used over open ocean instead of the bulk formula calculations (but over seaice the bulkf formula is always used). This is "hardwired" into *bulkf_forcing* and the "ch" in the variable names suggests that this is "cheating". SST and SSS need to be read in if there is any relaxation used.

subroutine BULKF_FORCING

In *bulkf_forcing.F*, we calculate heat and fresh water fluxes (and wind stress, if

necessary) for each grid cell. First we determine if the grid cell is open water or seaice and this information is carried by **iceornot**. There is a provision here for a different designation if there is snow cover (but currently this does not make any difference). We then call *bulkf_formula_lanl.F* which provides values for: up long wave radiation, latent and sensible heat fluxes, the derivative of these three with respect to surface temperature, wind stress, evaporation. Net long wave radiation is calculated from the combination of the down long wave read in and the up long wave calculated.

We then find the albedo of the surface - with a call to *sfc_albedo* if there is sea-ice (see the seaice package for information on the subroutine). If the grid cell is open ocean the albedo is set as 0.1. Note that this is a parameter that can be used to tune the results. The net short wave radiation is then the down shortwave radiation minus the amount reflected.

If the wind stress needed to be calculated in *bulkf_formula_lanl.F*, it was calculated to grid cell center points, so in *bulkf_forcing.F* we regrid to **u** and **v** points. We let the model know if it has read in stresses or calculated stresses by the switch **readwindstress** which is can be set in data.blk, and defaults to **.TRUE.**.

We then calculate **Qnet** and **EmPmR** that will be used as the fluxes over the open ocean. There is a provision for using runoff. If we are "cheating" and using observed fluxes over the open ocean, then there is a provision here to use read in **Qnet** and **EmPmR**.

The final call is to calculate averages of the terms found in this subroutine.

subroutine BULKF_FORMULA_LANL

This is the main program of the package where the heat fluxes and freshwater fluxes over ice and open water are calculated. Note that this subroutine is also called from the seaice package during the iterations to find the ice surface temperature.

Latent heat (L) used in this subroutine depends on the state of the surface: vaporization for open water, fusion and vaporization for ice surfaces. Air temperature is converted from Celsius to Kelvin. If there is no wind speed (u_s) given, then the wind speed is calculated from the zonal and meridional components.

We calculate the virtual temperature:

$$T_o = T_{air}(1 + \gamma q_{air})$$

where T_{air} is the air temperature at h_T, q_{air} is humidity at h_q and γ is a constant.

The saturated vapor pressure is calculate (QQ ref):

$$q_{sat} = \frac{a}{p_o} e^{L(b - \frac{c}{T_{srf}})}$$

where a, b, c are constants, T_{srf} is surface temperature and p_o is the surface pressure.

The two values crucial for the bulk formula calculations are the difference between air at sea surface and sea surface temperature:

$$\Delta T = T_{air} - T_{srf} + \alpha h_T$$

where α is adiabatic lapse rate and h_T is the height where the air temperature was taken; and the difference between the air humidity and the saturated humidity

$$\Delta q = q_{air} - q_{sat}.$$

We then calculate the turbulent exchange coefficients following Bryan et al (1996) and the numerical scheme of Hunke and Lipscombe (1998). We estimate initial values for the exchange coefficients, c_u, c_T and c_q as

$$\frac{\kappa}{ln(z_{ref}/z_{rou})}$$

where κ is the Von Karman constant, z_{ref} is a reference height and z_{rou} is a roughness length scale which could be a function of type of surface, but is here set as a constant. Turbulent scales are:

$$\begin{aligned} u^* &= c_u u_s \\ T^* &= c_T \Delta T \\ q^* &= c_q \Delta q \end{aligned}$$

We find the "integrated flux profile" for momentum and stability if there are stable QQ conditions ($\Upsilon > 0$) :

$$\psi_m = \psi_s = -5\Upsilon$$

and for unstable QQ conditions ($\Upsilon < 0$):

$$\begin{aligned} \psi_m &= 2ln(0.5(1+\chi)) + ln(0.5(1+\chi^2)) - 2\tan^{-1}\chi + \pi/2 \\ \psi_s &= 2ln(0.5(1+\chi^2)) \end{aligned}$$

where

$$\Upsilon = \frac{\kappa g z_{ref}}{u^{*2}}\left(\frac{T^*}{T_o} + \frac{q^*}{1/\gamma + q_a}\right)$$

and $\chi = (1 - 16\Upsilon)^{1/2}$.

The coefficients are updated through 5 iterations as:

$$\begin{aligned} c_u &= \frac{\hat{c_u}}{1 + \hat{c_u}(\lambda - \psi_m)/\kappa} \\ c_T &= \frac{\hat{c_T}}{1 + \hat{c_T}(\lambda - \psi_s)/\kappa} \\ c_q &= c'_T \end{aligned} \tag{6.24}$$

where $\lambda = ln(h_T/z_{ref})$.

We can then find the bulk formula heat fluxes:

Sensible heat flux:

$$Q_s = \rho_{air} c_{p_{air}} u_s c_u c_T \Delta T$$

Latent heat flux:

$$Q_l = \rho_{air} L u_s c_u c_q \Delta q$$

Up long wave radiation

$$Q_{lw}^{up} = \epsilon \sigma T_{srf}^4$$

where ϵ is emissivity (which can be different for open ocean, ice and snow), σ is Stefan-Boltzman constant.

We calculate the derivatives of the three above functions with respect to surface temperature

$$\begin{aligned} \frac{dQ_s}{d_T} &= \rho_{air} c_{p_{air}} u_s c_u c_T \\ \frac{dQ_l}{d_T} &= \frac{\rho_{air} L^2 u_s c_u c_q c}{T_{srf}^2} \\ \frac{dQ_{]lw}^{up}}{d_T} &= 4\epsilon\sigma t_{srf}^3 \end{aligned}$$

And total derivative $\frac{dQ_o}{dT} = \frac{dQ_s}{dT} + \frac{dQ_l}{dT} + \frac{dQ_{lw}^{up}}{dT}$.

If we do not read in the wind stress, it is calculated here.

Initializing subroutines

bulkf_init.F: Set bulkf variables to zero.
bulkf_readparms.F: Reads **data.blk**

Diagnostic subroutines

bulkf_ave.F: Keeps track of means of the bulkf variables
bulkf_diags.F: Finds averages and writes out diagnostics

Common Blocks

BULKF.h: BULKF Variables, data file names, and logicals **readwindstress** and **readsurface**
BULKF_DIAGS.h: matrices for diagnostics: averages of fields from *bulkf_diags.F*
BULKF_ICE_CONSTANTS.h: all the parameters need by the ice model and in the bulkf formula calculations.

Input file DATA.ICE

We read in the file names of atmospheric data used in the bulk formula calculations. Here we can also set the logicals: **readwindstress** if we read in the wind stress rather than calculate it from the wind speed; and **readsurface** to read in the surface temperature and salinity if these will be used as part of a relaxing term.

Important Notes

1) heat fluxes have different signs in the ocean and ice models.
2) StartIceModel must be changed in **data.ice**: 1 (if starting from no ice), 0 (if using pickup.ic file).

References

Bryan F.O., B.G Kauffman, W.G. Large, P.R. Gent, 1996: The NCAR CSM flux coupler. Technical note TN-425+STR, NCAR.

Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual. LACC-98-16v.2.
(note: this documentation is no longer available as CICE has progressed to a very different version 3)

6.9 Generic Advection/Diffusion

The `generic_advdiff` package contains high-level subroutines to solve the advection-diffusion equation of any tracer, either active (potential temperature, salinity or water vapor) or passive (see pkg/ptracers). (see also sections 2.15 to 2.18).

6.9.1 Introduction

Package `generic_advdiff` is a...

6.9.2 Key subroutines, parameters and files

The `generic_advdiff` package has...

6.10 Atmospheric Intermediate Physics: AIM

Note: *kpg/aim* corresponds to an old version of the AIM physics (release.1). The folowing document below describes the more recent *aim_v23* package that is based on the version v23 of the SPEEDY code ([51]).

6.10.1 Key subroutines, parameters and files

6.11 Land package

This package provides a simple land model based on Rong Zhang [e-mail:roz@gfdl.noaa.gov] 2 layers model (see documentation below).

It is primarily implemented for AIM (_v23) atmospheric physics but could be adapted to work with a different atmospheric physics. Two subroutines (*aim_aim2land.F aim_land2aim.F* in *pkg/aim_v23*) are used as interface with AIM physics.

Number of layers is a parameter (*land_nLev* in *LAND_SIZE.h*) and can be changed.

Note on Land Model
date: June 1999
author: Rong Zhang

This is a simple 2-layer land model. The top layer depth $z1 = 0.1m$, the second layer depth $z2 = 4m$.

Let T_{g1}, T_{g2} be the temperature of each layer, W_1,W_2 be the soil moisture of each layer. The field capacity f_1, f_2 are the maximum water amount in each layer, so W_i is the ratio of available water to field capacity. $f_i = \gamma z_i, \gamma = 0.24$ is the field capapcity per meter soil, so $f_1 = 0.024m$, $f_2 = 0.96m$.

The land temperature is determined by total surface downward heat flux F,

$$z_1 C_1 \frac{dT_{g1}}{dt} = F - \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2} \tag{6.25}$$

$$z_2 C_2 \frac{dT_{g2}}{dt} = \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2} \tag{6.26}$$

here C_1, C_2 are the heat capacity of each layer , λ is the thermal conductivity, $\lambda = 0.42 W m^{-1} K^{-1}$.

$$C_1 = C_w W_1 \gamma + C_s \tag{6.27}$$

$$C_2 = C_w W_2 \gamma + C_s \tag{6.28}$$

C_w, C_s are the heat capacity of water and dry soil respectively. $C_w = 4.2 \times 10^6 J m^{-3} K^{-1}, C_s = 1.13 \times 10^6 J m^{-3} K^{-1}$.

The soil moisture is determined by precipitation $P(m/s)$,surface evaporation $E(m/s)$ and runoff $R(m/s)$.

$$\frac{dW_1}{dt} = \frac{P - E - R}{f_1} + \frac{W_2 - W_1}{\tau} \tag{6.29}$$

$\tau = 2$ *days* is the time constant for diffusion of moisture between layers.

$$\frac{dW_2}{dt} = \frac{f_1}{f_2}\frac{W_1 - W_2}{\tau} \tag{6.30}$$

In the code, $R = 0$ gives better result, W_1, W_2 are set to be within [0, 1]. If W_1 is greater than 1, then let $\delta W_1 = W_1 - 1, W_1 = 1$ and $W_2 = W_2 + p\delta W_1 \frac{f_1}{f_2}$, i.e. the runoff of top layer is put into second layer. $p = 0.5$ is the fraction of top layer runoff that is put into second layer.

The time step is 1 hour, it takes several years to reach equalibrium offline.

References

Hansen J. et al. Efficient three-dimensional global models for climate studies: models I and II. *Monthly Weather Review*, vol.111, no.4, pp. 609-62, 1983

6.12 Coupling interface for Atmospheric Intermediate code

6.12.1 Key subroutines, parameters and files

6.13 Coupler for mapping between AIM and ocean

6.13.1 Key subroutines, parameters and files

6.14 Toolkit for building couplers

6.14.1 Key subroutines, parameters and files

6.15 NetCDF I/O Integration: MNC

The `mnc` package is a set of convenience routines written to expedite the process of creating, appending, and reading NetCDF files. NetCDF is an increasingly popular self-describing file format [44] intended primarily for scientific data sets. An extensive collection of NetCDF reference papers, user guides, software, FAQs, and other information can be obtained from UCAR's web site at:

`http://www.unidata.ucar.edu/packages/netcdf/`

6.15.1 Using MNC

6.15.1.1 MNC Configuration

As with all MITgcm packages, MNC can be turned on or off at compile time using the `packages.conf` file or the `genmake2 -enable=mnc` or `-disable=mnc` switches.

While MNC is likely to work "as is", there are a few compile–time constants that may need to be increased for simulations that employ large numbers of tiles within each process. Note that the important quantity is the maximum number of tiles **per process**. Since MPI configurations tend to distribute large numbers of tiles over relatively large numbers of MPI processes, these constants will rarely need to be increased.

If MNC runs out of space within its "lookup" tables during a simulation, then it will provide an error message along with a recommendation of which parameter to increase. The parameters are all located within `pkg/mnc/mnc_common.h` and the ones that may need to be increased are:

Name	Default	Description
MNC_MAX_ID	1000	**IDs for various low-level entities**
MNC_MAX_INFO	400	**IDs (mostly for object sizes)**
MNC_CW_MAX_I	150	**IDs for the "wrapper" layer**

In those rare cases where MNC "out-of-memory" error messages are encountered, it is a good idea to increase the too-small parameter by a factor of **2–10** in order to avoid wasting time on an iterative compile–test sequence.

6.15.1.2 MNC Inputs

For run-time configuration, most of the MNC–related model parameters are contained within a Fortran namelist file called `data.mnc`. If this file does not exist, then the MNC package will interpret that as an indication that it is not to be used. If the `data.mnc` file does exist, then it may contain the following parameters:

Name	T	Default	Description
useMNC	L	.FALSE.	**overall MNC ON/OFF switch**
mnc_echo_gvtypes	L	.FALSE.	echo pre-defined "types" (debugging)
mnc_use_outdir	L	.FALSE.	create a directory for output
mnc_outdir_str	S	'mnc_'	output directory name
mnc_outdir_date	L	.FALSE.	embed date in the output dir name
pickup_write_mnc	L	.FALSE.	use MNC to write (create) pickup files
pickup_read_mnc	L	.FALSE.	use MNC to read pickup files
mnc_use_indir	L	.FALSE.	use a directory (path) for input
mnc_indir_str	S	''	input directory (or path) name
snapshot_mnc	L	.FALSE.	write snapshot (instantaneous) w/MNC
monitor_mnc	L	.FALSE.	write monitor w/MNC
timeave_mnc	L	.FALSE.	write timeave w/MNC
autodiff_mnc	L	.FALSE.	write autodiff w/MNC

Additional MNC–related parameters are contained within the main `data` namelist file and in some of the namelist files for individual packages. These options are:

Name	T	Default	Description
Main namelist file: "**data**"			
snapshot_ioinc	L	.FALSE.	write snapshot "inclusively"
timeave_ioinc	L	.FALSE.	write timeave "inclusively"
monitor_ioinc	L	.FALSE.	write monitor "inclusively"
the_run_name	C	"name..."	name is included in all MNC output
Diagnostics namelist file: "**data.diagnostics**"			
diag_mnc	L	.FALSE.	write diagnostics w/MNC
diag_ioinc	L	.FALSE.	write diagnostics "inclusively"

By default, turning on MNC for a particular output type will result in turning off all the corresponding (usually, default) MDSIO or STDOUT output mechanisms. In other words, output defaults to being an exclusive selection. To enable multiple kinds of simultaneous output, flags of the form `NAME_ioinc` have been created where `NAME` corresponds to the various MNC output flags. When a `NAME_ioinc` flag is set to `.TRUE.`, then multiple simultaneous forms of output are allowed for the `NAME` output mechanism. The intent of this design is that typical users will only want one kind of output while people debugging the code (particularly the I/O routines) may want simultaneous types of output.

This "inclusive" versus "exclusive" design is easily applied in cases where three or more kinds of output may be generated. Thus, it can be readily extended to additional new output types (eg. HDF5).

Input types are always exclusive.

6.15.1.3 MNC Output

While NetCDF files are supposed to be "self-describing", it is helpful to note the following:

- The constraints placed upon the "unlimited" (or "record") dimension inherent with NetCDF v3.x make it very inefficient to put variables written

at potentially different intervals within the same file. For this reason, MNC output is split into a few file "base names" which try to reflect the nature of their content.

- All MNC output is currently done in a "tile-per-file" fashion since most NetCDF v3.x implementions cannot write safely within MPI or multi-threaded environments. This tiling is done in a global fashion and the tile numbers are appended to the base names described above. Some scripts to "assemble" output are available (`MITgcm/utils/matlab`). More general manipulations can be accomplished with the

  ```
  NetCDF Operators (or ‘‘NCO’’) at http://nco.sourceforge.net
  ```

  which is a very powerful and convenient set of tools for working with all NetCDF files.

- On many systems, NetCDF has practical file size limits on the order of 2–4GB (the maximium memory addressable with 32bit pointers) due to a lack of operating system, compiler, and/or library support. In cases where this limit is reached, it is generally a good idea to reduce write frequencies or restart from pickups.

- MNC does not (yet) provide a mechanism for reading information from a single "global" file as can be done with the MDSIO package. This is in progress.

6.15.2 MNC Internals

The `mnc` package is a two-level convenience library (or "wrapper") for most of the NetCDF Fortran API. Its purpose is to streamline the user interface to NetCDF by maintaining internal relations (look-up tables) keyed with strings (or names) and entities such as NetCDF files, variables, and attributes.

The two levels of the `mnc` package are:

Upper level

The upper level contains information about two kinds of associations:

grid type is lookup table indexed with a grid type name. Each grid type name is associated with a number of dimensions, the dimension sizes (one of which may be unlimited), and starting and ending index arrays. The intent is to store all the necessary size and shape information for the Fortran arrays containing MITgcm–style "tile" variables (that is, a central region surrounded by a variably-sized "halo" or exchange region as shown in Figures 4.7 and 4.8).

variable type is a lookup table indexed by a variable type name. For each name, the table contains a reference to a grid type for the variable and the names and values of various attributes.

Within the upper level, these associations are not permanently tied to any particular NetCDF file. This allows the information to be re-used over multiple file reads and writes.

Lower level

In the lower (or internal) level, associations are stored for NetCDF files and many of the entities that they contain including dimensions, variables, and global attributes. All associations are on a per-file basis. Thus, each entity is tied to a unique NetCDF file and will be created or destroyed when files are, respectively, opened or closed.

6.15.2.1 MNC Grid–Types and Variable–Types

As a convenience for users, the MNC package includes numerous routines to aid in the writing of data to NetCDF format. Probably the biggest convenience is the use of pre-defined "grid types" and "variable types". These "types" are simply look-up tables that store dimensions, indicies, attributes, and other information that can all be retrieved using a single character string.

The "grid types" are a way of mapping variables within MITgcm to NetCDF arrays. Within MITgcm, most spatial variables are defined using two– or three–dimensional arrays with "overlap" regions (see Figures 4.7, a possible vertical index, and 4.8) and tile indicies such as the following "U" velocity:

```
_RL  uVel (1-OLx:sNx+OLx,1-OLy:sNy+OLy,Nr,nSx,nSy)
```

as defined in model/inc/DYNVARS.h

The grid type is a character string that encodes the presence and types associated with the four possible dimensions. The character string follows the format

```
H0_H1_H2_V_T
```

where the terms *H0*, *H1*, *H2*, *V*, *T* can be almost any combination of the following:

Horizontal			Vertical	Time
H0: location	**H1**: dimensions	**H2**: halo	**V**: location	**T**: level
-	xy	Hn	-	-
U	x	Hy	i	t
V	y		c	
Cen				
Cor				

A example list of all pre-defined combinations is contained in the file

```
pkg/mnc/pre-defined_grids.txt.
```

The variable type is an association between a variable type name and the following items:

Item	Purpose
grid type	defines the in-memory arrangement
bi,bj dimensions	tiling indices, if present

and is used by the mnc_cw_*_[R|W] subroutines for reading and writing variables.

6.15.2.2 Using MNC: Examples

Writing variables to NetCDF files can be accomplished in as few as two function calls. The first function call defines a variable type, associates it with a name (character string), and provides additional information about the indicies for the tile (bi,bj) dimensions. The second function call will write the data at, if necessary, the current time level within the model.

Examples of the initialization calls can be found in the file model/src/ini_mnc_io.F where these function calls:

```
C     Create MNC definitions for DYNVARS.h variables
      CALL MNC_CW_ADD_VNAME('iter', '-_-_--__-__t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('iter',1,
     &     'long_name','iteration_count', myThid)

      CALL MNC_CW_ADD_VNAME('model_time', '-_-_--__-__t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,
     &     'long_name','Model Time', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,'units','s', myThid)

      CALL MNC_CW_ADD_VNAME('U', 'U_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,'units','m/s', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,
     &     'coordinates','XU YU RC iter', myThid)

      CALL MNC_CW_ADD_VNAME('T', 'Cen_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'units','degC', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'long_name',
     &     'potential_temperature', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,
     &     'coordinates','XC YC RC iter', myThid)
```

initialize four VNAMEs and add one or more NetCDF attributes to each.

The four variables defined above are subsequently written at specific time steps within model/src/write_state.F using the function calls:

```
C       Write dynvars using the MNC package
        CALL MNC_CW_SET_UDIM('state', -1, myThid)
        CALL MNC_CW_I_W('I','state',0,0,'iter', myIter, myThid)
        CALL MNC_CW_SET_UDIM('state', 0, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'model_time',myTime, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'U', uVel, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'T', theta, myThid)
```

While it is easiest to write variables within typical 2D and 3D fields where all data is known at a given time, it is also possible to write fields where only a portion (*eg.* a "slab" or "slice") is known at a given instant. An example is provided within pkg/mom_vecinv/mom_vecinv.F where an offset vector is used:

```
      IF (useMNC .AND. snapshot_mnc) THEN
        CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fV', uCf,
     &          offsets, myThid)
        CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fU', vCf,
     &          offsets, myThid)
      ENDIF
```

to write a 3D field one depth slice at a time.

Each element in the offset vector corresponds (in order) to the dimensions of the "full" (or virtual) array and specifies which are known at the time of the call. A zero within the offset array means that all values along that dimension are available while a positive integer means that only values along that index of the dimension are available. In all cases, the matrix passed is assumed to start (that is, have an in-memory structure) coinciding with the start of the specified slice. Thus, using this offset array mechanism, a slice can be written along any single dimension or combinations of dimensions.

6.16 MDSIO

The `mdsio` package contains a group of Fortran routines intended as a general interface for reading and writing direct-access ("binary") Fortran files. The `mdsio` routines are used by the `rw` package.

6.17 Simulation state monitoring toolkit

6.17.1 Key subroutines, parameters and files

6.18 exch2: Extended Cubed Sphere Topology

6.18.1 Introduction

The exch2 package extends the original cubed sphere topology configuration to allow more flexible domain decomposition and parallelization. Cube faces (also called subdomains) may be divided into any number of tiles that divide evenly into the grid point dimensions of the subdomain. Furthermore, the tiles can run on separate processors individually or in groups, which provides for manual compile-time load balancing across a relatively arbitrary number of processors.

The exchange parameters are declared in pkg/exch2/W2_EXCH2_TOPOLOGY.h and assigned in pkg/exch2/w2_e2setup.F. The validity of the cube topology depends on the SIZE.h file as detailed below. The default files provided in the release configure a cubed sphere topology of six tiles, one per subdomain, each with 32×32 grid points, with all tiles running on a single processor. Both files are generated by Matlab scripts in utils/exch2/matlab-topology-generator; see Section 6.18.3 *Generating Topology Files for exch2* for details on creating alternate topologies. Pregenerated examples of these files with alternate topologies are provided under utils/exch2/code-mods along with the appropriate SIZE.h file for single-processor execution.

6.18.2 Invoking exch2

To use exch2 with the cubed sphere, the following conditions must be met:

• The exch2 package is included when genmake2 is run. The easiest way to do this is to add the line exch2 to the profile.conf file – see Section 3.5 *Building the code* for general details.

• An example of W2_EXCH2_TOPOLOGY.h and w2_e2setup.F must reside in a directory containing files symbolically linked by the genmake2 script. The safest place to put these is the directory indicated in the -mods=DIR command line modifier (typically ../code), or the build directory. The default versions of these files reside in pkg/exch2 and are linked automatically if no other versions exist elsewhere in the build path, but they should be left untouched to avoid breaking configurations other than the one you intend to modify.

• Files containing grid parameters, named tile00n.mitgrid where n=(1:6) (one per subdomain), must be in the working directory when the MITgcm executable is run. These files are provided in the example experiments for cubed sphere configurations with 32×32 cube sides – please contact MITgcm support if you want to generate files for other configurations.

• As always when compiling MITgcm, the file SIZE.h must be placed where genmake2 will find it. In particular for exch2, the domain decomposition spec-

ified in `SIZE.h` must correspond with the particular configuration's topology specified in `W2_EXCH2_TOPOLOGY.h` and `w2_e2setup.F`. Domain decomposition issues particular to exch2 are addressed in Section 6.18.3 *Generating Topology Files for exch2* and 6.18.4 *exch2, SIZE.h, and Multiprocessing*; a more general background on the subject relevant to MITgcm is presented in Section 4.3.1 *Specifying a decomposition*.

At the time of this writing the following examples use exch2 and may be used for guidance:

```
verification/adjust_nlfs.cs-32x32x1
verification/adjustment.cs-32x32x1
verification/aim.5l_cs
verification/global_ocean.cs32x15
verification/hs94.cs-32x32x5
```

6.18.3 Generating Topology Files for exch2

Alternate cubed sphere topologies may be created using the Matlab scripts in `utils/exch2/matlab-topology-generator`. Running the m-file `driver.m` from the Matlab prompt (there are no parameters to pass) generates exch2 topology files `W2_EXCH2_TOPOLOGY.h` and `w2_e2setup.F` in the working directory and displays a figure of the topology via Matlab – figures 6.6, 6.5, and 6.4 are examples of the generated diagrams. The other m-files in the directory are subroutines called from `driver.m` and should not be run "bare" except for development purposes.

The parameters that determine the dimensions and topology of the generated configuration are `nr`, `nb`, `ng`, `tnx` and `tny`, and all are assigned early in the script.

The first three determine the height and width of the subdomains and hence the size of the overall domain. Each one determines the number of grid points, and therefore the resolution, along the subdomain sides in a "great circle" around each the three spatial axes of the cube. At the time of this writing MITgcm requires these three parameters to be equal, but they provide for future releases to accomodate different resolutions around the axes to allow subdomains with differing resolutions.

The parameters `tnx` and `tny` determine the width and height of the tiles into which the subdomains are decomposed, and must evenly divide the integer assigned to `nr`, `nb` and `ng`. The result is a rectangular tiling of the subdomain. Figure 6.4 shows one possible topology for a twenty-four-tile cube, and figure 6.5 shows one for twelve tiles.

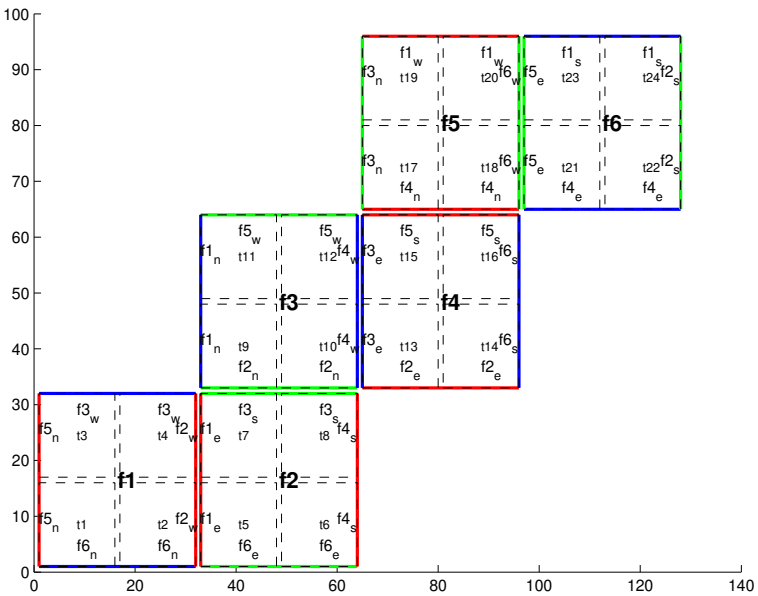


Figure 6.4: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains, each of which is divided into four tiles of width `tnx=16` and height `tny=16` for a total of twenty-four tiles. The colored borders of the subdomains represent the parameters `nr` (red), `nb` (blue), and `ng` (green).

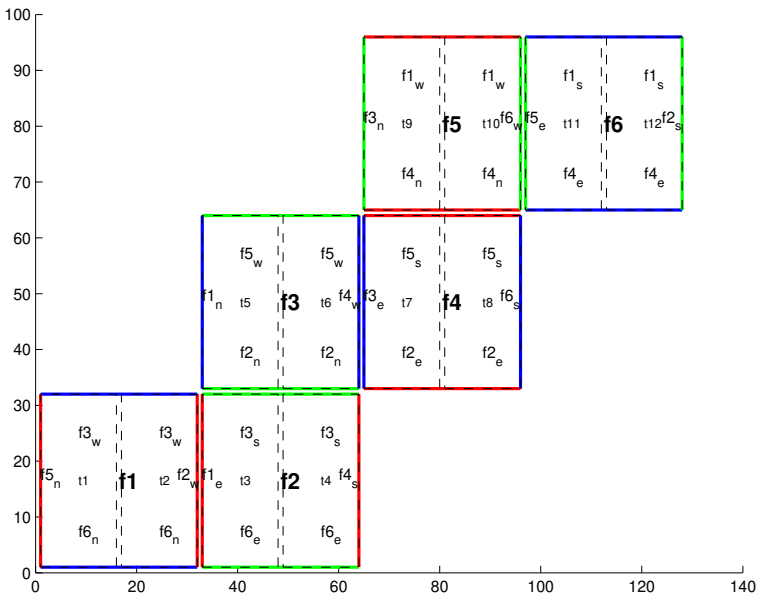


Figure 6.5: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains of two tiles each (`tnx=16, tny=32`).

Tiles can be selected from the topology to be omitted from being allocated memory and processors. This tuning is useful in ocean modeling for omitting tiles that fall entirely on land. The tiles omitted are specified in the file `blanklist.txt` by their tile number in the topology, separated by a newline.

6.18.4 exch2, SIZE.h, and Multiprocessing

Once the topology configuration files are created, the Fortran `PARAMETER`s in `SIZE.h` must be configured to match. Section 4.3.1 *Specifying a decomposition* provides a general description of domain decomposition within MITgcm and its relation to `SIZE.h`. The current section specifies constraints that the exch2 package imposes and describes how to enable parallel execution with MPI.

As in the general case, the parameters `sNx` and `sNy` define the size of the individual tiles, and so must be assigned the same respective values as `tnx` and `tny` in `driver.m`.

The halo width parameters `OLx` and `OLy` have no special bearing on exch2 and may be assigned as in the general case. The same holds for `Nr`, the number of vertical levels in the model.

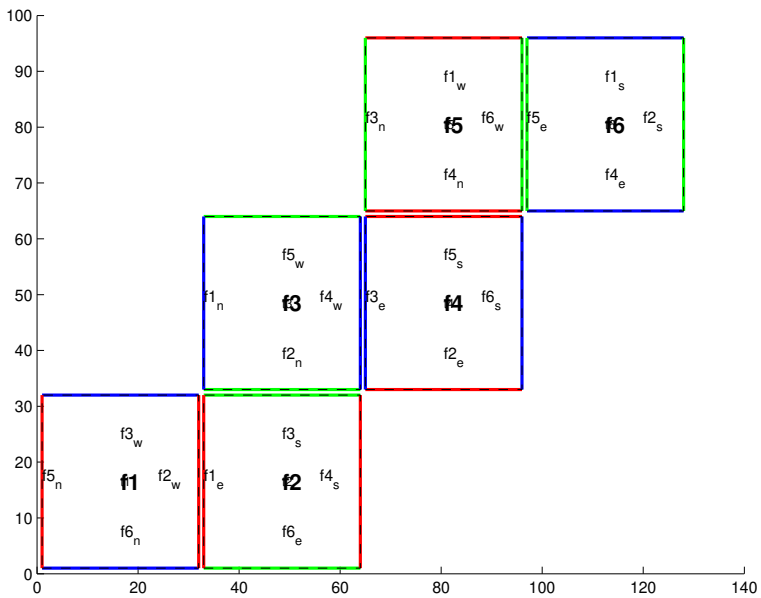


Figure 6.6: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains with one tile each (`tnx=32, tny=32`). This is the default configuration.

The parameters nSx, nSy, nPx, and nPy relate to the number of tiles and how they are distributed on processors. When using exch2, the tiles are stored in the x dimension, and so nSy=1 in all cases. Since the tiles as configured by exch2 cannot be split up accross processors without regenerating the topology, nPy=1 as well.

The number of tiles MITgcm allocates and how they are distributed between processors depends on nPx and nSx. nSx is the number of tiles per processor and nPx is the number of processors. The total number of tiles in the topology minus those listed in `blanklist.txt` must equal `nSx*nPx`. Note that in order to obtain maximum usage from a given number of processors in some cases, this restriction might entail sharing a processor with a tile that would otherwise be excluded because it is topographically outside of the domain and therefore in `blanklist.txt`. For example, suppose you have five processors and a domain decomposition of thirty-six tiles that allows you to exclude seven tiles. To evenly distribute the remaining twenty-nine tiles among five processors, you would have to run one "dummy" tile to make an even six tiles per processor. Such dummy tiles are *not* listed in `blanklist.txt`.

The following is an example of `SIZE.h` for the twelve-tile configuration illustrated in figure 6.5 running on one processor:

```
 PARAMETER (
&           sNx =  16,
&           sNy =  32,
&           OLx =   2,
&           OLy =   2,
&           nSx =  12,
&           nSy =   1,
&           nPx =   1,
&           nPy =   1,
&           Nx  = sNx*nSx*nPx,
&           Ny  = sNy*nSy*nPy,
&           Nr  =   5)
```

The following is an example for the twenty-four-tile topology in figure 6.4 running on six processors:

```
 PARAMETER (
&           sNx =  16,
&           sNy =  16,
&           OLx =   2,
&           OLy =   2,
&           nSx =   4,
```

```
&           nSy =   1,
&           nPx =   6,
&           nPy =   1,
&           Nx  = sNx*nSx*nPx,
&           Ny  = sNy*nSy*nPy,
&           Nr  =   5)
```

6.18.5 Key Variables

The descriptions of the variables are divided up into scalars, one-dimensional arrays indexed to the tile number, and two and three-dimensional arrays indexed to tile number and neighboring tile. This division reflects the functionality of these variables: The scalars are common to every part of the topology, the tile-indexed arrays to individual tiles, and the arrays indexed by tile and neighbor to relationships between tiles and their neighbors.

6.18.5.1 Scalars

The number of tiles in a particular topology is set with the parameter `NTILES`, and the maximum number of neighbors of any tiles by `MAX_NEIGHBOURS`. These parameters are used for defining the size of the various one and two dimensional arrays that store tile parameters indexed to the tile number and are assigned in the files generated by `driver.m`.

The scalar parameters `exch2_domain_nxt` and `exch2_domain_nyt` express the number of tiles in the x and y global indices. For example, the default setup of six tiles (Fig. 6.6) has `exch2_domain_nxt=6` and `exch2_domain_nyt=1`. A topology of twenty-four square tiles, four per subdomain (as in figure 6.4), will have `exch2_domain_nxt=12` and `exch2_domain_nyt=2`. Note that these parameters express the tile layout in order to allow global data files that are tile-layout-neutral. They have no bearing on the internal storage of the arrays. The tiles are stored internally in a range from `bi=(1:NTILES)` in the x axis, and the y axis variable `bj` is assumed to equal `1` throughout the package.

6.18.5.2 Arrays indexed to tile number

The following arrays are of length `NTILES` and are indexed to the tile number, which is indicated in the diagrams with the notation `t`n. The indices are omitted in the descriptions.

The arrays `exch2_tnx` and `exch2_tny` express the x and y dimensions of each tile. At present for each tile `exch2_tnx=sNx` and `exch2_tny=sNy`, as assigned in `SIZE.h` and described in Section 6.18.4 *exch2, SIZE.h, and Multiprocessing*.

Future releases of MITgcm may allow varying tile sizes.

The arrays exch2_tbasex and exch2_tbasey determine the tiles' Cartesian origin within a subdomain and locate the edges of different tiles relative to each other. As an example, in the default six-tile topology (Fig. 6.6) each index in these arrays is set to 0 since a tile occupies its entire subdomain. The twenty-four-tile case discussed above will have values of 0 or 16, depending on the quadrant of the tile within the subdomain. The elements of the arrays exch2_txglobalo and exch2_txglobalo are similar to exch2_tbasex and exch2_tbasey, but locate the tile edges within the global address space, similar to that used by global output and input files.

The array exch2_myFace contains the number of the subdomain of each tile, in a range (1:6) in the case of the standard cube topology and indicated by **f***n* in figures 6.5 and 6.4. exch2_nNeighbours contains a count of the neighboring tiles each tile has, and sets the bounds for looping over neighboring tiles. exch2_tProc holds the process rank of each tile, and is used in interprocess communication.

The arrays exch2_isWedge, exch2_isEedge, exch2_isSedge, and exch2_isNedge are set to 1 if the indexed tile lies on the edge of its subdomain, 0 if not. The values are used within the topology generator to determine the orientation of neighboring tiles, and to indicate whether a tile lies on the corner of a subdomain. The latter case requires special exchange and numerical handling for the singularities at the eight corners of the cube.

6.18.5.3 Arrays Indexed to Tile Number and Neighbor

The following arrays have vectors of length MAX_NEIGHBOURS and NTILES and describe the orientations between the the tiles.

The array exch2_neighbourId(a,T) holds the tile number Tn for each of the tile number T's neighboring tiles a. The neighbor tiles are indexed (1:exch2_nNeighbours(T)) in the order right to left on the north then south edges, and then top to bottom on the east then west edges.

The exch2_opposingSend_record(a,T) array holds the index b of the element in exch2_neighbourId(b,Tn) that holds the tile number T, given Tn=exch2_neighborId(a,T). In other words,

```
exch2_neighbourId( exch2_opposingSend_record(a,T),
                   exch2_neighbourId(a,T) ) = T
```

This provides a back-reference from the neighbor tiles.

The arrays exch2_pi and exch2_pj specify the transformations of indices in exchanges between the neighboring tiles. These transformations are necessary in exchanges between subdomains because a horizontal dimension in one subdomain may map to other horizonal dimension in an adjacent subdomain, and may also have its indexing reversed. This swapping arises from the "folding" of two-dimensional arrays into a three-dimensional cube.

The dimensions of `exch2_pi(t,N,T)` and `exch2_pj(t,N,T)` are the neighbor ID `N` and the tile number `T` as explained above, plus a vector of length `2` containing transformation factors `t`. The first element of the transformation vector holds the factor to multiply the index in the same dimension, and the second element holds the the same for the orthogonal dimension. To clarify, `exch2_pi(1,N,T)` holds the mapping of the x axis index of tile `T` to the x axis of tile `T`'s neighbor `N`, and `exch2_pi(2,N,T)` holds the mapping of `T`'s x index to the neighbor `N`'s y index.

One of the two elements of `exch2_pi` or `exch2_pj` for a given tile `T` and neighbor `N` will be `0`, reflecting the fact that the two axes are orthogonal. The other element will be `1` or `-1`, depending on whether the axes are indexed in the same or opposite directions. For example, the transform vector of the arrays for all tile neighbors on the same subdomain will be `(1,0)`, since all tiles on the same subdomain are oriented identically. An axis that corresponds to the orthogonal dimension with the same index direction in a particular tile-neighbor orientation will have `(0,1)`. Those with the opposite index direction will have `(0,-1)` in order to reverse the ordering.

The arrays exch2_oi, exch2_oj, exch2_oi_f, and exch2_oj_f are indexed to tile number and neighbor and specify the relative offset within the subdomain of the array index of a variable going from a neighboring tile `N` to a local tile `T`. Consider `T=1` in the six-tile topology (Fig. 6.6), where

```
exch2_oi(1,1)=33
exch2_oi(2,1)=0
exch2_oi(3,1)=32
exch2_oi(4,1)=-32
```

The simplest case is `exch2_oi(2,1)`, the southern neighbor, which is `Tn=6`. The axes of `T` and `Tn` have the same orientation and their x axes have the same origin, and so an exchange between the two requires no changes to the x index. For the western neighbor (`Tn=5`), `code_oi(3,1)=32` since the `x=0` vector on `T` corresponds to the `y=32` vector on `Tn`. The eastern edge of `T` shows the reverse case (`exch2_oi(4,1)=-32`), where `x=32` on `T` exchanges with `x=0` on `Tn=2`.

The most interesting case, where `exch2_oi(1,1)=33` and `Tn=3`, involves a reversal of indices. As in every case, the offset `exch2_oi` is added to the original x index of `T` multiplied by the transformation factor `exch2_pi(t,N,T)`. Here

exch2_pi(1,1,1)=0 since the x axis of T is orthogonal to the x axis of Tn. exch2_pi(2,1,1)=-1 since the x axis of T corresponds to the y axis of Tn, but the index is reversed. The result is that the index of the northern edge of T, which runs (1:32), is transformed to (-1:-32). exch2_oi(1,1) is then added to this range to get back (32:1) – the index of the y axis of Tn relative to T. This transformation may seem overly convoluted for the six-tile case, but it is necessary to provide a general solution for various topologies.

Finally, exch2_itlo_c, exch2_ithi_c, exch2_jtlo_c and exch2_jthi_c hold the location and index bounds of the edge segment of the neighbor tile N's subdomain that gets exchanged with the local tile T. To take the example of tile T=2 in the twelve-tile topology (Fig. 6.5):

```
exch2_itlo_c(4,2)=17
exch2_ithi_c(4,2)=17
exch2_jtlo_c(4,2)=0
exch2_jthi_c(4,2)=33
```

Here N=4, indicating the western neighbor, which is Tn=1. Tn resides on the same subdomain as T, so the tiles have the same orientation and the same x and y axes. The x axis is orthogonal to the western edge and the tile is 16 points wide, so exch2_itlo_c and exch2_ithi_c indicate the column beyond Tn's eastern edge, in that tile's halo region. Since the border of the tiles extends through the entire height of the subdomain, the y axis bounds exch2_jtlo_c to exch2_jthi_c cover the height of (1:32), plus 1 in either direction to cover part of the halo.

For the north edge of the same tile T=2 where N=1 and the neighbor tile is Tn=5:

```
exch2_itlo_c(1,2)=0
exch2_ithi_c(1,2)=0
exch2_jtlo_c(1,2)=0
exch2_jthi_c(1,2)=17
```

T's northern edge is parallel to the x axis, but since Tn's y axis corresponds to T's x axis, T's northern edge exchanges with Tn's western edge. The western edge of the tiles corresponds to the lower bound of the x axis, so exch2_itlo_c and exch2_ithi_c are 0, in the western halo region of Tn. The range of exch2_jtlo_c and exch2_jthi_c correspond to the width of T's northern edge, expanded by one into the halo.

6.18.6 Key Routines

Most of the subroutines particular to exch2 handle the exchanges themselves and are of the same format as those described in 4.3.3.3 *Cube sphere communi-*

cation. Like the original routines, they are written as templates which the local Makefile converts from `RX` into `RL` and `RS` forms.

The interfaces with the core model subroutines are `EXCH_UV_XY_RX`, `EXCH_UV_XYZ_RX` and `EXCH_XY_RX`. They override the standard exchange routines when `genmake2` is run with `exch2` option. They in turn call the local exch2 subroutines `EXCH2_UV_XY_RX` and `EXCH2_UV_XYZ_RX` for two and three-dimensional vector quantities, and `EXCH2_XY_RX` and `EXCH2_XYZ_RX` for two and three-dimensional scalar quantities. These subroutines set the dimensions of the area to be exchanged, call `EXCH2_RX1_CUBE` for scalars and `EXCH2_RX2_CUBE` for vectors, and then handle the singularities at the cube corners.

The separate scalar and vector forms of `EXCH2_RX1_CUBE` and `EXCH2_RX2_CUBE` reflect that the vector-handling subroutine needs to pass both the u and v components of the physical vectors. This swapping arises from the topological folding discussed above, where the x and y axes get swapped in some cases, and is not an issue with the scalar case. These subroutines call `EXCH2_SEND_RX1` and `EXCH2_SEND_RX2`, which do most of the work using the variables discussed above.

6.19 Diagnostics–A Flexible Infrastructure

6.19.1 Introduction

This section of the documentation describes the Diagnostics package available within the GCM. A large selection of model diagnostics is available for output. In addition to the diagnostic quantities pre-defined in the GCM, there exists the option, in any experiment, to define a new diagnostic quantity and include it as part of the diagnostic output with the addition of a single subroutine call in the routine where the field is computed. As a matter of philosophy, no diagnostic is enabled as default, thus each user must specify the exact diagnostic information required for an experiment. This is accomplished by enabling the specific diagnostic of interest cataloged in the Diagnostic Menu (see Section 6.19.4.1). Instructions for enabling diagnostic output and defining new diagnostic quantities are found in Section 6.19.4 of this document.

The Diagnostic Menu is a hard-wired enumeration of diagnostic quantities available within the GCM. Once a diagnostic is enabled, the GCM will continually increment an array specifically allocated for that diagnostic whenever the appropriate quantity is computed. A counter is defined which records how many times each diagnostic quantity has been incremented. Several special diagnostics are included in the menu. Quantities refered to as "Counter Diagnostics", are defined for selected diagnostics which record the frequency at which a diagnostic is incremented separately for each model grid location. Quantitied refered to as "User Diagnostics" are included in the menu to facilitate defining new diagnostics for a particular experiment.

6.19.2 Equations

Not relevant.

6.19.3 Key Subroutines and Parameters

The diagnostics are computed at various times and places within the GCM. Because the MIT GCM may employ a staggered grid, diagnostics may be computed at grid box centers, corners, or edges, and at the middle or edge in the vertical. Some diagnostics are scalars, while others are components of vectors. An internal array is defined which contains information concerning various grid attributes of each diagnostic. The GDIAG array (in common block
diagnostics in file diagnostics.h) is internally defined as a character*8 variable, and is equivalenced to a character*1 "parse" array in output in order to extract the grid-attribute information. The GDIAG array is described in Table 6.1.

As an example, consider a diagnostic whose associated GDIAG parameter is equal to "UU 002". From GDIAG we can determine that this diagnostic is a U-vector component located at the C-grid U-point. Its corresponding V-component diagnostic is located in Diagnostic # 002.

Table 6.1: Diagnostic Parsing Array

Diagnostic Parsing Array		
Array	Value	Description
parse(1)	→ S	Scalar Diagnostic
	→ U	U-vector component Diagnostic
	→ V	V-vector component Diagnostic
parse(2)	→ U	C-Grid U-Point
	→ V	C-Grid V-Point
	→ M	C-Grid Mass Point
	→ Z	C-Grid Vorticity (Corner) Point
parse(3)	→ R	Not Currently in Use
parse(4)	→ P	Positive Definite Diagnostic
parse(5)	→ C	Counter Diagnostic
	→ D	Disabled Diagnostic for output
parse(6-8)	→ C	3-digit integer corresponding to vector or counter component mate

In this way, each Diagnostic in the model has its attributes (ie. vector or scalar, C-grid location, etc.) defined internally. The Output routines use this information in order to determine what type of transformations need to be performed. Any interpolations are done at the time of output rather than during each model step. In this way the User has flexibility in determining the type of gridded data which is output.
There are several utilities within the GCM available to users to enable, disable, clear, write and retrieve model diagnostics, and may be called from any routine. The available utilities and the CALL sequences are listed below.
fill_diagnostics: This routine will increment the specified diagnostic quantity with a field sent through the argument list.

call fill_diagnostics (myThid, chardiag, levflg, nlevs,
bibjflg, bi, bj, arrayin)

where myThid = Current Process(or)
chardiag = Character *8 expression for diag to fill
levflg = Integer flag for vertical levels:
0 indicates multiple levels incremented in qdiag
non-0 (any integer) - WHICH single level to increment.
negative integer - the input data array is single-leveled
positive integer - the input data array is multi-leveled
nlevs = indicates Number of levels to be filled (1 if levflg ¡¿ 0)
positive: fill in "nlevs" levels in the same order as
the input array
negative: fill in -nlevs levels in reverse order.

bibjflg	= Integer flag to indicate instructions for bi bj loop 0 indicates that the bi-bj loop must be done here 1 indicates that the bi-bj loop is done OUTSIDE 2 indicates that the bi-bj loop is done OUTSIDE AND that we have been sent a local array 3 indicates that the bi-bj loop is done OUTSIDE AND that we have been sent a local array AND that the array has the shadow regions
bi	= X-direction process(or) number - used for bibjflg=1-3
bj	= Y-direction process(or) number - used for bibjflg=1-3
arrayin	= Field to increment diagnostics array

setdiag: This subroutine enables a diagnostic from the Diagnostic Menu, meaning that space is allocated for the diagnostic and the model routines will increment the diagnostic value during execution. This routine is the underlying interface between the user and the desired diagnostic. The diagnostic is referenced by its diagnostic number from the menu, and its calling sequence is given by:

call setdiag (num)

where	num	= Diagnostic number from menu

getdiag: This subroutine retrieves the value of a model diagnostic. This routine is particulary useful when called from a user output routine, although it can be called from any routine. This routine returns the time-averaged value of the diagnostic by dividing the current accumulated diagnostic value by its corresponding counter. This routine does not change the value of the diagnostic itself, that is, it does not replace the diagnostic with its time-average. The calling sequence for this routine is givin by:

call getdiag (lev,num,qtmp,undef)

where	lev	= Model Level at which the diagnostic is desired
	num	= Diagnostic number from menu
	qtmp	= Time-Averaged Diagnostic Output
	undef	= Fill value to be used when diagnostic is undefined

clrdiag: This subroutine initializes the values of model diagnostics to zero, and is particularly useful when called from user output routines to re-initialize diagnostics during the run. The calling sequence is:

call clrdiag (num)

where	num	= Diagnostic number from menu

zapdiag: This entry into subroutine SETDIAG disables model diagnostics, meaning that the diagnostic is no longer available to the user. The memory previously allocated to the diagnostic is released when ZAPDIAG is invoked. The calling sequence is given by:

call zapdiag (NUM)

where num = Diagnostic number from menu

6.19.4 Usage Notes

We begin this section with a discussion on the manner in which computer memory is allocated for diagnostics. All GCM diagnostic quantities are stored in the single diagnostic array QDIAG which is located in the file pkg/diagnostics/diagnostics.h. and has the form:

common /diagnostics/ qdiag(1-Olx,sNx+Olx,1-Olx,sNx+Olx,numdiags,Nsx,Nsy)

where numdiags is an Integer variable which should be set equal to the number of enabled diagnostics, and qdiag is a three-dimensional array. The first two-dimensions of qdiag correspond to the horizontal dimension of a given diagnostic, while the third dimension of qdiag is used to identify diagnostic fields and levels combined. In order to minimize the memory requirement of the model for diagnostics, the default GCM executable is compiled with room for only one horizontal diagnostic array, or with numdiags set to 1. In order for the User to enable more than 1 two-dimensional diagnostic, the size of the diagnostics common must be expanded to accomodate the desired diagnostics. This can be accomplished by manually changing the parameter numdiags in the file pkg/diagnostics/diagnostics_SIZE.h. numdiags should be set greater than or equal to the sum of all the diagnostics activated for output each multiplied by the number of levels defined for that diagnostic quantity. This is illustrated in the example below:

To use the diagnostics package, other than enabling it in packages.conf and turning the usediagnostics flag in data.pkg to .TRUE., a namelist must be supplied in the run directory called data.diagnostics. The namelist will activate a user-defined list of diagnostics quantities to be computed, specify the frequency of output, the number of levels, and the name of up to 10 separate output files. A sample data.diagnostics namelist file:

```
# Diagnostic Package Choices
&diagnostics_list
frequency(1) = 10,
levels(1,1) = 1.,2.,3.,4.,5.,
fields(1,1) = 'UVEL ','VVEL ',
filename(1) = 'diagout1',
frequency(2) = 100,
levels(1,2) = 1.,2.,3.,4.,5.,
fields(1,2) = 'THETA ','SALT ',
```

filename(2) = 'diagout2',
&end

In this example, there are two output files that will be generated for each tile and for each output time. The first set of output files has the prefix diagout1, does time averaging every 10 time steps (frequency is 10), they will write fields which are multiple-level fields and output levels 1-5. The names of diagnostics quantities are UVEL and VVEL. The second set of output files has the prefix diagout2, does time averaging every 100 time steps, they include fields which are multiple-level fields, levels output are 1-5, and the names of diagnostics quantities are THETA and SALT.
In order to define and include as part of the diagnostic output any field that is desired for a particular experiment, two steps must be taken. The first is to enable the "User Diagnostic" in data.diagnostics. This is accomplished by setting one of the fields slots to either UDIAG1 through UDIAG10, for multi-level fields, or SDIAG1 through SDIAG10 for single level fields. These are listed in the diagnostics menu. The second step is to add a call to fill_diagnostics from the subroutine in which the quantity desired for diagnostic output is computed.

6.19.4.1 GCM Diagnostic Menu

N	NAME	UNITS	LEVELS	DESCRIPTION
1	UFLUX	$Newton/m^2$	1	Surface U-Wind Stress on the atmosphere
2	VFLUX	$Newton/m^2$	1	Surface V-Wind Stress on the atmosphere
3	HFLUX	$Watts/m^2$	1	Surface Flux of Sensible Heat
4	EFLUX	$Watts/m^2$	1	Surface Flux of Latent Heat
5	QICE	$Watts/m^2$	1	Heat Conduction through Sea-Ice
6	RADLWG	$Watts/m^2$	1	Net upward LW flux at the ground
7	RADSWG	$Watts/m^2$	1	Net downward SW flux at the ground
8	RI	$dimensionless$	Nrphys	Richardson Number
9	CT	$dimensionless$	1	Surface Drag coefficient for T and Q
10	CU	$dimensionless$	1	Surface Drag coefficient for U and V
11	ET	m^2/sec	Nrphys	Diffusivity coefficient for T and Q
12	EU	m^2/sec	Nrphys	Diffusivity coefficient for U and V
13	TURBU	$m/sec/day$	Nrphys	U-Momentum Changes due to Turbulence
14	TURBV	$m/sec/day$	Nrphys	V-Momentum Changes due to Turbulence
15	TURBT	deg/day	Nrphys	Temperature Changes due to Turbulence
16	TURBQ	$g/kg/day$	Nrphys	Specific Humidity Changes due to Turbulence
17	MOISTT	deg/day	Nrphys	Temperature Changes due to Moist Processes
18	MOISTQ	$g/kg/day$	Nrphys	Specific Humidity Changes due to Moist Processes
19	RADLW	deg/day	Nrphys	Net Longwave heating rate for each level
20	RADSW	deg/day	Nrphys	Net Shortwave heating rate for each level
21	PREACC	mm/day	1	Total Precipitation
22	PRECON	mm/day	1	Convective Precipitation
23	TUFLUX	$Newton/m^2$	Nrphys	Turbulent Flux of U-Momentum
24	TVFLUX	$Newton/m^2$	Nrphys	Turbulent Flux of V-Momentum
25	TTFLUX	$Watts/m^2$	Nrphys	Turbulent Flux of Sensible Heat

N	NAME	UNITS	LEVELS	DESCRIPTION
26	TQFLUX	$Watts/m^2$	Nrphys	Turbulent Flux of Latent Heat
27	CN	$dimensionless$	1	Neutral Drag Coefficient
28	WINDS	m/sec	1	Surface Wind Speed
29	DTSRF	deg	1	Air/Surface virtual temperature difference
30	TG	deg	1	Ground temperature
31	TS	deg	1	Surface air temperature (Adiabatic from lowest model layer)
32	DTG	deg	1	Ground temperature adjustment
33	QG	g/kg	1	Ground specific humidity
34	QS	g/kg	1	Saturation surface specific humidity
35	TGRLW	deg	1	Instantaneous ground temperature used as input to the Longwave radiation subroutine
36	ST4	$Watts/m^2$	1	Upward Longwave flux at the ground (σT^4)
37	OLR	$Watts/m^2$	1	Net upward Longwave flux at the top of the model
38	OLRCLR	$Watts/m^2$	1	Net upward clearsky Longwave flux at the top of the model
39	LWGCLR	$Watts/m^2$	1	Net upward clearsky Longwave flux at the ground
40	LWCLR	deg/day	Nrphys	Net clearsky Longwave heating rate for each level
41	TLW	deg	Nrphys	Instantaneous temperature used as input to the Longwave radiation subroutine
42	SHLW	g/g	Nrphys	Instantaneous specific humidity used as input to the Longwave radiation subroutine
43	OZLW	g/g	Nrphys	Instantaneous ozone used as input to the Longwave radiation subroutine
44	CLMOLW	$0-1$	Nrphys	Maximum overlap cloud fraction used in the Longwave radiation subroutine
45	CLDTOT	$0-1$	Nrphys	Total cloud fraction used in the Longwave and Shortwave radiation subroutines
46	LWGDOWN	$Watts/m^2$	1	Downwelling Longwave radiation at the ground
47	GWDT	deg/day	Nrphys	Temperature tendency due to Gravity Wave Drag
48	RADSWT	$Watts/m^2$	1	Incident Shortwave radiation at the top of the atmosphere
49	TAUCLD	$per100mb$	Nrphys	Counted Cloud Optical Depth (non-dimensional) per 100 mb
50	TAUCLDC	$Number$	Nrphys	Cloud Optical Depth Counter

N	NAME	UNITS	LEVELS	DESCRIPTION
51	CLDLOW	$0-1$	Nrphys	Low-Level (1000-700 hPa) Cloud Fraction (0-1)
52	EVAP	mm/day	1	Surface evaporation
53	DPDT	hPa/day	1	Surface Pressure tendency
54	UAVE	m/sec	Nrphys	Average U-Wind
55	VAVE	m/sec	Nrphys	Average V-Wind
56	TAVE	deg	Nrphys	Average Temperature
57	QAVE	g/kg	Nrphys	Average Specific Humidity
58	OMEGA	hPa/day	Nrphys	Vertical Velocity
59	DUDT	$m/sec/day$	Nrphys	Total U-Wind tendency
60	DVDT	$m/sec/day$	Nrphys	Total V-Wind tendency
61	DTDT	deg/day	Nrphys	Total Temperature tendency
62	DQDT	$g/kg/day$	Nrphys	Total Specific Humidity tendency
63	VORT	$10^{-4}/sec$	Nrphys	Relative Vorticity
64	NOT USED			
65	DTLS	deg/day	Nrphys	Temperature tendency due to Stratiform Cloud Formation
66	DQLS	$g/kg/day$	Nrphys	Specific Humidity tendency due to Stratiform Cloud Formation
67	USTAR	m/sec	1	Surface USTAR wind
68	Z0	m	1	Surface roughness
69	FRQTRB	$0-1$	Nrphys-1	Frequency of Turbulence
70	PBL	mb	1	Planetary Boundary Layer depth
71	SWCLR	deg/day	Nrphys	Net clearsky Shortwave heating rate for each level
72	OSR	$Watts/m^2$	1	Net downward Shortwave flux at the top of the model
73	OSRCLR	$Watts/m^2$	1	Net downward clearsky Shortwave flux at the top of the model
74	CLDMAS	kg/m^2	Nrphys	Convective cloud mass flux
75	UAVE	m/sec	Nrphys	Time-averaged $u-Wind$

N	NAME	UNITS	LEVELS	DESCRIPTION
76	VAVE	m/sec	Nrphys	Time-averaged $v - Wind$
77	TAVE	deg	Nrphys	Time-averaged $Temperature$
78	QAVE	g/g	Nrphys	Time-averaged $Specific\ Humidity$
79	RFT	deg/day	Nrphys	Temperature tendency due Rayleigh Friction
80	PS	mb	1	Surface Pressure
81	QQAVE	$(m/sec)^2$	Nrphys	Time-averaged $TurbulentKineticEnergy$
82	SWGCLR	$Watts/m^2$	1	Net downward clearsky Shortwave flux at the ground
83	PAVE	mb	1	Time-averaged Surface Pressure
84	SDIAG1		1	User-Defined Surface Diagnostic-1
85	SDIAG2		1	User-Defined Surface Diagnostic-2
86	UDIAG1		Nrphys	User-Defined Upper-Air Diagnostic-1
87	UDIAG2		Nrphys	User-Defined Upper-Air Diagnostic-2
88	DIABU	$m/sec/day$	Nrphys	Total Diabatic forcing on $u - Wind$
89	DIABV	$m/sec/day$	Nrphys	Total Diabatic forcing on $v - Wind$
90	DIABT	deg/day	Nrphys	Total Diabatic forcing on $Temperature$
91	DIABQ	$g/kg/day$	Nrphys	Total Diabatic forcing on $Specific\ Humidity$
92	RFU	$m/sec/day$	Nrphys	U-Wind tendency due to Rayleigh Friction
93	RFV	$m/sec/day$	Nrphys	V-Wind tendency due to Rayleigh Friction
94	GWDU	$m/sec/day$	Nrphys	U-Wind tendency due to Gravity Wave Drag
95	GWDU	$m/sec/day$	Nrphys	V-Wind tendency due to Gravity Wave Drag
96	GWDUS	N/m^2	1	U-Wind Gravity Wave Drag Stress at Surface
97	GWDVS	N/m^2	1	V-Wind Gravity Wave Drag Stress at Surface
98	GWDUT	N/m^2	1	U-Wind Gravity Wave Drag Stress at Top
99	GWDVT	N/m^2	1	V-Wind Gravity Wave Drag Stress at Top
100	LZRAD	mg/kg	Nrphys	Estimated Cloud Liquid Water used in Radiation

N	NAME	UNITS	LEVELS	DESCRIPTION
101	SLP	mb	1	Time-averaged Sea-level Pressure
102	NOT USED			
103	NOT USED			
104	NOT USED			
105	NOT USED			
106	CLDFRC	$0-1$	1	Total Cloud Fraction
107	TPW	gm/cm^2	1	Precipitable water
108	U2M	m/sec	1	U-Wind at 2 meters
109	V2M	m/sec	1	V-Wind at 2 meters
110	T2M	deg	1	Temperature at 2 meters
111	Q2M	g/kg	1	Specific Humidity at 2 meters
112	U10M	m/sec	1	U-Wind at 10 meters
113	V10M	m/sec	1	V-Wind at 10 meters
114	T10M	deg	1	Temperature at 10 meters
115	Q10M	g/kg	1	Specific Humidity at 10 meters
116	DTRAIN	kg/m^2	Nrphys	Detrainment Cloud Mass Flux
117	QFILL	$g/kg/day$	Nrphys	Filling of negative specific humidity
118	NOT USED			
119	NOT USED			
120	SHAPU	$m/sec/day$	Nrphys	U-Wind tendency due to Shapiro Filter
121	SHAPV	$m/sec/day$	Nrphys	V-Wind tendency due to Shapiro Filter
122	SHAPT	deg/day	Nrphys	Temperature tendency due Shapiro Filter
123	SHAPQ	$g/kg/day$	Nrphys	Specific Humidity tendency due to Shapiro Filter
124	SDIAG3		1	User-Defined Surface Diagnostic-3
125	SDIAG4		1	User-Defined Surface Diagnostic-4

N	NAME	UNITS	LEVELS	DESCRIPTION
126	SDIAG5		1	User-Defined Surface Diagnostic-5
127	SDIAG6		1	User-Defined Surface Diagnostic-6
128	SDIAG7		1	User-Defined Surface Diagnostic-7
129	SDIAG8		1	User-Defined Surface Diagnostic-8
130	SDIAG9		1	User-Defined Surface Diagnostic-9
131	SDIAG10		1	User-Defined Surface Diagnostic-1-
132	UDIAG3		Nrphys	User-Defined Multi-Level Diagnostic-3
133	UDIAG4		Nrphys	User-Defined Multi-Level Diagnostic-4
134	UDIAG5		Nrphys	User-Defined Multi-Level Diagnostic-5
135	UDIAG6		Nrphys	User-Defined Multi-Level Diagnostic-6
136	UDIAG7		Nrphys	User-Defined Multi-Level Diagnostic-7
137	UDIAG8		Nrphys	User-Defined Multi-Level Diagnostic-8
138	UDIAG9		Nrphys	User-Defined Multi-Level Diagnostic-9
139	UDIAG10		Nrphys	User-Defined Multi-Level Diagnostic-10

N	NAME	UNITS	LEVELS	DESCRIPTION
238	ETAN	(hPa, m)	1	Perturbation of Surface (pressure, height)
239	ETANSQ	(hPa^2, m^2)	1	Square of Perturbation of Surface (pressure, height)
240	THETA	$degK$	Nr	Potential Temperature
241	SALT	g/kg	Nr	Salt (or Water Vapor Mixing Ratio)
242	UVEL	m/sec	Nr	U-Velocity
243	VVEL	m/sec	Nr	V-Velocity
244	WVEL	m/sec	Nr	Vertical-Velocity
245	THETASQ	deg^2	Nr	Square of Potential Temperature
246	SALTSQ	g^2/kg^2	Nr	Square of Salt (or Water Vapor Mixing Ratio)
247	UVELSQ	m^2/sec^2	Nr	Square of U-Velocity
248	VVELSQ	m^2/sec^2	Nr	Square of V-Velocity
249	WVELSQ	m^2/sec^2	Nr	Square of Vertical-Velocity
250	UVELVVEL	m^2/sec^2	Nr	Meridional Transport of Zonal Momentum

N	NAME	UNITS	LEVELS	DESCRIPTION
251	UVELMASS	m/sec	Nr	Zonal Mass-Weighted Component of Velocity
252	VVELMASS	m/sec	Nr	Meridional Mass-Weighted Component of Velo
253	WVELMASS	m/sec	Nr	Vertical Mass-Weighted Component of Velocit
254	UTHMASS	$m - deg/sec$	Nr	Zonal Mass-Weight Transp of Pot Temp
255	VTHMASS	$m - deg/sec$	Nr	Meridional Mass-Weight Transp of Pot Temp
256	WTHMASS	$m - deg/sec$	Nr	Vertical Mass-Weight Transp of Pot Temp
257	USLTMASS	$m - kg/sec - kg$	Nr	Zonal Mass-Weight Transp of Salt (or W.Vap Rat.)
258	VSLTMASS	$m - kg/sec - kg$	Nr	Meridional Mass-Weight Transp of Salt (or W. Mix Rat.)
259	WSLTMASS	$m - kg/sec - kg$	Nr	Vertical Mass-Weight Transp of Salt (or W. Mix Rat.)
260	UVELTH	$m - deg/sec$	Nr	Zonal Transp of Pot Temp
261	VVELTH	$m - deg/sec$	Nr	Meridional Transp of Pot Temp
262	WVELTH	$m - deg/sec$	Nr	Vertical Transp of Pot Temp
263	UVELSLT	$m - kg/sec - kg$	Nr	Zonal Transp of Salt (or W.Vap Mix Rat.)
264	VVELSLT	$m - kg/sec - kg$	Nr	Meridional Transp of Salt (or W.Vap Mix Ra
265	WVELSLT	$m - kg/sec - kg$	Nr	Vertical Transp of Salt (or W.Vap Mix Rat.)
266	UTRAC1	$m - kg/sec - kg$	Nr	Zonal Transp of Tracer 1
267	VTRAC1	$m - kg/sec - kg$	Nr	Meridional Transp of Tracer 1
268	WTRAC1	$m - kg/sec - kg$	Nr	Vertical Transp of Tracer 1
269	UTRAC2	$m - kg/sec - kg$	Nr	Zonal Transp of Tracer 2
270	VTRAC2	$m - kg/sec - kg$	Nr	Meridional Transp of Tracer 2
271	WTRAC2	$m - kg/sec - kg$	Nr	Vertical Transp of Tracer 2
272	UTRAC3	$m - kg/sec - kg$	Nr	Zonal Transp of Tracer 3
273	VTRAC3	$m - kg/sec - kg$	Nr	Meridional Transp of Tracer 3
274	WTRAC3	$m - kg/sec - kg$	Nr	Vertical Transp of Tracer 3
275	WSLTMASS	$m - kg/sec - kg$	Nr	Vertical Mass-Weight Transp of Salt (or W. Mix Rat.)

N	NAME	UNITS	LEVELS	DESCRIPTION
275	UTRAC4	$m-kg/sec-kg$	Nr	Zonal Transp of Tracer 4
276	VTRAC4	$m-kg/sec-kg$	Nr	Meridional Transp of Tracer 4
277	WTRAC4	$m-kg/sec-kg$	Nr	Vertical Transp of Tracer 4
278	UTRAC5	$m-kg/sec-kg$	Nr	Zonal Transp of Tracer 5
279	VTRAC5	$m-kg/sec-kg$	Nr	Meridional Transp of Tracer 5
280	WTRAC5	$m-kg/sec-kg$	Nr	Vertical Transp of Tracer 5
281	TRAC1	kg/kg	Nr	Mass-Weight Tracer 1
282	TRAC2	kg/kg	Nr	Mass-Weight Tracer 2
283	TRAC3	kg/kg	Nr	Mass-Weight Tracer 3
284	TRAC4	kg/kg	Nr	Mass-Weight Tracer 4
285	TRAC5	kg/kg	Nr	Mass-Weight Tracer 5
286	DICBIOA	$mol/m3/s$	Nr	Biological Productivity
287	DICCARB	$moleq/m3/s$	Nr	Carbonate chg-biol prod and remin
288	DICTFLX	$mol/m3/s$	1	Tendency of DIC due to air-sea exch
289	DICOFLX	$mol/m3/s$	1	Tendency of O2 due to air-sea exch
290	DICCFLX	$mol/m2/s$	1	Flux of CO2 - air-sea exch
291	DICPCO2	atm	1	Partial Pressure of CO2
292	DICPHAV	$dimensionless$	1	Average pH
293	DTCONV	deg/sec	Nr	Temp Change due to Convection
294	DQCONV	$g/kg/sec$	Nr	Specific Humidity Change due to Convection
295	RELHUM	$percent$	Nr	Relative Humidity
296	PRECLS	$g/m^2/sec$	1	Large Scale Precipitation
297	ENPREC	J/g	1	Energy of Precipitation (snow, rain Temp)
298	VISCA4	m^4/sec	1	Biharmonic Viscosity Coefficient
299	VISCAH	m^2/sec	1	Harmonic Viscosity Coefficient
300	DRHODR	$kg/m^3/r-unit$	Nr	Stratification: d.Sigma/dr

N	NAME	UNITS	LEVELS	DESCRIPTION
301	DETADT2	$r - unit^2/s^2$	1	Square of Eta (Surf.P,SSH) Tendency

6.19.4.2 Diagnostic Description

In this section we list and describe the diagnostic quantities available within the GCM. The diagnostics are listed in the order that they appear in the Diagnostic Menu, Section 6.19.4.1. In all cases, each diagnostic as currently archived on the output datasets is time-averaged over its diagnostic output frequency:

$$\textbf{DIAGNOSTIC} = \frac{1}{TTOT} \sum_{t=1}^{t=TTOT} diag(t)$$

where $TTOT = \frac{\textbf{NQDIAG}}{\Delta t}$, **NQDIAG** is the output frequency of the diagnostic, and Δt is the timestep over which the diagnostic is updated.

1) UFLUX Surface Zonal Wind Stress on the Atmosphere ($Newton/m^2$)

The zonal wind stress is the turbulent flux of zonal momentum from the surface. See section 3.3 for a description of the surface layer parameterization.

$$\textbf{UFLUX} = -\rho C_D W_s u \qquad where: \;\; C_D = C_u^2$$

where ρ = the atmospheric density at the surface, C_D is the surface drag coefficient, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), W_s is the magnitude of the surface layer wind, and u is the zonal wind in the lowest model layer.

2) VFLUX Surface Meridional Wind Stress on the Atmosphere ($Newton/m^2$)

The meridional wind stress is the turbulent flux of meridional momentum from the surface. See section 3.3 for a description of the surface layer parameterization.

$$\textbf{VFLUX} = -\rho C_D W_s v \qquad where: \;\; C_D = C_u^2$$

where ρ = the atmospheric density at the surface, C_D is the surface drag coefficient, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), W_s is the magnitude of the surface layer wind, and v is the meridional wind in the lowest model layer.

3) HFLUX Surface Flux of Sensible Heat ($Watts/m^2$)

The turbulent flux of sensible heat from the surface to the atmosphere is a function of the gradient of virtual potential temperature and the eddy exchange coefficient:

$$\textbf{HFLUX} = P^{\kappa} \rho c_p C_H W_s (\theta_{surface} - \theta_{Nrphys}) \qquad where: \;\; C_H = C_u C_t$$

where ρ = the atmospheric density at the surface, c_p is the specific heat of air, C_H is the dimensionless surface heat transfer coefficient, W_s is the magnitude of the surface layer wind, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), C_t is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and θ is the

potential temperature at the surface and at the bottom model level.

4) EFLUX Surface Flux of Latent Heat ($Watts/m^2$)

The turbulent flux of latent heat from the surface to the atmosphere is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

$$\mathbf{EFLUX} = \rho\beta L C_H W_s (q_{surface} - q_{Nrphys}) \qquad where: \;\; C_H = C_u C_t$$

where $\rho =$ the atmospheric density at the surface, β is the fraction of the potential evapotranspiration actually evaporated, L is the latent heat of evaporation, C_H is the dimensionless surface heat transfer coefficient, W_s is the magnitude of the surface layer wind, C_u is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), C_t is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and $q_{surface}$ and q_{Nrphys} are the specific humidity at the surface and at the bottom model level, respectively.

5) QICE Heat Conduction Through Sea Ice ($Watts/m^2$)

Over sea ice there is an additional source of energy at the surface due to the heat conduction from the relatively warm ocean through the sea ice. The heat conduction through sea ice represents an additional energy source term for the ground temperature equation.

$$\mathbf{QICE} = \frac{C_{ti}}{H_i}(T_i - T_g)$$

where C_{ti} is the thermal conductivity of ice, H_i is the ice thickness, assumed to be $3\,m$ where sea ice is present, T_i is 273 degrees Kelvin, and T_g is the temperature of the sea ice.

NOTE: QICE is not available through model version 5.3, but is available in subsequent versions.

6) RADLWG Net upward Longwave Flux at the surface ($Watts/m^2$)

$$\begin{aligned} \mathbf{RADLWG} &= F^{Net}_{LW,Nrphys+1} \\ &= F^{\uparrow}_{LW,Nrphys+1} - F^{\downarrow}_{LW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F^{\uparrow}_{LW}$ is the upward Longwave flux and $F^{\downarrow}_{LW}$ is the downward Longwave flux.

7) RADSWG Net downard shortwave Flux at the surface ($Watts/m^2$)

$$\begin{aligned} \mathbf{RADSWG} &= F^{Net}_{SW,Nrphys+1} \\ &= F^{\downarrow}_{SW,Nrphys+1} - F^{\uparrow}_{SW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F_{SW}^{\downarrow}$ is the downward Shortwave flux and $F_{SW}^{\uparrow}$ is the upward Shortwave flux.

8) RI Richardson Number (*dimensionless*)
The non-dimensional stability indicator is the ratio of the buoyancy to the shear:

$$\mathbf{RI} = \frac{\frac{g}{\theta_v}\frac{\partial \theta_v}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2} = \frac{c_p \frac{\partial \theta_v}{\partial z}\frac{\partial P^\kappa}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2}$$

where we used the hydrostatic equation:

$$\frac{\partial \Phi}{\partial P^\kappa} = c_p \theta_v$$

Negative values indicate unstable buoyancy **AND** shear, small positive values (< 0.4) indicate dominantly unstable shear, and large positive values indicate dominantly stable stratification.

9) CT Surface Exchange Coefficient for Temperature and Moisture (*dimensionless*)
The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

$$\mathbf{CT} = -\frac{(\overline{w'\theta'})}{u_* \Delta\theta} = -\frac{(\overline{w'q'})}{u_* \Delta q} = \frac{k}{(\psi_h + \psi_g)}$$

where ψ_h is the surface layer non-dimensional temperature change and ψ_g is the viscous sublayer non-dimensional temperature or moisture change:

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta \qquad and \qquad \psi_g = \frac{0.55(Pr^{2/3} - 0.2)}{\nu^{1/2}} (h_0 u_* - h_{0_{ref}} u_{*_{ref}})^{1/2}$$

and: $h_0 = 30 z_0$ with a maximum value over land of 0.01
ϕ_h is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to Helfand and Schubert, 1993. k is the Von Karman constant, ζ is the non-dimensional stability parameter, Pr is the Prandtl number for air, ν is the molecular viscosity, z_0 is the surface roughness length, u_* is the surface stress velocity (see diagnostic number 67), and the subscript ref refers to a reference value.

10) CU Surface Exchange Coefficient for Momentum (*dimensionless*)
The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

$$\mathbf{CU} = \frac{u_*}{W_s} = \frac{k}{\psi_m}$$

where ψ_m is the surface layer non-dimensional wind shear:

$$\psi_m = \int_{\zeta_0}^{\zeta} \frac{\phi_m}{\zeta} d\zeta$$

ϕ_m is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to Helfand and Schubert, 1993. k is the Von Karman constant, ζ is the non-dimensional stability parameter, u_* is the surface stress velocity (see diagnostic number 67), and W_s is the magnitude of the surface layer wind.

11) ET Diffusivity Coefficient for Temperature and Moisture (m^2/sec)
In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat or moisture flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient K_h times the negative of the gradient of potential temperature or moisture. In the Helfand and Labraga (1988) adaptation of this closure, K_h takes the form:

$$\mathbf{ET} = K_h = -\frac{(\overline{w'\theta'_v})}{\frac{\partial \theta_v}{\partial z}} = \begin{cases} q\,\ell\,S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_H(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * turbulent\ \ kinetic\ \ energy}$, q_e is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, ℓ is the master length scale related to the layer depth, S_H is a function of G_H and G_M, the dimensionless buoyancy and wind shear parameters, respectively, or a function of G_{H_e} and G_{M_e}, the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number. For the detailed equations and derivations of the modified level 2.5 closure scheme, see Helfand and Labraga, 1988.
In the surface layer, **ET** is the exchange coefficient for heat and moisture, in units of m/sec, given by:

$$\mathbf{ET_{Nrphys}} = C_t * u_* = C_H W_s$$

where C_t is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions (see diagnostic number 9), u_* is the surface friction velocity (see diagnostic number 67), C_H is the heat transfer coefficient, and W_s is the magnitude of the surface layer wind.

12) EU Diffusivity Coefficient for Momentum (m^2/sec)
In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat momentum flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient K_m times the negative of the gradient of the u-wind. In the Helfand and Labraga (1988) adaptation of this closure, K_m takes

the form:

$$\mathbf{EU} = K_m = -\frac{(\overline{u'w'})}{\frac{\partial U}{\partial z}} = \begin{cases} q\,\ell\,S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_M(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * turbulent\ \ kinetic\ \ energy}$, q_e is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, ℓ is the master length scale related to the layer depth, S_M is a function of G_H and G_M, the dimensionless buoyancy and wind shear parameters, respectively, or a function of G_{H_e} and G_{M_e}, the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number. For the detailed equations and derivations of the modified level 2.5 closure scheme, see Helfand and Labraga, 1988.
In the surface layer, **EU** is the exchange coefficient for momentum, in units of m/sec, given by:

$$\mathbf{EU_{Nrphys}} = C_u * u_* = C_D W_s$$

where C_u is the dimensionless exchange coefficient for momentum from the surface layer similarity functions (see diagnostic number 10), u_* is the surface friction velocity (see diagnostic number 67), C_D is the surface drag coefficient, and W_s is the magnitude of the surface layer wind.

13) TURBU Zonal U-Momentum changes due to Turbulence ($m/sec/day$)

The tendency of U-Momentum due to turbulence is written:

$$\mathbf{TURBU} = \frac{\partial u}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{u'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of u-momentum in terms of K_m, and the equation has the form of a diffusion equation.

14) TURBV Meridional V-Momentum changes due to Turbulence ($m/sec/day$)

The tendency of V-Momentum due to turbulence is written:

$$\mathbf{TURBV} = \frac{\partial v}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{v'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of v-momentum in terms of K_m, and the equation has the form of a diffusion equation.

15) TURBT Temperature changes due to Turbulence (deg/day)

The tendency of temperature due to turbulence is written:

$$\mathbf{TURBT} = \frac{\partial T}{\partial t} = P^\kappa \frac{\partial \theta}{\partial t}_{turb} = P^\kappa \frac{\partial}{\partial z}(-\overline{w'\theta'}) = P^\kappa \frac{\partial}{\partial z}(K_h \frac{\partial \theta_v}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of temperature in terms of K_h, and the equation has the form of a diffusion equation.

16) TURBQ Specific Humidity changes due to Turbulence ($g/kg/day$)
The tendency of specific humidity due to turbulence is written:

$$\mathbf{TURBQ} = \frac{\partial q}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{w'q'}) = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of temperature in terms of K_h, and the equation has the form of a diffusion equation.

17) MOISTT Temperature Changes Due to Moist Processes (deg/day)

$$\mathbf{MOISTT} = \left.\frac{\partial T}{\partial t}\right|_c + \left.\frac{\partial T}{\partial t}\right|_{ls}$$

where:

$$\left.\frac{\partial T}{\partial t}\right|_c = R \sum_i \left(\alpha \frac{m_B}{c_p} \Gamma_s\right)_i \quad and \quad \left.\frac{\partial T}{\partial t}\right|_{ls} = \frac{L}{c_p}(q^* - q)$$

and

$$\Gamma_s = g\eta \frac{\partial s}{\partial p}$$

The subscript c refers to convective processes, while the subscript ls refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as s, the convective cloud base mass flux is given as m_B, and the cloud entrainment is given as η, which are explicitly defined in Section **??**, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as α, while R is the rain re-evaporation adjustment.

18) MOISTQ Specific Humidity Changes Due to Moist Processes ($g/kg/day$)

$$\mathbf{MOISTQ} = \left.\frac{\partial q}{\partial t}\right|_c + \left.\frac{\partial q}{\partial t}\right|_{ls}$$

where:

$$\left.\frac{\partial q}{\partial t}\right|_c = R \sum_i \left(\alpha \frac{m_B}{L}(\Gamma_h - \Gamma_s)\right)_i \quad and \quad \left.\frac{\partial q}{\partial t}\right|_{ls} = (q^* - q)$$

and

$$\Gamma_s = g\eta \frac{\partial s}{\partial p} \quad and \quad \Gamma_h = g\eta \frac{\partial h}{\partial p}$$

The subscript c refers to convective processes, while the subscript ls refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as s, the moist static energy is given as h, the convective cloud

base mass flux is given as m_B, and the cloud entrainment is given as η, which are explicitly defined in Section **??**, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as α, while R is the rain re-evaporation adjustment.

19) RADLW Heating Rate due to Longwave Radiation (*deg/day*)
The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, $F_{LW}^{clearsky}$, first. For a given cloud fraction, the clear line-of-sight probability $C(p,p')$ is computed from the current level pressure p to the model top pressure, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see Section **??**). The cloudy-sky flux is then obtained as:

$$F_{LW} = C(p,p') \cdot F_{LW}^{clearsky},$$

Finally, the net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F_{LW}^{NET},$$

or

$$\mathbf{RADLW} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F_{LW}^{NET}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, and

$$F_{LW}^{NET} = F_{LW}^{\uparrow} - F_{LW}^{\downarrow}$$

20) RADSW Heating Rate due to Shortwave Radiation (*deg/day*)
The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section **??**). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.
The heating rate due to Shortwave Radiation under cloudy skies is defined as:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F(cloudy)_{SW}^{NET} \cdot \text{RADSWT},$$

or

$$\mathbf{RADSW} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F(cloudy)_{SW}^{NET} \cdot \text{RADSWT}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, RADSWT is the true incident shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

$$F(cloudy)_{SW}^{Net} = F(cloudy)_{SW}^{\uparrow} - F(cloudy)_{SW}^{\downarrow}$$

21) PREACC Total (Large-scale + Convective) Accumulated Precipition (mm/day)
For a change in specific humidity due to moist processes, Δq_{moist}, the vertical integral or total precipitable amount is given by:

$$\textbf{PREACC} = \int_{surf}^{top} \rho \Delta q_{moist} dz = -\int_{surf}^{top} \Delta q_{moist} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{moist} dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day.

22) PRECON Convective Precipition (mm/day)
For a change in specific humidity due to sub-grid scale cumulus convective processes, Δq_{cum}, the vertical integral or total precipitable amount is given by:

$$\textbf{PRECON} = \int_{surf}^{top} \rho \Delta q_{cum} dz = -\int_{surf}^{top} \Delta q_{cum} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{cum} dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day.

23) TUFLUX Turbulent Flux of U-Momentum ($Newton/m^2$)
The turbulent flux of u-momentum is calculated for *diagnostic purposes only* from the eddy coefficient for momentum:

$$\textbf{TUFLUX} = \rho(\overline{u'w'}) = \rho(-K_m \frac{\partial U}{\partial z})$$

where ρ is the air density, and K_m is the eddy coefficient.

24) TVFLUX Turbulent Flux of V-Momentum ($Newton/m^2$)
The turbulent flux of v-momentum is calculated for *diagnostic purposes only* from the eddy coefficient for momentum:

$$\textbf{TVFLUX} = \rho(\overline{v'w'}) = \rho(-K_m \frac{\partial V}{\partial z})$$

where ρ is the air density, and K_m is the eddy coefficient.

25) TTFLUX Turbulent Flux of Sensible Heat (*Watts/m²*)
The turbulent flux of sensible heat is calculated for *diagnostic purposes only* from the eddy coefficient for heat and moisture:

$$\mathbf{TTFLUX} = c_p \rho P^{\kappa} (\overline{w'\theta'}) = c_p \rho P^{\kappa} (-K_h \frac{\partial \theta_v}{\partial z})$$

where ρ is the air density, and K_h is the eddy coefficient.

26) TQFLUX Turbulent Flux of Latent Heat (*Watts/m²*)
The turbulent flux of latent heat is calculated for *diagnostic purposes only* from the eddy coefficient for heat and moisture:

$$\mathbf{TQFLUX} = L\rho(\overline{w'q'}) = L\rho(-K_h \frac{\partial q}{\partial z})$$

where ρ is the air density, and K_h is the eddy coefficient.

27) CN Neutral Drag Coefficient (*dimensionless*)
The drag coefficient for momentum obtained by assuming a neutrally stable surface layer:

$$\mathbf{CN} = \frac{k}{\ln(\frac{h}{z_0})}$$

where k is the Von Karman constant, h is the height of the surface layer, and z_0 is the surface roughness.
NOTE: CN is not available through model version 5.3, but is available in subsequent versions.

28) WINDS Surface Wind Speed (*meter/sec*)
The surface wind speed is calculated for the last internal turbulence time step:

$$\mathbf{WINDS} = \sqrt{u^2_{Nrphys} + v^2_{Nrphys}}$$

where the subscript $Nrphys$ refers to the lowest model level.

29) DTSRF Air/Surface Virtual Temperature Difference (*deg K*)
The air/surface virtual temperature difference measures the stability of the surface layer:

$$\mathbf{DTSRF} = (\theta_{vNrphys+1} - \theta v_{Nrphys}) P^{\kappa}_{surf}$$

where

$$\theta_{vNrphys+1} = \frac{T_g}{P^{\kappa}_{surf}}(1+.609 q_{Nrphys+1}) \qquad \textit{and} \qquad q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

β is the surface potential evapotranspiration coefficient ($\beta = 1$ over oceans), $q^*(T_g, P_s)$ is the saturation specific humidity at the ground temperature and surface pressure, level $Nrphys$ refers to the lowest model level and level $Nrphys+1$ refers to the surface.

30) TG Ground Temperature ($deg\ K$)
The ground temperature equation is solved as part of the turbulence package using a backward implicit time differencing scheme:

$$\mathbf{TG}\ is\ obtained\ from:\ C_g \frac{\partial T_g}{\partial t} = R_{sw} - R_{lw} + Q_{ice} - H - LE$$

where R_{sw} is the net surface downward shortwave radiative flux, R_{lw} is the net surface upward longwave radiative flux, Q_{ice} is the heat conduction through sea ice, H is the upward sensible heat flux, LE is the upward latent heat flux, and C_g is the total heat capacity of the ground. C_g is obtained by solving a heat diffusion equation for the penetration of the diurnal cycle into the ground (Blackadar, 1977), and is given by:

$$C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{(0.386 + 0.536W + 0.15W^2)2x10^{-3}\frac{86400.}{2\pi}}\ .$$

Here, the thermal conductivity, λ, is equal to $2x10^{-3}\ \frac{ly}{sec}\frac{cm}{^oK}$, the angular velocity of the earth, ω, is written as 86400 sec/day divided by 2π $radians/day$, and the expression for C_s, the heat capacity per unit volume at the surface, is a function of the ground wetness, W.

31) TS Surface Temperature ($deg\ K$)
The surface temperature estimate is made by assuming that the model's lowest layer is well-mixed, and therefore that θ is constant in that layer. The surface temperature is therefore:

$$\mathbf{TS} = \theta_{Nrphys} P_{surf}^{\kappa}$$

32) DTG Surface Temperature Adjustment ($deg\ K$)
The change in surface temperature from one turbulence time step to the next, solved using the Ground Temperature Equation (see diagnostic number 30) is calculated:

$$\mathbf{DTG} = {T_g}^n - {T_g}^{n-1}$$

where superscript n refers to the new, updated time level, and the superscript $n-1$ refers to the value at the previous turbulence time level.

33) QG Ground Specific Humidity (g/kg)
The ground specific humidity is obtained by interpolating between the specific humidity at the lowest model level and the specific humidity of a saturated ground. The interpolation is performed using the potential evapotranspiration function:

$$\mathbf{QG} = q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

where β is the surface potential evapotranspiration coefficient ($\beta = 1$ over oceans), and $q^*(T_g, P_s)$ is the saturation specific humidity at the ground temperature and surface pressure.

34) QS Saturation Surface Specific Humidity (g/kg)
The surface saturation specific humidity is the saturation specific humidity at the ground temprature and surface pressure:

$$\mathbf{QS} = q^*(T_g, P_s)$$

35) TGRLW Instantaneous ground temperature used as input to the Longwave radiation subroutine (deg)

$$\mathbf{TGRLW} = T_g(\lambda, \phi, n)$$

where T_g is the model ground temperature at the current time step n.

36) ST4 Upward Longwave flux at the surface ($Watts/m^2$)

$$\mathbf{ST4} = \sigma T^4$$

where σ is the Stefan-Boltzmann constant and T is the temperature.

37) OLR Net upward Longwave flux at $p = p_{top}$ ($Watts/m^2$)

$$\mathbf{OLR} = F^{NET}_{LW,top}$$

where top indicates the top of the first model layer. In the GCM, $p_{top} = 0.0$ mb.

38) OLRCLR Net upward clearsky Longwave flux at $p = p_{top}$ ($Watts/m^2$)

$$\mathbf{OLRCLR} = F(clearsky)^{NET}_{LW,top}$$

where top indicates the top of the first model layer. In the GCM, $p_{top} = 0.0$ mb.

39) LWGCLR Net upward clearsky Longwave flux at the surface ($Watts/m^2$)

$$\begin{aligned} \mathbf{LWGCLR} &= F(clearsky)^{Net}_{LW,Nrphys+1} \\ &= F(clearsky)^{\uparrow}_{LW,Nrphys+1} - F(clearsky)^{\downarrow}_{LW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F(clearsky)^{\uparrow}_{LW}$ is the upward clearsky Longwave flux and the $F(clearsky)^{\downarrow}_{LW}$ is the downward clearsky Longwave flux.

40) LWCLR Heating Rate due to Clearsky Longwave Radiation (deg/day)

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, $F_{LW}^{clearsky}$, first. For a given cloud fraction, the clear line-of-sight probability $C(p,p')$ is computed from the current level pressure p to the model top pressure, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see Section **??**). The cloudy-sky flux is then obtained as:

$$F_{LW} = C(p,p') \cdot F_{LW}^{clearsky},$$

Thus, **LWCLR** is defined as the net longwave heating rate due to the vertical divergence of the clear-sky longwave radiative flux:

$$\frac{\partial \rho c_p T}{\partial t}_{clearsky} = -\frac{\partial}{\partial z} F(clearsky)_{LW}^{NET},$$

or

$$\mathbf{LWCLR} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F(clearsky)_{LW}^{NET}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, and

$$F(clearsky)_{LW}^{Net} = F(clearsky)_{LW}^{\uparrow} - F(clearsky)_{LW}^{\downarrow}$$

41) TLW Instantaneous temperature used as input to the Longwave radiation subroutine (deg)

$$\mathbf{TLW} = T(\lambda, \phi, level, n)$$

where T is the model temperature at the current time step n.

42) SHLW Instantaneous specific humidity used as input to the Longwave radiation subroutine (kg/kg)

$$\mathbf{SHLW} = q(\lambda, \phi, level, n)$$

where q is the model specific humidity at the current time step n.

43) OZLW Instantaneous ozone used as input to the Longwave radiation subroutine (kg/kg)

$$\mathbf{OZLW} = \mathrm{OZ}(\lambda, \phi, level, n)$$

where OZ is the interpolated ozone data set from the climatological monthly mean zonally averaged ozone data set.

44) CLMOLW Maximum Overlap cloud fraction used in LW Radiation $(0-1)$
CLMOLW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Longwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section **??**.

$$\textbf{CLMOLW} = CLMO_{RAS,LW}(\lambda, \phi, level)$$

45) CLDTOT Total cloud fraction used in LW and SW Radiation $(0-1)$
CLDTOT is the time-averaged total cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Longwave and Shortwave Radiation packages. For a complete description of cloud/radiative interactions, see Section **??**.

$$\textbf{CLDTOT} = F_{RAS} + F_{LS}$$

where F_{RAS} is the time-averaged cloud fraction due to sub-grid scale convection, and F_{LS} is the time-averaged cloud fraction due to precipitating and non-precipitating large-scale moist processes.

46) CLMOSW Maximum Overlap cloud fraction used in SW Radiation $(0-1)$
CLMOSW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Shortwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section **??**.

$$\textbf{CLMOSW} = CLMO_{RAS,SW}(\lambda, \phi, level)$$

47) CLROSW Random Overlap cloud fraction used in SW Radiation $(0-1)$
CLROSW is the time-averaged random overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Shortwave Radiation algorithm. These are convective and large-scale clouds whose radiative characteristics are not assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section **??**.

$$\textbf{CLROSW} = CLRO_{RAS,LargeScale,SW}(\lambda, \phi, level)$$

48) RADSWT Incident Shortwave radiation at the top of the atmosphere ($Watts/m^2$)

$$\mathbf{RADSWT} = \frac{S_0}{R_a^2} \cdot cos\phi_z$$

where S_0, is the extra-terrestial solar contant, R_a is the earth-sun distance in Astronomical Units, and $cos\phi_z$ is the cosine of the zenith angle. It should be noted that **RADSWT**, as well as **OSR** and **OSRCLR**, are calculated at the top of the atmosphere (p=0 mb). However, the **OLR** and **OLRCLR** diagnostics are currently calculated at $p = p_{top}$ (0.0 mb for the GCM).

49) EVAP Surface Evaporation (mm/day)
The surface evaporation is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

$$\mathbf{EVAP} = \rho\beta K_h(q_{surface} - q_{Nrphys})$$

where ρ = the atmospheric density at the surface, β is the fraction of the potential evapotranspiration actually evaporated ($\beta = 1$ over oceans), K_h is the turbulent eddy exchange coefficient for heat and moisture at the surface in m/sec and $qsurface$ and q_{Nrphys} are the specific humidity at the surface (see diagnostic number 34) and at the bottom model level, respectively.

50) DUDT Total Zonal U-Wind Tendency ($m/sec/day$)
DUDT is the total time-tendency of the Zonal U-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DUDT} = \frac{\partial u}{\partial t}_{Dynamics} + \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

51) DVDT Total Zonal V-Wind Tendency ($m/sec/day$)
DVDT is the total time-tendency of the Meridional V-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DVDT} = \frac{\partial v}{\partial t}_{Dynamics} + \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

52) DTDT Total Temperature Tendency (deg/day)
DTDT is the total time-tendency of Temperature due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\begin{aligned}\mathbf{DTDT} &= \frac{\partial T}{\partial t}_{Dynamics} + \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis}\end{aligned}$$

53) DQDT Total Specific Humidity Tendency $(g/kg/day)$
DQDT is the total time-tendency of Specific Humidity due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DQDT} = \frac{\partial q}{\partial t}_{Dynamics} + \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

54) USTAR Surface-Stress Velocity (m/sec)
The surface stress velocity, or the friction velocity, is the wind speed at the surface layer top impeded by the surface drag:

$$\mathbf{USTAR} = C_u W_s \qquad where: \;\; C_u = \frac{k}{\psi_m}$$

C_u is the non-dimensional surface drag coefficient (see diagnostic number 10), and W_s is the surface wind speed (see diagnostic number 28).
55) Z0 Surface Roughness Length (m)
Over the land surface, the surface roughness length is interpolated to the local time from the monthly mean data of Dorman and Sellers (1989). Over the ocean, the roughness length is a function of the surface-stress velocity, u_*.

$$\mathbf{Z0} = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + \frac{c_5}{u_*}$$

where the constants are chosen to interpolate between the reciprocal relation of Kondo(1975) for weak winds, and the piecewise linear relation of Large and Pond(1981) for moderate to large winds.

56) FRQTRB Frequency of Turbulence $(0-1)$
The fraction of time when turbulence is present is defined as the fraction of time when the turbulent kinetic energy exceeds some minimum value, defined here to be $0.005\ m^2/sec^2$. When this criterion is met, a counter is incremented. The fraction over the averaging interval is reported.

57) PBL Planetary Boundary Layer Depth (mb)
The depth of the PBL is defined by the turbulence parameterization to be the depth at which the turbulent kinetic energy reduces to ten percent of its surface value.

$$\mathbf{PBL} = P_{PBL} - P_{surface}$$

where P_{PBL} is the pressure in mb at which the turbulent kinetic energy reaches one tenth of its surface value, and P_s is the surface pressure.

58) SWCLR Clear sky Heating Rate due to Shortwave Radiation (deg/day)
The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section **??**). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.
The heating rate due to Shortwave Radiation under clear skies is defined as:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F(clear)_{SW}^{NET} \cdot \text{RADSWT},$$

or

$$\mathbf{SWCLR} = \frac{g}{c_p}\frac{\partial}{\partial p} F(clear)_{SW}^{NET} \cdot \text{RADSWT}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, RADSWT is the true incident shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

$$F(clear)_{SW}^{Net} = F(clear)_{SW}^{\uparrow} - F(clear)_{SW}^{\downarrow}$$

59) OSR Net upward Shortwave flux at the top of the model ($Watts/m^2$)

$$\mathbf{OSR} = F_{SW,top}^{NET}$$

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW_{top}} = 0$ mb.

60) OSRCLR Net upward clearsky Shortwave flux at the top of the model ($Watts/m^2$)

$$\mathbf{OSRCLR} = F(clearsky)_{SW,top}^{NET}$$

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW_{top}} = 0$ mb.

61) CLDMAS Convective Cloud Mass Flux (kg/m^2)
The amount of cloud mass moved per RAS timestep from all convective clouds is written:

$$\mathbf{CLDMAS} = \eta m_B$$

where η is the entrainment, normalized by the cloud base mass flux, and m_B is the cloud base mass flux. m_B and η are defined explicitly in Section **??**, the

description of the convective parameterization.

62) UAVE Time-Averaged Zonal U-Wind (m/sec)
The diagnostic **UAVE** is simply the time-averaged Zonal U-Wind over the **NUAVE** output frequency. This is contrasted to the instantaneous Zonal U-Wind which is archived on the Prognostic Output data stream.

$$\mathbf{UAVE} = u(\lambda, \phi, level, t)$$

Note, **UAVE** is computed and stored on the staggered C-grid.

63) VAVE Time-Averaged Meridional V-Wind (m/sec)
The diagnostic **VAVE** is simply the time-averaged Meridional V-Wind over the **NVAVE** output frequency. This is contrasted to the instantaneous Meridional V-Wind which is archived on the Prognostic Output data stream.

$$\mathbf{VAVE} = v(\lambda, \phi, level, t)$$

Note, **VAVE** is computed and stored on the staggered C-grid.

64) TAVE Time-Averaged Temperature ($Kelvin$)
The diagnostic **TAVE** is simply the time-averaged Temperature over the **NTAVE** output frequency. This is contrasted to the instantaneous Temperature which is archived on the Prognostic Output data stream.

$$\mathbf{TAVE} = T(\lambda, \phi, level, t)$$

65) QAVE Time-Averaged Specific Humidity (g/kg)
The diagnostic **QAVE** is simply the time-averaged Specific Humidity over the **NQAVE** output frequency. This is contrasted to the instantaneous Specific Humidity which is archived on the Prognostic Output data stream.

$$\mathbf{QAVE} = q(\lambda, \phi, level, t)$$

66) PAVE Time-Averaged Surface Pressure - PTOP (mb)
The diagnostic **PAVE** is simply the time-averaged Surface Pressure - PTOP over the **NPAVE** output frequency. This is contrasted to the instantaneous Surface Pressure - PTOP which is archived on the Prognostic Output data stream.

$$\begin{aligned} \mathbf{PAVE} &= \pi(\lambda, \phi, level, t) \\ &= p_s(\lambda, \phi, level, t) - p_T \end{aligned}$$

67) QQAVE Time-Averaged Turbulent Kinetic Energy $(m/sec)^2$
The diagnostic **QQAVE** is simply the time-averaged prognostic Turbulent Kinetic Energy produced by the GCM Turbulence parameterization over the **NQQAVE** output frequency. This is contrasted to the instantaneous Turbulent Kinetic Energy which is archived on the Prognostic Output data stream.

$$\mathbf{QQAVE} = qq(\lambda, \phi, level, t)$$

Note, **QQAVE** is computed and stored at the "mass-point" locations on the staggered C-grid.

68) SWGCLR Net downward clearsky Shortwave flux at the surface $(Watts/m^2)$

$$\begin{aligned} \mathbf{SWGCLR} &= F(clearsky)^{Net}_{SW,Nrphys+1} \\ &= F(clearsky)^{\downarrow}_{SW,Nrphys+1} - F(clearsky)^{\uparrow}_{SW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F(clearsky)SW^{\downarrow}$ is the downward clearsky Shortwave flux and $F(clearsky)^{\uparrow}_{SW}$ is the upward clearsky Shortwave flux.

69) SDIAG1 User-Defined Surface Diagnostic-1
The GCM provides Users with a built-in mechanism for archiving user-defined diagnostics. The generic diagnostic array QDIAG located in COMMON /DIAG/, and the associated diagnostic counters and pointers located in COMMON /DIAGP/, must be accessable in order to use the user-defined diagnostics (see Section 6.19.3). A convenient method for incorporating all necessary COMMON files is to include the GCM *vstate.com* file in the routine which employs the user-defined diagnostics.
In addition to enabling the user-defined diagnostic (ie., CALL SETDIAG(84)), the User must fill the QDIAG array with the desired quantity within the User's application program or within modified GCM subroutines, as well as increment the diagnostic counter at the time when the diagnostic is updated. The QDIAG location index for **SDIAG1** and its corresponding counter is automatically defined as **ISDIAG1** and **NSDIAG1**, respectively, after the diagnostic has been enabled. The syntax for its use is given by

```
do j=1,jm
do i=1,im
qdiag(i,j,ISDIAG1) = qdiag(i,j,ISDIAG1) + ...
enddo
enddo

NSDIAG1 = NSDIAG1 + 1
```

The diagnostics defined in this manner will automatically be archived by the output routines.

70) SDIAG2 User-Defined Surface Diagnostic-2
The GCM provides Users with a built-in mechanism for archiving user-defined diagnostics. For a complete description refer to Diagnostic #84. The syntax for using the surface SDIAG2 diagnostic is given by

```
do j=1,jm
do i=1,im
qdiag(i,j,ISDIAG2) = qdiag(i,j,ISDIAG2) + ...
enddo
enddo

NSDIAG2 = NSDIAG2 + 1
```

The diagnostics defined in this manner will automatically be archived by the output routines.

71) UDIAG1 User-Defined Upper-Air Diagnostic-1
The GCM provides Users with a built-in mechanism for archiving user-defined diagnostics. For a complete description refer to Diagnostic #84. The syntax for using the upper-air UDIAG1 diagnostic is given by

```
do L=1,Nrphys
do j=1,jm
do i=1,im
qdiag(i,j,IUDIAG1+L-1) = qdiag(i,j,IUDIAG1+L-1) + ...
enddo
enddo
enddo

NUDIAG1 = NUDIAG1 + 1
```

The diagnostics defined in this manner will automatically be archived by the output programs.

72) UDIAG2 User-Defined Upper-Air Diagnostic-2
The GCM provides Users with a built-in mechanism for archiving user-defined diagnostics. For a complete description refer to Diagnostic #84. The syntax for using the upper-air UDIAG2 diagnostic is given by

```
do L=1,Nrphys
do j=1,jm
do i=1,im
qdiag(i,j,IUDIAG2+L-1) = qdiag(i,j,IUDIAG2+L-1) + ...
enddo
enddo
```

```
enddo

NUDIAG2 = NUDIAG2 + 1
```

The diagnostics defined in this manner will automatically be archived by the output programs.

73) DIABU Total Diabatic Zonal U-Wind Tendency ($m/sec/day$)
DIABU is the total time-tendency of the Zonal U-Wind due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABU} = \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

74) DIABV Total Diabatic Meridional V-Wind Tendency ($m/sec/day$)
DIABV is the total time-tendency of the Meridional V-Wind due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABV} = \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

75) DIABT Total Diabatic Temperature Tendency (deg/day)
DIABT is the total time-tendency of Temperature due to Diabatic processes and the Analysis forcing.

$$\begin{aligned}\mathbf{DIABT} &= \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis}\end{aligned}$$

If we define the time-tendency of Temperature due to Diabatic processes as

$$\begin{aligned}\frac{\partial T}{\partial t}_{Diabatic} &= \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence}\end{aligned}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial T}{\partial t}_{Diabatic} = \frac{p^\kappa}{\pi}\frac{\partial \pi\theta}{\partial t}_{Diabatic}$$

where $\theta = T/p^\kappa$. Thus, **DIABT** may be written as

$$\mathbf{DIABT} = \frac{p^\kappa}{\pi}\left(\frac{\partial \pi\theta}{\partial t}_{Diabatic} + \frac{\partial \pi\theta}{\partial t}_{Analysis}\right)$$

76) DIABQ Total Diabatic Specific Humidity Tendency ($g/kg/day$)
DIABQ is the total time-tendency of Specific Humidity due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABQ} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

If we define the time-tendency of Specific Humidity due to Diabatic processes as

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{1}{\pi}\frac{\partial \pi q}{\partial t}_{Diabatic}$$

Thus, **DIABQ** may be written as

$$\mathbf{DIABQ} = \frac{1}{\pi}\left(\frac{\partial \pi q}{\partial t}_{Diabatic} + \frac{\partial \pi q}{\partial t}_{Analysis}\right)$$

77) VINTUQ Vertically Integrated Moisture Flux ($m/sec \cdot g/kg$)
The vertically integrated moisture flux due to the zonal u-wind is obtained by integrating uq over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTUQ} = \frac{\int_{surf}^{top} uq\rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g} = -\frac{1}{g}\delta p$, we have

$$\mathbf{VINTUQ} = \int_0^1 uqdp$$

78) VINTVQ Vertically Integrated Moisture Flux ($m/sec \cdot g/kg$)
The vertically integrated moisture flux due to the meridional v-wind is obtained by integrating vq over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTVQ} = \frac{\int_{surf}^{top} vq\rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g} = -\frac{1}{g}\delta p$, we have

$$\mathbf{VINTVQ} = \int_0^1 vqdp$$

79) VINTUT Vertically Integrated Heat Flux $(m/sec \cdot deg)$
The vertically integrated heat flux due to the zonal u-wind is obtained by integrating uT over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTUT} = \frac{\int_{surf}^{top} uT\rho dz}{\int_{surf}^{top} \rho dz}$$

Or,

$$\mathbf{VINTUT} = \int_0^1 uTdp$$

80) VINTVT Vertically Integrated Heat Flux $(m/sec \cdot deg)$
The vertically integrated heat flux due to the meridional v-wind is obtained by integrating vT over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTVT} = \frac{\int_{surf}^{top} vT\rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g}$, we have

$$\mathbf{VINTVT} = \int_0^1 vTdp$$

81 CLDFRC Total 2-Dimensional Cloud Fracton $(0-1)$

If we define the time-averaged random and maximum overlapped cloudiness as CLRO and CLMO respectively, then the probability of clear sky associated with random overlapped clouds at any level is (1-CLRO) while the probability of clear sky associated with maximum overlapped clouds at any level is (1-CLMO). The total clear sky probability is given by (1-CLRO)*(1-CLMO), thus the total cloud fraction at each level may be obtained by 1-(1-CLRO)*(1-CLMO).

At any given level, we may define the clear line-of-site probability by appropriately accounting for the maximum and random overlap cloudiness. The clear line-of-site probability is defined to be equal to the product of the clear line-of-site probabilities associated with random and maximum overlap cloudiness. The

clear line-of-site probability $C(p,p')$ associated with maximum overlap clouds, from the current pressure p to the model top pressure, $p' = p_{top}$, or the model surface pressure, $p' = p_{surf}$, is simply 1.0 minus the largest maximum overlap cloud value along the line-of-site, ie.

$$1 - MAX_p^{p'}\left(CLMO_p\right)$$

Thus, even in the time-averaged sense it is assumed that the maximum overlap clouds are correlated in the vertical. The clear line-of-site probability associated with random overlap clouds is defined to be the product of the clear sky probabilities at each level along the line-of-site, ie.

$$\prod_p^{p'}(1 - CLRO_p)$$

The total cloud fraction at a given level associated with a line- of-site calculation is given by

$$1 - \left(1 - MAX_p^{p'}\left[CLMO_p\right]\right)\prod_p^{p'}(1 - CLRO_p)$$

The 2-dimensional net cloud fraction as seen from the top of the atmosphere is given by

$$\mathbf{CLDFRC} = 1 - \left(1 - MAX_{l=l_1}^{Nrphys}\left[CLMO_l\right]\right)\prod_{l=l_1}^{Nrphys}(1 - CLRO_l)$$

For a complete description of cloud/radiative interactions, see Section **??**.

82) QINT Total Precipitable Water (gm/cm^2)

The Total Precipitable Water is defined as the vertical integral of the specific humidity, given by:

$$\begin{aligned}\mathbf{QINT} &= \int_{surf}^{top}\rho q dz \\ &= \frac{\pi}{g}\int_0^1 q dp\end{aligned}$$

where we have used the hydrostatic relation $\rho\delta z = -\frac{\delta p}{g}$.

83) U2M Zonal U-Wind at 2 Meter Depth (m/sec)

The u-wind at the 2-meter depth is determined from the similarity theory:

$$\mathbf{U2M} = \frac{u_*}{k}\psi_{m_{2m}}\frac{u_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}}u_{sl}$$

where $\psi_m(2m)$ is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness

height is above two meters, **U2M** is undefined.

84) V2M Meridional V-Wind at 2 Meter Depth (m/sec)
The v-wind at the 2-meter depth is a determined from the similarity theory:

$$\mathbf{V2M} = \frac{u_*}{k}\psi_{m_{2m}}\frac{v_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}}v_{sl}$$

where $\psi_m(2m)$ is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, **V2M** is undefined.

85) T2M Temperature at 2 Meter Depth ($deg\ K$)
The temperature at the 2-meter depth is a determined from the similarity theory:

$$\mathbf{T2M} = P^{\kappa}(\frac{\theta*}{k}(\psi_{h_{2m}} + \psi_g) + \theta_{surf}) = P^{\kappa}(\theta_{surf} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(\theta_{sl} - \theta_{surf}))$$

where:

$$\theta_* = -\frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, **T2M** is undefined.

86) Q2M Specific Humidity at 2 Meter Depth (g/kg)
The specific humidity at the 2-meter depth is determined from the similarity theory:

$$\mathbf{Q2M} = P^{\kappa}(\frac{q_*}{k}(\psi_{h_{2m}} + \psi_g) + q_{surf}) = P^{\kappa}(q_{surf} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(q_{sl} - q_{surf}))$$

where:

$$q_* = -\frac{(\overline{w'q'})}{u_*}$$

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, **Q2M** is undefined.

87) U10M Zonal U-Wind at 10 Meter Depth (m/sec)
The u-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

$$\mathbf{U10M} = \frac{u_*}{k}\psi_{m_{10m}}\frac{u_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}}u_{sl}$$

where $\psi_m(10m)$ is the non-dimensional wind shear at ten meters, and the subscript sl refers to the height of the top of the surface layer.

88) V10M Meridional V-Wind at 10 Meter Depth (m/sec)
The v-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

$$\mathbf{V10M} = \frac{u_*}{k}\psi_{m_{10m}}\frac{v_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}}v_{sl}$$

where $\psi_m(10m)$ is the non-dimensional wind shear at ten meters, and the subscript sl refers to the height of the top of the surface layer.

89) T10M Temperature at 10 Meter Depth ($deg\ K$)
The temperature at the 10-meter depth is an interpolation between the surface potential temperature and the model lowest level potential temperature using the ratio of the non-dimensional temperature gradient at the two levels:

$$\mathbf{T10M} = P^{\kappa}(\frac{\theta*}{k}(\psi_{h_{10m}} + \psi_g) + \theta_{surf}) = P^{\kappa}(\theta_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(\theta_{sl} - \theta_{surf}))$$

where:

$$\theta_* = -\frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer.

90) Q10M Specific Humidity at 10 Meter Depth (g/kg)
The specific humidity at the 10-meter depth is an interpolation between the surface specific humidity and the model lowest level specific humidity using the ratio of the non-dimensional temperature gradient at the two levels:

$$\mathbf{Q10M} = P^{\kappa}(\frac{q_*}{k}(\psi_{h_{10m}} + \psi_g) + q_{surf}) = P^{\kappa}(q_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(q_{sl} - q_{surf}))$$

where:

$$q_* = -\frac{(\overline{w'q'})}{u_*}$$

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer.

91) DTRAIN Cloud Detrainment Mass Flux (kg/m^2)
The amount of cloud mass moved per RAS timestep at the cloud detrainment level is written:

$$\mathbf{DTRAIN} = \eta_{r_D} m_B$$

where r_D is the detrainment level, m_B is the cloud base mass flux, and η is the entrainment, defined in Section **??**.

92) QFILL Filling of negative Specific Humidity ($g/kg/day$)
Due to computational errors associated with the numerical scheme used for the advection of moisture, negative values of specific humidity may be generated. The specific humidity is checked for negative values after every dynamics timestep. If negative values have been produced, a filling algorithm is invoked which redistributes moisture from below. Diagnostic **QFILL** is equal to the net filling needed to eliminate negative specific humidity, scaled to a per-day rate:

$$\mathbf{QFILL} = q^{n+1}_{final} - q^{n+1}_{initial}$$

where

$$q^{n+1} = (\pi q)^{n+1}/\pi^{n+1}$$

6.19.5 Dos and Donts

6.19.6 Diagnostics Reference

6.20 RW Basic binary I/O utilities

The `rw` package provides a very rudimentary binary I/O capability for quickly writing *single record* direct-access Fortran binary files. It is primarily used for writing diagnostic output.

6.20.1 Introduction

Package `rw` is an interface to the more general `mdsio` package. The `rw` package can be used to write or read direct-access Fortran binary files for two-dimensional XY and three-dimensional XYZ arrays. The arrays are assumed to have been declared according to the standard MITgcm two-dimensional or three-dimensional floating point array type:

```
C     Example of declaring a standard two dimensional "long"
C     floating point type array (the _RL macro is usually
C     mapped to 64-bit floats in most configurations)
      _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
```

Each call to an `rw` read or write routine will read (or write) to the first record of a file. To write direct access Fortran files with multiple records use the package `mdsio` (see section 6.16). To write self-describing files that contain embedded information describing the variables being written and the spatial and temporal locations of those variables use the package `mnc` (see section 6.15) which produces netCDF[1] [44] based output.

[1] `http://www.unidata.ucar.edu/packages/netcdf`

6.21 FFT Filtering Code

6.21.1 Key subroutines, parameters and files

6.22 GCHEM Package

6.22.1 Introduction

This package has been developed as interface to the PTRACERS package. The purpose is to provide a structure where various (any) tracer experiments can be added to the code. For instance there are placeholders for routines to read in parameters needed for any tracer experiments, a routine to read in extra fields required for the tracer code, routines for either external forcing or internal interactions between tracers and routines for additional diagnostics relating to the tracers. Note that the gchem package itself is only a means to call the subroutines used by specific biogeochemical experiments, and does not "do" anything on its own.

There are two examples: cfc which looks at 2 tracers with a simple external forcing and dic with 5 tracers whose tendency terms are related to one another. We will discuss these here only as how they provide examples to use this package.

6.22.2 Key subroutines and parameters

FRAMEWORK
GCHEM_OPTIONS.h includes the compiler options to be used in any experiment. For instance #define ALLOW_CFC allows the CFC code to be run. An important compiler option is #define GCHEM_SEPARATE_FORCING which determined how and when the tracer forcing is applied (see discussion on Forcing below). There are further runtime parameters set in *data.gchem* and kept in common block *GCHEM.h*. These runtime options include:
• **tIter0** which is the integer timestep when the tracer experiment is initialized. If **nIter0** = **tIter0** then the tracers are initialized to zero or from initial files. If **nIter0** > **tIter0** then tracers (and previous timestep tendency terms) are read in from a the ptracers pickup file. Note that tracers of zeros will be carried around if **nIter0** < **tIter0**.
• **nsubtime** is the integer number of extra timesteps required by the tracer experiment. This will give a timestep of **deltaTtracer**/**nsubtime** for the dependencies between tracers. The default is one.
• File names - these are several filenames than can be read in for external fields needed in the tracer forcing - for instance wind speed is needed in both DIC and CFC packages to calculate the air-sea exchange of gases. Not all file names will be used for every tracer experiment.

INITIALIZATION
The values set at runtime in data.gchem are read in using *gchem_readparms.F* which is called from packages_readparms.F. This will include any external forcing files that will be needed by the tracer experiment.

There are two routine used to initialize parameters and fields needed by the experiment packages. These are *gchem_init_fixed.F* which is called from *packages_init_fixed.F*, and *gchem_init_vari.F* called from packages_init_variable.F. The

first should be used to call a subroutine specific to the tracer experiment which sets fixed parameters, the second should call a subroutine specific to the tracer experiment which sets (or initializes) time fields that will vary with time.

LOADING FIELDS
External forcing fields used by the tracer experiment are read in by a subroutine (specific to the tracer experiment) called from *gchem_fields_load.F*. This latter is called from *forward_step.F*.

FORCING
Tracer fields are advected-and-diffused by the ptracer package. Additional changes (e.g. surface forcing or interactions between tracers) to these fields are taken care of by the gchem interface. For tracers that are essentially passive (e.g. CFC's) but may have some surface boundary conditions this can easily be done within the regular tracer timestep. In this case *gchem_calc_tendency.F* is called from *forward_step.F*, where the reactive (as opposed to the advective diffusive) tendencies are computed. These tendencies, stored on the 3D field **gchemTendency**, are added to the passive tracer tendencies **gPtr** in *gchem_add_tendency.F*, which is called from *ptracers_forcing.F*. For tracers with more complicated dependencies on each other, and especially tracers which require a smaller timestep than deltaTtracer, it will be easier to use *gchem_forcing_sep.F* which is called from forward_step.F. There is a compiler option set in *GCHEM_OPTIONS.h* that determines which method is used: #define GCHEM_SEPARATE_FORCING does the latter where tracers are forced separately from the advection-diffusion code, and #undef GCHEM_SEPARATE_FORCING includes the forcing in the regular timestepping.

DIAGNOSTICS
This package also also used the passive tracer routine *ptracers_monitor.F* which prints out tracer statistics as often as the model dynamic statistic diagnostics (dynsys) are written (or as prescribed by the runtime flag **PTRACERS_monitorFreq**, set in *data.ptracers*). There is also a placeholder for any tracer experiment specific diagnostics to be calculated and printed to files. This is done in *gchem_diags.F*. For instance the time average CO2 air-sea fluxes, and sea surface pH (among others) are written out by *dic_biotic_diags.F* which is called from *gchem_diags.F*.

6.22.3 Do's and Don'ts

The pkg ptracer is required with use with this pkg. Also, as usual, the runtime flag **useGCHEM** must be set to **.TRUE.** in **data.pkg**. By itself, gchem pkg will read in **data.gchem** and will write out gchem diagnostics. It requires tracer experiment specific calls to do anything else (for instance the calls to dic and cfc pkgs).

6.22.4 Reference Material

Chapter 7

Diagnostics and tools

There are numerous tools for pre-processing data, converting model output and analysis written in Matlab, fortran (f77 and f90) and perl. As yet they remain undocumented although many are self-documenting (Matlab routines have "help" written into them).

Here we'll summarize what is available but this is an ever growing resource so this may not cover everything that is out there:

7.1 Utilities supplied with the model

We supply some basic scripts with the model to facilitate conversion or reading of data into analysis software.

7.1.1 utils/scripts

In the directory *utils/scripts* you will find *joinds* and *joinmds*: these are perl scripts used from joining the multi-part files created by MITgcm. **Use *joinmds* always**. You will only need *joinds* if you are working with output older than two years (prior to c23).

7.1.2 utils/matlab

In the directory *utils/matlab* you will find several Matlab scripts (.m or dot-em files). The priniciple script is *rdmds.m* used for reading the multi-part model output files in to matlab. Place the scripts in your matlab path or change the path appropriately, then at the matlab prompt type:

```
>> help rdmds
```

to get help on how to use rdmds.

Another useful script scans the terminal output file for "monitor" information.

Most other scripts are for working in the curvilinear coordinate systems which as yet are unpublished and undocumented.

7.2 Pre-processing software

There is a suite of pre-processing software for intepolating bathymetry and forcing data, written by Adcroft and Biastoch. At some point, these will be made available for download. If you are in need of such software, contact one of them.

Chapter 8

Interface with ECCO

This chapter describes the interface between MITgcm and the large scale optimization schemes employed in the ECCO project. The interface provides the so-called "ECCO environment", which is maintained alongside the kernel MITgcm code.

8.1 The ECCO state estimation cost function DRAFT!!!

The current ECCO state estimation covers an $nYears = 11$ year model trajectory. A variety of data sets enter a least squares cost function, in addition to penalty terms which constrain deviations of control variables beyound their a priori errors.

8.1.1 Sea surface height from TOPEX/Poseidon and ERS-1/2 altimetry

Altimetric SSH contributions from T/P and ERS-1/2 are four-fold:

1. an $nYears$ time mean SSH misfit between model and T/P
2. daily SSH anomaly misfits between T/P and model
3. daily SSH anomaly misfits between ERS-1/2 and model
4. daily absolute SSH misfit between T/P and model, weighted by the full geoid error covariance.

8.1.1.1 Input fields

field	file name	deccription	unit
psbar	`psbarfile`	daily model mean SSH fields	[m]
tpmean	`topexmeanfile`	$nYears$ T/P mean	[cm]
tpobs	`topexfile`	daily T/P SSH anomalies	[cm]
erspobs	`ersfile`	daily ERS-1/2 SSH anomalies	[cm]
wp	`geoid_errfile`	diagonal of geoid error covariance	[m]
wtp, wers	`ssh_errfile`	rms of SSH anomalies	[cm]

8.1.1.2 *nYears* time mean SSH misfit

1. Compute $nYears$ model mean spatial distribution

$$psmean(i,j) = \frac{1}{nDaysRec} \sum_{i=1}^{nDaysRec} psbar(i,j) \tag{8.1}$$

2. Compute global offset between $nYears$ model and T/P mean:

$$\begin{aligned} offset &= \overline{tpmean} - \overline{psmean} \\ &= \frac{1}{normaliz.} \sum_{i,j} \{tpmean(i,j) - psmean(i,j)\} \cdot cosphi(i,j) \cdot tpmeanmask(i,j) \end{aligned} \tag{8.2}$$

3. Misfits are computed w.r.t. global $offset$.
   First spatial distribution:

$$\begin{aligned} cost_ssh_mean(i,j) &= \frac{1}{wp^2} \left\{ \left[psmean(i,j) - \overline{psmean} \right] - \left[tpmean(i,j) - \overline{tpmean} \right] \right\}^2 \\ &= \frac{1}{wp^2} \left\{ psmean(i,j) - tpmean(i,j) + offset \right\}^2 \end{aligned} \tag{8.3}$$

   Finally, sum over all spatial entries:

$$\overline{cost_ssh_mean} = \sum_{i,j} cost_ssh_mean(i,j) \tag{8.4}$$

8.1.1.3 Misfit of daily SSH anomalies

Computation is same for T/P and ERS-1/2. Here we write out computation for T/P.

1. Compute difference in anomalies:

$$cost_ssh_anom(i,j,t) = \frac{1}{wtp^2}\{\,[\,psbar(i,j,t) - psmean(i,j)\,] - [\,tpobs(i,j,t)\,]\,\}^2 \tag{8.5}$$

   where t denotes time (day) index, and where it is assumed that $nYears$ mean T/P spatial distribution $tpmean(i,j)$ has already been removed from data $tpobs(i,j)$!

2. Sum over all spatial points and all times

$$\overline{cost_ssh_anom} = \sum_t \sum_{i,j} cost_ssh_anom(i,j,t) \tag{8.6}$$

8.1.1.4 Flow chart

```
cost_ssh
|
|- < compute nYears model mean >
|
|- < read nYears T/P mean >
|  CALL COST_READTOPEXMEAN
|
|- < compute global T/P vs. model offset >
|
|- < compute cost_hmean >
|  CALL COST_SSH_MEAN
|
|- < ... >
```

8.1.1.5 Weights and notes

- All data are currently masked to zero where less than 13 depth levels, mimicing no contribution for depth less than 1000m.
- *cosphi* term in weights is set to 1.
- bad T/P and ERS-1/2 values are flagged ≤ -9990.
- T/P and ERS-1/2 data $\leq 1.\exp^{-8}$ cm are flagged as bad values
- wp is read from `geoid_errfile` and $1/wp^2$ is pre-computed in `ecco_cost_weights`

wp **for SSH mean misfit**
$1/wp^2$ is pre-computed in `ecco_cost_weights`;
wp is read from `geoid_errfile`;

wtp **and** *wers* **for SSH anomaly misfit**
$1/wtp^2$, $1/wers^2$ are pre-computed in `ecco_cost_weights`;

- *wtp*, *wers* are read from single `ssh_errfile`
- both are converted to meters and halved
  $wtp \longrightarrow wtp \cdot 0.01 \cdot 0.5$
- ERS error is set to T/P error + 5cm
  $wers = wtp + 0.5cm$

8.1.1.6 Cost diagnostics

- Map out $cost_ssh_mean(i, j)$
- Map out $cost_ssh_anom(i, j, t)$ averaged over 1 month, i.e.

$$\frac{1}{\text{monthly entries}} \sum_{t}^{monthly} cost_ssh_anom(i, j, t)$$

- sum over daily entries and plot daily average as function of time. i.e.

$$\frac{1}{\text{daily entries}} \sum_{i,j} cost_ssh_anom(i, j, t)$$

8.1.2 Hydrographic constraints

Observation of temperature and salinity from various sources are used to constrain the model. These are:

1. CTD obs. for T, S from various WOCE sections
2. XBT obs. for T
3. Sea surface temperature (SST) and salinity (SSS) from Reynolds et al. (???)
4. T, S from ARGO floats
5. T, S from fields from Levitus (???)

8.1.2.1 Input fields

field	file name	deccription	unit
tbar	`tbarfile`	monthly model mean pot. temperature	[°C]
sbar	`sbarfile`	monthly model mean salinity	[ppt]
tdat	`tdatfile`	monthly mean Levitus pot. temperature	[°C]
sdat	`sdatfile`	monthly mean Levitus salinity	[ppt]
ctdtobs	`ctdtfile`	monthly WOCE CTD pot. temperature	[°C]
ctdsobs	`ctdsfile`	monthly WOCE CTD salinity	[ppt]
xbtobs	`xbtfile`	monthly XBT in-situ(!) temperature	[°C]
sstdat	`sstdatfile`	monthly Reynolds pot. SST	[°C]
sssdat	`sssdatfile`	monthly Reynolds SSS	[ppt]
argotobs	`argotfile`	monthly ARGO in-situ(!) temperature	[°C]
argosobs	`argosfile`	monthly ARGO salinity	[ppt]
wti, wsi	`data_errfile`	vert. stdev. profile for T, S	
wtvar	`temperrfile`	spatially varying stdev.	[°C]
wsvar	`salterrfile`	spatially varying stdev.	[ppt]

8.1.2.2 XBT data

$$cost_xbt_t(i,j,k) = \left[\frac{fac \cdot ratio}{wti^2 + wtvar^2} \sum_{\tau=1}^{nMonsRec} \{Tbar(\tau) - T2\theta[xbtobs(\tau)]\}^2 \right](i,j,k) \tag{8.7}$$

8.1.2.3 WOCE CTD data

$$\begin{aligned} cost_ctd_t(i,j,k) &= \left[\frac{fac \cdot ratio}{wti^2 + wtvar^2} \sum_{\tau=1}^{nMonsRec} \{Tbar(\tau) - ctdTobs(\tau)\}^2 \right](i,j,k) \\ cost_ctd_s(i,j,k) &= \left[\frac{fac \cdot ratio}{wsi^2 + wsvar^2} \sum_{\tau=1}^{nMonsRec} \{Sbar(\tau) - ctdSobs(\tau)\}^2 \right](i,j,k) \end{aligned} \tag{8.8}$$

8.1.2.4 ARGO float data

$$\begin{aligned} cost_argo_t(i,j,k) &= \left[\frac{fac \cdot ratio}{wti^2 + wvar^2} \sum_{\tau=1}^{nMonsRec} \{Tbar(\tau) - T2\theta[argoTobs(\tau)]\}^2 \right](i,j,k) \\ cost_argo_s(i,j,k) &= \left[\frac{fac \cdot ratio}{wsi^2 + wsvar^2} \sum_{\tau=1}^{nMonsRec} \{Sbar(\tau) - argoSobs(\tau)\}^2 \right](i,j,k) \end{aligned} \tag{8.9}$$

8.1.2.5 Reynolds sea surface T, S data

$$\begin{aligned} cost_sst(i,j) &= \left[wsst \sum_{\tau=1}^{nMonsRec} \{Tbar(\tau) - sstDat(\tau)\}^2 \right](i,j) \\ cost_sss(i,j) &= \left[wsss \sum_{\tau=1}^{nMonsRec} \{Sbar(\tau) - sssDat(\tau)\}^2 \right](i,j) \end{aligned} \tag{8.10}$$

8.1.2.6 Levitus montly T, S climatological data

Model vs. data misfits are taken from $nYears$ monthly model means vs. Levitus monthly data. The description below is for potential temperature. Procedure for salinity is fully analogous. Spatial indices (i, j, k) are omitted throughout.

1. Compute $nYears$ monthly model means for each month $imon$:

$$\overline{Tbar}(imon) = \frac{1}{nYears} \sum_{iyear=1}^{nYears} Tbar(iyear, imon)$$

2. Compute misfit:

$$cost_theta(i,j,k) = \left[\frac{fac \cdot ratio}{wti^2} \sum_{imon=1}^{12} \left\{ \overline{Tbar}(imon) - Tdat(imon) \right\}^2 \right](i,j,k)$$

8.1.2.7 Weights and notes

- $T2\theta$ is an operator mapping in-situ to potential temperatures
- Latitudinal weight not used:

$$cosphi(i,j) = 1$$

- $fac = cosphi \cdot mask$
- Spatially *constant* weights:
  1. Read standard deviation vertical profiles for T, S
     `data_errfile` $\longrightarrow$ $wti(k)$, $wsi(k)$
     `data_errfile` $\longrightarrow$ $ratio = 0.25 = \left(\frac{1}{2}\right)^2$
  2. Take inverse squares:

$$\begin{aligned} wtheta(k) &= \frac{ratio}{wti(k)^2} \\ wsalt(k) &= \frac{ratio}{wsi(k)^2} \end{aligned}$$

- Spatially *varying* weights:

1. Read standard deviation fields
   `temperrfile` $\longrightarrow$ $wtvar(i,j,k)$
   `salterrfile` $\longrightarrow$ $wsvar(i,j,k)$

2. Weights are combination of spatially constant and varying parts:

$$wtheta2(i,j,k) = \frac{ratio}{wti(k)^2 + wtvar(i,j,k)^2}$$
$$wsalt2(i,j,k) = \frac{ratio}{wsi(k)^2 + wsvar(i,j,k)^2}$$

- Sea surface T, S weights:
  - SST: $wsst = fac \cdot wtheta(1)$: horizontally constant
  - SSS: $wsss = fac \cdot wsalt2(i,j,1)$: horizontally varying

  (Why this difference? I don't know.)

8.1.2.8 Diagnostics

- Map out $wtheta2(i,j,k)$, $wsalt2(i,j,k)$.

8.2 The external forcing package exf

8.2.1 Summary

```
c     Field definitions, units, and sign conventions:
c     ===============================================
c
c     ustress   :: Zonal surface wind stress in N/m^2
c                  > 0 for increase in uVel, which is west to
c                      east for cartesian and spherical polar grids
c                  Typical range: -0.5 < ustress < 0.5
c                  Southwest C-grid U point
c                  Input field
c
c     vstress   :: Meridional surface wind stress in N/m^2
c                  > 0 for increase in vVel, which is south to
c                      north for cartesian and spherical polar grids
c                  Typical range: -0.5 < vstress < 0.5
c                  Southwest C-grid V point
c                  Input field
c
c     hflux     :: Net upward surface heat flux excluding shortwave in W/m^2
c                  hflux = latent + sensible + lwflux
c                  > 0 for decrease in theta (ocean cooling)
c                  Typical range: -250 < hflux < 600
c                  Southwest C-grid tracer point
c                  Input field
c
c     sflux     :: Net upward freshwater flux in m/s
c                  sflux = evap - precip - runoff
c                  > 0 for increase in salt (ocean salinity)
c                  Typical range: -1e-7 < sflux < 1e-7
c                  Southwest C-grid tracer point
c                  Input field
c
c     swflux    :: Net upward shortwave radiation in W/m^2
c                  swflux = - ( swdown - ice and snow absorption - reflected )
c                  > 0 for decrease in theta (ocean cooling)
c                  Typical range: -350 < swflux < 0
c                  Southwest C-grid tracer point
c                  Input field
c
c     uwind     :: Surface (10-m) zonal wind velocity in m/s
c                  > 0 for increase in uVel, which is west to
c                      east for cartesian and spherical polar grids
c                  Typical range: -10 < uwind < 10
c                  Southwest C-grid U point
c                  Input or input/output field
c
c     vwind     :: Surface (10-m) meridional wind velocity in m/s
c                  > 0 for increase in vVel, which is south to
c                      north for cartesian and spherical polar grids
c                  Typical range: -10 < vwind < 10
c                  Southwest C-grid V point
c                  Input or input/output field
c
c     atemp     :: Surface (2-m) air temperature in deg K
```

```
c                  Typical range: 200 < atemp < 300
c                  Southwest C-grid tracer point
c                  Input or input/output field
c
c     aqh       :: Surface (2m) specific humidity in kg/kg
c                  Typical range: 0 < aqh < 0.02
c                  Southwest C-grid tracer point
c                  Input or input/output field
c
c     lwflux    :: Net upward longwave radiation in W/m^2
c                  lwflux = - ( lwdown - ice and snow absorption - emitted )
c                  > 0 for decrease in theta (ocean cooling)
c                  Typical range: -20 < lwflux < 170
c                  Southwest C-grid tracer point
c                  Input field
c
c     evap      :: Evaporation in m/s
c                  > 0 for increase in salt (ocean salinity)
c                  Typical range: 0 < evap < 2.5e-7
c                  Southwest C-grid tracer point
c                  Input, input/output, or output field
c
c     precip    :: Precipitation in m/s
c                  > 0 for decrease in salt (ocean salinity)
c                  Typical range: 0 < precip < 5e-7
c                  Southwest C-grid tracer point
c                  Input or input/output field
c
c     runoff    :: River and glacier runoff in m/s
c                  > 0 for decrease in salt (ocean salinity)
c                  Typical range: 0 < runoff < ????
c                  Southwest C-grid tracer point
c                  Input or input/output field
c                  !!! WATCH OUT: Default exf_inscal_runoff !!!
c                  !!! in exf_readparms.F is not 1.0        !!!
c
c     swdown    :: Downward shortwave radiation in W/m^2
c                  > 0 for increase in theta (ocean warming)
c                  Typical range: 0 < swdown < 450
c                  Southwest C-grid tracer point
c                  Input/output field
c
c     lwdown    :: Downward longwave radiation in W/m^2
c                  > 0 for increase in theta (ocean warming)
c                  Typical range: 50 < lwdown < 450
c                  Southwest C-grid tracer point
c                  Input/output field
c
c     apressure :: Atmospheric pressure field in N/m^2
c                  > 0 for ????
c                  Typical range: ???? < apressure < ????
c                  Southwest C-grid tracer point
c                  Input field
C
C
c     NOTES:
c     ======
```

```
c
c     Input and output units and sign conventions can be customized
c     using variables exf_inscal_* and exf_outscal_*, which are set
c     by exf_readparms.F
c
c     Output fields fu, fv, Qnet, Qsw, and EmPmR are
c     defined in FFIELDS.h
c
c     #ifndef SHORTWAVE_HEATING, hflux includes shortwave,
c     that is, hflux = latent + sensible + lwflux +swflux
c
c     If (EXFwindOnBgrid .EQ. .TRUE.), uwind and vwind are
c     defined on northeast B-grid U and V points, respectively.
c
c     Arrays *0 and *1 below are used for temporal interpolation.
```

8.3 The calendar package cal

Christian Eckert, MIT/EAPS, May-2000

This calendar tool was originally intended to enable the use of absolute dates (Gregorian Calendar dates) in the ocean general circulation model MITgcmuv. There is, however, a fair amount of routines that can be used independently of the MITgcmuv. After some minor modifications the whole package can be used either as a stand-alone calendar or in connection with any dynamical model that needs calendar dates. Some straightforward extensions are still pending e.g. the availability of the Julian Calendar, to be able to resolve fractions of a second, and to have a time- step that is longer than one day.

8.3.1 Basic assumptions for the calendar tool

It is assumed that the SMALLEST TIME INTERVAL to be resolved is ONE SECOND.

Further assumptions are that there is an INTEGER NUMBER OF MODEL STEPS EACH DAY, and that AT LEAST ONE STEP EACH DAY is made.

Not each individual routine depends on these assumptions; there are only a few places where they enter.

8.3.2 Format of calendar dates

In this calendar tool a complete date specification is defined as the following integer array:

```
c           integer date(4)
c
c           ( yyyymmdd, hhmmss, leap_year, dayofweek )
c
c             date(1) = yyyymmdd    <-- Year-Month-Day
c             date(2) =   hhmmss    <-- Hours-Minutes-Seconds
c             date(3) = leap_year   <-- Leap Year/No Leap Year
c             date(4) = dayofweek   <-- Day of the Week
c
c             leap_year is either equal to 1 (normal year)
c                              or equal to 2 (leap year)
c
c             dayofweek has a range of 1 to 7.
```

In case the Gregorian Calendar is used, the first day of the week is Friday, since day of the Gregorian Calendar was Friday, 15 Oct. 1582. As a date array this date would be specified as

```
c               refdate(1) = 15821015
c               refdate(2) =        0
c               refdate(3) =        1
c               refdate(4) =        1
```

8.3.3 Calendar dates and time intervals

Subtracting calendar dates yields time intervals. Time intervals have the following format:

```
c          integer datediff(4)
c
c            datediff(1) = # Days
c            datediff(2) = hhmmss
c            datediff(3) =      0
c            datediff(4) =     -1
```

Such time intervals can be added to or can be subtracted from calendar dates. Time intervals can be added to and be subtracted from each other.

8.3.4 Using the calendar together with MITgcm

Each routine has as an argument the thread number that it is belonging to, even if this number is not used in the routine itself.

In order to include the calendar tool into the MITgcm setup the MITgcm subroutine "initialise.F" or the routine "initilise_fixed.F", depending on the MITgcm release, has to be modified in the following way:

```
c         #ifdef ALLOW_CALENDAR
c         C--   Initialise the calendar package.
c         #ifdef USE_CAL_NENDITER
c               CALL cal_Init(
c              I               startTime,
c              I               endTime,
c              I               deltaTclock,
c              I               nIter0,
c              I               nEndIter,
c              I               nTimeSteps,
c              I               myThid
c              &             )
c         #else
c               CALL cal_Init(
c              I               startTime,
c              I               endTime,
c              I               deltaTclock,
c              I               nIter0,
c              I               nTimeSteps,
c              I               myThid
c              &             )
c         #endif
c               _BARRIER
c         #endif
```

It is useful to have the CPP flag ALLOW_CALENDAR in order to switch from the usual MITgcm setup to the one that includes the calendar tool. The CPP flag USE_CAL_NENDITER has been introduced in order to enable the use of the calendar for MITgcm releases earlier than checkpoint 25 which do not have the global variable *nEndIter*.

8.3.5 The individual calendars

8.3.5.1 Simple model calendar

This calendar can be used by defining

```
c                   TheCalendar='model'
```

in the calendar's data file "data.cal".

In this case a year is assumed to have 360 days. The model year is divided into 12 months with 30 days each.

8.3.5.2 Gregorian Calendar

This calendar can be used by defining

```
c                   TheCalendar='gregorian'
```

in the calendar's data file "data.cal".

8.3.6 Short routine description

```
c      o  cal_Init          - Initialise the calendar. This is the interface
c                             to the MITgcm.
c
c      o  cal_Set           - Sets the calendar according to the user
c                             specifications.
c
c      o  cal_GetDate       - Given the model's current timestep or the
c                             model's current time return the corresponding
c                             calendar date.
c
c      o  cal_FullDate      - Complete a date specification (leap year and
c                             day of the week).
c
c      o  cal_IsLeap        - Determine whether a given year is a leap year.
c
c      o  cal_TimePassed    - Determine the time passed between two dates.
c
c      o  cal_AddTime       - Add a time interval either to a time interval
c                             or to a date.
c
c      o  cal_TimeInterval  - Given a time interval return the corresponding
c                             date array.
c
c      o  cal_SubDates      - Determine the time interval between two dates
c                             or between two time intervals.
c
c      o  cal_ConvDate      - Decompose a date array or a time interval
c                             array into its components.
c
c      o  cal_CopyDate      - Copy a date array or a time interval array to
c                             another array.
c
c      o  cal_CompDates     - Compare two calendar dates or time intervals.
```

```
c
c      o  cal_ToSeconds     - Given a time interval array return the number
c                             of seconds.
c
c      o  cal_WeekDay       - Return the weekday as a string given the
c                             calendar date.
c
c      o  cal_NumInts       - Return the number of time intervals between two
c                             given dates.
c
c      o  cal_StepsPerDay   - Given an iteration number or the current
c                             integration time return the number of time
c                             steps to integrate in the current calendar day.
c
c      o  cal_DaysPerMonth  - Given an iteration number or the current
c                             integration time return the number of days
c                             to integrate in this calendar month.
c
c      o  cal_MonthsPerYear - Given an iteration number or the current
c                             integration time return the number of months
c                             to integrate in the current calendar year.
c
c      o  cal_StepsForDay   - Given the integration day return the number
c                             of steps to be integrated, the first step,
c                             and the last step in the day specified. The
c                             first and the last step refer to the total
c                             number of steps (1, ... , cal_IntSteps).
c
c      o  cal_DaysForMonth  - Given the integration month return the number
c                             of days to be integrated, the first day,
c                             and the last day in the month specified. The
c                             first and the last day refer to the total
c                             number of steps (1, ... , cal_IntDays).
c
c      o  cal_MonthsForYear - Given the integration year return the number
c                             of months to be integrated, the first month,
c                             and the last month in the year specified. The
c                             first and the last step refer to the total
c                             number of steps (1, ... , cal_IntMonths).
c
c      o  cal_Intsteps      - Return the number of calendar years that are
c                             affected by the current integration.
c
c      o  cal_IntDays       - Return the number of calendar days that are
c                             affected by the current integration.
c
c      o  cal_IntMonths     - Return the number of calendar months that are
c                             affected by the current integration.
c
c      o  cal_IntYears      - Return the number of calendar years that are
c                             affected by the current integration.
c
c      o  cal_nStepDay      - Return the number of time steps that can be
c                             performed during one calendar day.
c
c      o  cal_CheckDate     - Do some simple checks on a date array or on a
c                             time interval array.
```

```
c
c      o  cal_PrintError    - Print error messages according to the flags
c                             raised by the calendar routines.
c
c      o  cal_PrintDate     - Print a date array in some format suitable for
c                             the MITgcmuv's protocol output.
c
c      o  cal_TimeStamp     - Given the time and the iteration number return
c                             the date and print all the above numbers.
c
c      o  cal_Summary       - List all the setttings of the calendar tool.
```

8.4 The line search optimisation algorithm

8.4.1 General features

The line search algorithm is based on a quasi-Newton variable storage method which was implemented by [21].

TO BE CONTINUED...

8.4.2 The online vs. offline version

- **Online version**
  Every call to *simul* refers to an execution of the forward and adjoint model. Several iterations of optimization may thus be performed within a single run of the main program (lsopt_top). The following cases may occur:
  - cold start only (no optimization)
  - cold start, followed by one or several iterations of optimization
  - warm start from previous cold start with one or several iterations
  - warm start from previous warm start with one or several iterations
- **Offline version**
  Every call to simul refers to a read procedure which reads the result of a forward and adjoint run Therefore, only one call to simul is allowed, `itmax = 0`, for cold start `itmax = 1`, for warm start Also, at the end, **x(i+1)** needs to be computed and saved to be available for the offline model and adjoint run

In order to achieve minimum difference between the online and offline code **xdiff(i+1)** is stored to file at the end of an (offline) iteration, but recomputed identically at the beginning of the next iteration.

8.4.3 Number of iterations vs. number of simulations

- `itmax:` controls the max. number of iterations
- `nfunc:` controls the max. number of simulations within one iteration

Summary
From one iteration to the next the descent direction changes. Within one iteration more than one forward and adjoint run may be performed. The updated control used as input for these simulations uses the same descent direction, but different step sizes.

Description
From one iteration to the next the descent direction dd changes using the result for the adjoint vector gg of the previous iteration. In lsline the updated control

$$\texttt{xdiff}(\texttt{i},1) = \texttt{xx}(\texttt{i}-1) + \texttt{tact}(\texttt{i}-1,1) * \texttt{dd}(\texttt{i}-1)$$

serves as input for a forward and adjoint model run yielding a new `gg(i,1)`. In general, the new solution passes the 1st and 2nd Wolfe tests so `xdiff(i,1)` represents the solution sought:

$$\texttt{xx(i)} = \texttt{xdiff(i,1)}$$

If one of the two tests fails, an inter- or extrapolation is invoked to determine a new step size `tact(i-1,2)`. If more than one function call is permitted, the new step size is used together with the "old" descent direction `dd(i-1)` (i.e. dd is not updated using the new `gg(i)`), to compute a new

$$\texttt{xdiff(i,2)} = \texttt{xx(i-1)} + \texttt{tact(i-1,2)} * \texttt{dd(i-1)}$$

that serves as input in a new forward and adjoint run, yielding `gg(i,2)`. If now, both Wolfe tests are successful, the updated solution is given by

$$\texttt{xx(i)} = \texttt{xdiff(i,2)} = \texttt{xx(i-1)} + \texttt{tact(i-1,2)} * \texttt{dd(i-1)}$$

In order to save memory both the fields dd and xdiff have a double usage.

`xdiff`
- in *lsopt_top*: used as `x(i) - x(i-1)` for Hessian update
- in *lsline*: intermediate result for control update `x = x + tact*dd`

`dd`
- in *lsopt_top, lsline*: descent vector, `dd = -gg` and `hessupd`
- in *dgscale*: intermediate result to compute new preconditioner

The parameter file lsopt.par

- **NUPDATE** max. no. of update pairs `(gg(i)-gg(i-1), xx(i)-xx(i-1))` to be stored in `OPWARMD` to estimate Hessian [pair of current iter. is stored in `(2*jmax+2, 2*jmax+3)` jmax must be ¿ 0 to access these entries] Presently `NUPDATE` must be ¿ 0 (i.e. iteration without reference to previous iterations through `OPWARMD` has not been tested)
- **EPSX** relative precision on xx bellow which xx should not be improved
- **EPSG** relative precision on gg below which optimization is considered successful
- **IPRINT** controls verbose (¿=1) or non-verbose output
- **NUMITER** max. number of iterations of optimisation; NUMTER = 0: cold start only, no optimization
- **ITER_NUM** index of new restart file to be created (not necessarily = NUMITER!)

- **NFUNC** max. no. of simulations per iteration (must be ¿ 0); is used if step size `tact` is inter-/extrapolated; in this case, if NFUNC ¿ 1, a new simulation is performed with same gradient but "improved" step size
- **FMIN** first guess cost function value (only used as long as first iteration not completed, i.e. for jmax ¡= 0)

OPWARMI, OPWARMD files Two files retain values of previous iterations which are used in latest iteration to update Hessian:

- **OPWARMI**: contains index settings and scalar variables

`n = nn`	no. of control variables
`fc = ff`	cost value of last iteration
`isize`	no. of bytes per record in OPWARMD
`m = nupdate`	max. no. of updates for Hessian
`jmin, jmax`	pointer indices for OPWARMD file (cf. below)
`gnorm0`	norm of first (cold start) gradient gg
`iabsiter`	total number of iterations with respect to cold start

- **OPWARMD**: contains vectors (control and gradient)

entry	name	description
1	`xx(i)`	control vector of latest iteration
2	`gg(i)`	gradient of latest iteration
3	`xdiff(i),diag`	preconditioning vector; (1,...,1) for cold start
2*jmax+2	`gold=g(i)-g(i-1)`	for last update (jmax)
2*jmax+3	`xdiff=tact*d=xx(i)-xx(i-1)`	for last update (jmax)

Error handling

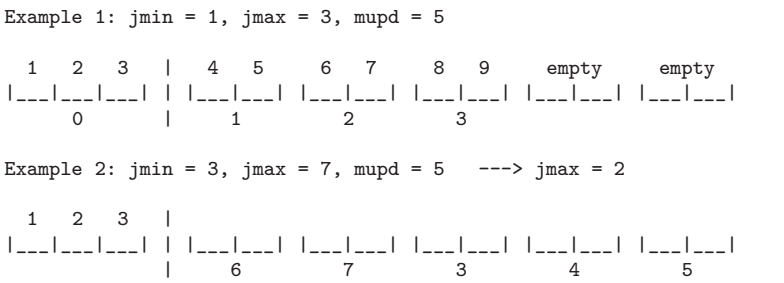


Figure 8.1: Examples of OPWARM file handling

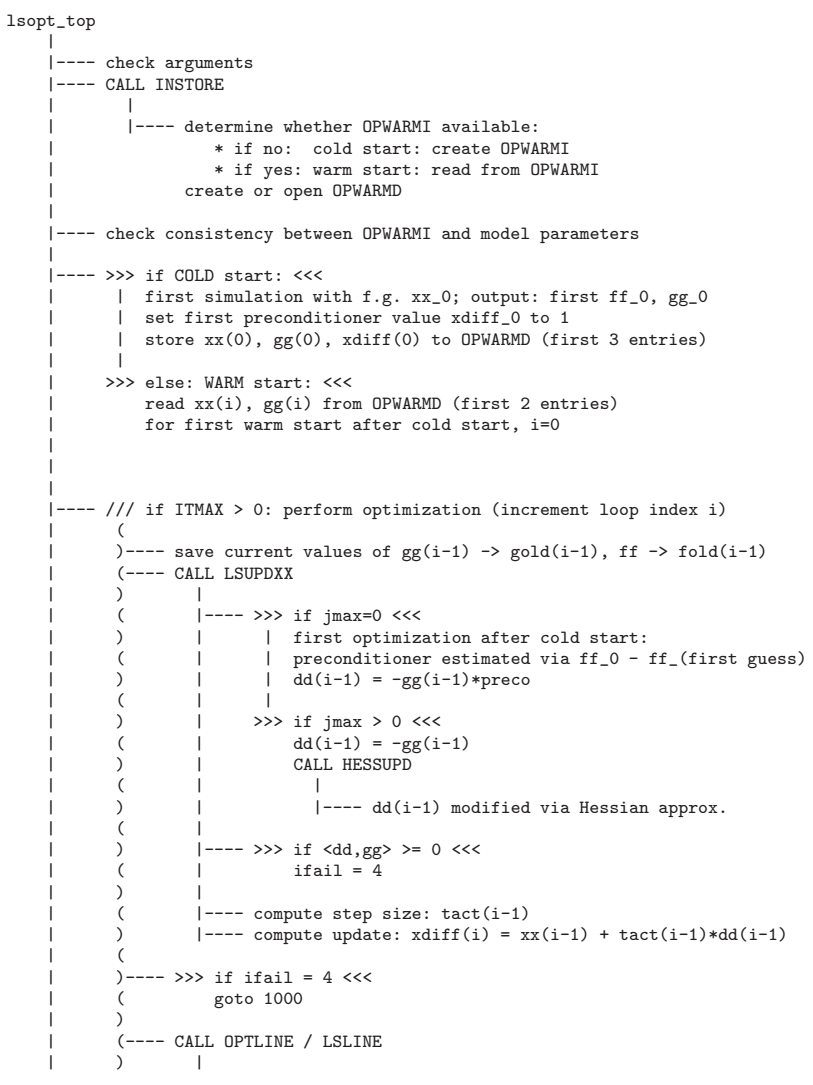


Figure 8.2: Flow chart (part 1 of 3)

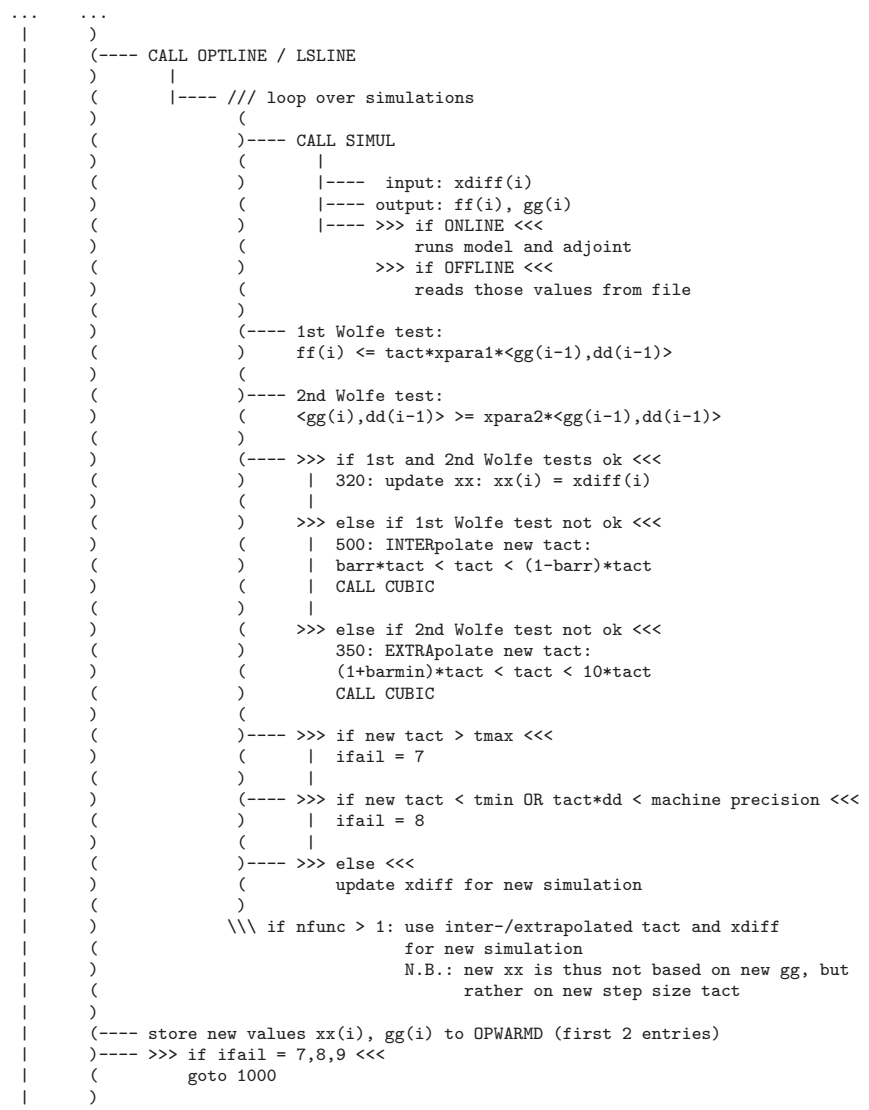


Figure 8.3: Flow chart (part 2 of 3)

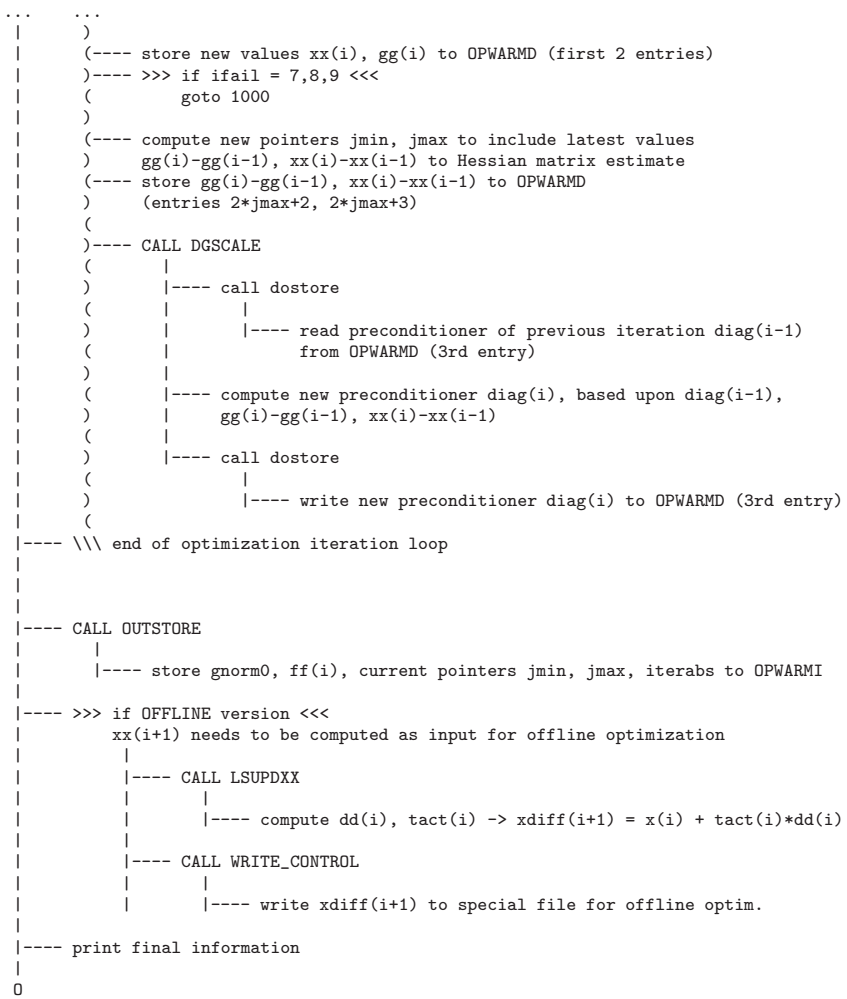


Figure 8.4: Flow chart (part 3 of 3)

Chapter 9

Model Uses

To date, the MITgcm model has been used for number of purposes. The following papers are an incomplete yet fairly broad sample of MITgcm applications:

- R.S. Gross and I. Fukumori and D. Menemenlis (2003). Atmospheric and oceanic excitation of the Earth's wobbles during 1980-2000. *Journal of Geophysical Research-Solid Earth*, vol.108.
- M.A. Spall and R.S. Pickart (2003). Wind-driven recirculations and exchange in the Labrador and Irminger Seas. *Journal of Physical Oceanography*, vol.33, pp.1829–1845.
- B. Ferron and J. Marotzke (2003). Impact of 4D-variational assimilation of WOCE hydrography on the meridional circulation of the Indian Ocean. *Deep-Sea Research Part II-Topical Studies in Oceanography*, vol.50, pp.2005–2021.
- G.A. McKinley and M.J. Follows and J. Marshall and S.M. Fan (2003). Interannual variability of air-sea O-2 fluxes and the determination of CO2 sinks using atmospheric O-2/N-2. *Geophysical Research Letters*, vol.30.
- B.Y. Huang and P.H. Stone and A.P. Sokolov and I.V. Kamenkovich (2003). The deep-ocean heat uptake in transient climate change. *Journal of Climate*, vol.16, pp.1352–1363.
- G. de Coetlogon and C. Frankignoul (2003). The persistence of winter sea surface temperature in the North Atlantic. *Journal of Climate*, vol.16, pp.1364–1377.
- K.H. Nisancioglu and M.E. Raymo and P.H. Stone (2003). Reorganization of Miocene deep water circulation in response to the shoaling of the Central American Seaway. *Paleoceanography*, vol.18.
- T. Radko and J. Marshall (2003). Equilibration of a Warm Pumped Lens on a beta plane. *Journal of Physical Oceanography*, vol.33, pp.885–899.

- T. Stipa (2002). The dynamics of the N/P ratio and stratification in a large nitrogen-limited estuary as a result of upwelling: a tendency for offshore Nodularia blooms. *Hydrobiologia*, vol.487, pp.219–227.

- D. Stammer and C. Wunsch and R. Giering and C. Eckert and P. Heimbach and J. Marotzke and A. Adcroft and C.N. Hill and J. Marshall (2003). Volume, heat, and freshwater transports of the global ocean circulation 1993-2000, estimated from a general circulation model constrained by World Ocean Circulation Experiment (WOCE) data. *Journal of Geophysical Research-Oceans*, vol.108.

- P. Heimbach and C. Hill and R. Giering (2002). Automatic generation of efficient adjoint code for a parallel Navier-Stokes solver. *Computational Science-ICCS 2002, PT II, Proceedings*, vol.2330, pp.1019–1028.

- D. Stammer and C. Wunsch and R. Giering and C. Eckert and P. Heimbach and J. Marotzke and A. Adcroft and C.N. Hill and J. Marshall (2002). Global ocean circulation during 1992-1997, estimated from ocean observations and a general circulation model. *Journal of Geophysical Research-Oceans*, vol.107.

- M. Solovev and P.H. Stone and P. Malanotte-Rizzoli (2002). Assessment of mesoscale eddy parameterizations for a single-basin coarse-resolution ocean model. *Journal of Geophysical Research-Oceans*, vol.107.

- C. Herbaut and J. Sirven and S. Fevrier (2002). Response of a simplified oceanic general circulation model to idealized NAO-like stochastic forcing. *Journal of Physical Oceanography*, vol.32, pp.3182–3192.

- C. Wunsch (2002). Oceanic age and transient tracers: Analytical and numerical solutions. *Journal of Geophysical Research-Oceans*, vol.107.

- J. Sirven and C. Frankignoul and D. de Coetlogon and V. Taillandier (2002). Spectrum of wind-driven baroclinic fluctuations of the ocean in the midlatitudes. *Journal of Physical Oceanography*, vol.32, pp.2405–2417.

- R. Zhang and M. Follows and J. Marshall (2002). Mechanisms of thermohaline mode switching with application to warm equable climates. *Journal of Climate*, vol.15, pp.2056–2072.

- G. Brostrom (2002). On advection and diffusion of plankton in coarse resolution ocean models. *Journal of Marine Systems*, vol.35, pp.99–110.

- T. Lee and I. Fukumori and D. Menemenlis and Z.F. Xing and L.L. Fu (2002). Effects of the Indonesian Throughflow on the Pacific and Indian oceans. *Journal of Physical Oceanography*, vol.32, pp.1404–1429.

- C. Herbaut and J. Marshall (2002). Mechanisms of buoyancy transport through mixed layers and statistical signatures from isobaric floats. *Journal of Physical Oceanography*, vol.32, pp.545–557.

- J. Marshall and H. Jones and R. Karsten and R. Wardle (2002). Can eddies set ocean stratification?, *Journal of Physical Oceanography*, vol.32, pp.26–38.

- C. Herbaut and J. Sirven and A. Czaja (2001). An idealized model study of the mass and heat transports between the subpolar and subtropical gyres. *Journal of Physical Oceanography*, vol.31, pp.2903–2916.

- R.H. Kase and A. Biastoch and D.B. Stammer (2001). On the Mid-Depth Circulation in the Labrador and Irminger Seas. *Geophysical Research Letters*, vol.28, pp.3433–3436.

- R. Zhang and M.J. Follows and J.P. Grotzinger and J. Marshall (2001). Could the Late Permian deep ocean have been anoxic?. *Paleoceanography*, vol.16, pp.317–329.

- A. Adcroft and J.R. Scott and J. Marotzke (2001). Impact of geothermal heating on the global ocean circulation. *Geophysical Research Letters*, vol.28, pp.1735–1738.

- J. Marshall and D. Jamous and J. Nilsson (2001). Entry, flux, and exit of potential vorticity in ocean circulation. *Journal of Physical Oceanography*, vol.31, pp.777–789.

- A. Mahadevan (2001). An analysis of bomb radiocarbon trends in the Pacific. *Marine Chemistry*, vol.73, pp.273–290.

- G. Brostrom (2000). The role of the annual cycles for the air-sea exchange of CO2. *Marine Chemistry*, vol.72, pp.151–169.

- Y.H. Zhou and H.Q. Wu and N.H. Yu and D.W. Zheng (2000). Excitation of seasonal polar motion by atmospheric and oceanic angular momentums. *Progress in Natural Science*, vol.10, pp.931–936.

- R. Wardle and J. Marshall (2000). Representation of eddies in primitive equation models by a PV flux. *Journal of Physical Oceanography*, vol.30, pp.2481–2503.

- P.S. Polito and O.T. Sato and W.T. Liu (2000). Characterization and validation of the heat storage variability from TOPEX/Poseidon at four oceanographic sites. *Journal of Geophysical Research-Oceans*, vol.105, pp.16911–16921.

- R.M. Ponte and D. Stammer (2000). Global and regional axial ocean angular momentum signals and length-of-day variations (1985-1996). *Journal of Geophysical Research-Oceans*, vol.105, pp.17161–17171.

- J. Wunsch (2000). Oceanic influence on the annual polar motion. *Journal of GEODYNAMICS*, vol.30, pp.389–399.

- D. Menemenlis and M. Chechelnitsky (2000). Error estimates for an ocean general circulation model from altimeter and acoustic tomography data. *Monthly Weather Review*, vol.128, pp.763–778.

- Y.H. Zhou and D.W. Zheng and N.H. Yu and H.Q. Wu (2000). Excitation of annual polar motion by atmosphere and ocean. *Chinese Science Bulletin*, vol.45, pp.139–142.

- J. Marotzke and R. Giering and K.Q. Zhang and D. Stammer and C. Hill and T. Lee (1999). Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *Journal of Geophysical Research-Oceans*, vol.104, pp.29529–29547.

- R.M. Ponte and D. Stammer (1999). Role of ocean currents and bottom pressure variability on seasonal polar motion. *Journal of Geophysical Research-Oceans*, vol.104, pp.23393–23409.

- J. Nastula and R.M. Ponte (1999). Further evidence for oceanic excitation of polar motion. *Geophysical Journal International*, vol.139, pp.123–130.

- K.Q. Zhang and J. Marotzke (1999). The importance of open-boundary estimation for an Indian Ocean GCM-data synthesis. *Journal of Marine Research*, vol.57, pp.305–334.

- J. Marshall and D. Jamous and J. Nilsson (1999). Reconciling thermodynamic and dynamic methods of computation of water-mass transformation rates. *Deep-Sea Research PART I-Oceanographic Research Papers*, vol.46, pp.545–572.

- J. Marshall and F. Schott (1999). Open-ocean convection: Observations, theory, and models. *Reviews of Geophysics*, vol.37, pp.1–64.

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

- A.B. Baggeroer and T.G. Birdsall and C. Clark and J.A. Colosi and B.D. Cornuelle and D. Costa and B.D. Dushaw and M. Dzieciuch and A.M.G. Forbes and C. Hill and B.M. Howe and J. Marshall and D. Menemenlis and J.A. Mercer and K. Metzger and W. Munk and R.C. Spindel and D. Stammer and P.F. Worcester and C. Wunsch (1998). Ocean climate change: Comparison of acoustic tomography, satellite altimetry, and modeling. *Science*, vol.281, pp.1327–1332.

- C. Wunsch and D. Stammer (1998). Satellite altimetry, the marine geoid, and the oceanic general circulation. *Annual Review of Earth and Planetary Sciences*, vol.26, pp.219–253.

- A. Shaw and Arvind and K.C. Cho and C. Hill and R.P. Johnson and J. Marshall (1998). A comparison of implicitly parallel multithreaded and data-parallel implementations of an ocean model. *Journal of Parallel and Distributed Computing*, vol.48, pp.1–51.

- T.W.N. Haine and J. Marshall (1998). Gravitational, symmetric, and baroclinic instability of the ocean mixed layer. *Journal of Physical Oceanography*, vol.28, pp.634–658.

- R.M. Ponte and D. Stammer and J. Marshall (1998). Oceanic signals in observed motions of the Earth's pole of rotation. *Nature*, vol.391, pp.476–479.

- D. Menemenlis and C. Wunsch (1997). Linearization of an oceanic general circulation model for data assimilation and climate studies. *Journal of Atmospheric and Oceanic Technology*, vol.14, pp.1420–1443.

- H. Jones and J. Marshall (1997). Restratification after deep convection. *Journal of Physical Oceanography*, vol.27, pp.2276–2287.

- S.R. Jayne and R. Tokmakian (1997). Forcing and sampling of ocean general circulation models: Impact of high-frequency motions. *Journal of Physical Oceanography*, vol.27, pp.1173–1179.

Bibliography

[1] A. Adcroft. *Numerical Algorithms for use in a Dynamical Model of the Ocean*. PhD thesis, Imperial College, London, 1995.

[2] A. Adcroft and J.-M. Campin. Comparison of finite volume schemes and direct-space-time methods for ocean circulation models. *Ocean Modelling*, 2002. in preparation.

[3] A. Adcroft and D. Marshall. How slippery are piecewise-constant coastlines in numerical ocean models? *Tellus*, 50(1):95–108, 1998.

[4] A.J. Adcroft and J.-M. Campin. Re-scaled height coordinates for accurate representation of free-surface flows in ocean circulation models. *Ocean Modelling*, 2004. in press.

[5] A.J. Adcroft, C.N. Hill, and J. Marshall. Representation of topography by shaved cells in a height coordinate ocean model. *MWR*, 125:2293–2315, 1997.

[6] Hill C. Adcroft, A. and J. Marshall. A new treatment of the coriolis terms in c-grid models at both high and low resolutions. *MWR*, 127:1928–1936, 1999.

[7] A. Arakawa and V. Lamb. Computational design of the basic dynamical processes of the ucla general circulation model. In *Methods in Computational Physics*, volume 17, pages 174–267. Academic Press, 1977.

[8] J.-M. Campin, A. Adcroft, C. Hill, and J. Marshall. Conservation of properties in a free-surface model. *Ocean Modelling*, 6:221–244, 2004.

[9] Daniel Jamous Chris Hill, Alistair Adcroft and John Marshall. A strategy for terascale climate modeling. In *In Proceedings of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, pages 406–425. World Scientific, 1999.

[10] A. da Silva, A. C. Young, and S. Levitus. Atlas of surface marine data 1994, volume 1: Algorithms and procedures. *NOAA Atlas NESDIS*, 6, 1994. http:////ingrid.ldeo.columbia.edu/SOURCES/.DASILVA/.SMD94/.

[11] G. Danabasoglu and McWilliams J.C. Sensitivity of the global ocean circulation to parameterizations of mesoscale tracer transports. *JC*, 8(12):2967–2987, 1995.

[12] Roland A. de Szoeke and Roger M. Samelson. The duality between the Boussinesq and Non-Boussinesq hydrostatic equations of motion. *J. Phys. Oceanogr.*, 32(8):2194–2203, 2002.

[13] Adcroft et al. Energy conversion in discrete numerical models. *Ocean Modelling*, 2002. in preparation.

[14] G. M. Flato and W. D. Hibler, III. Modeling pack ice as a cavitating fluid. *J. Phys. Oceanogr.*, 22:626–651, 1992.

[15] P. Fofonoff and R.C. Millard, Jr. Algorithms for computation of fundamental properties of seawater. Unesco Technical Papers in Marine Science 44, Unesco, 1983.

[16] P.R. Gent and J.C. McWilliams. Isopycnal mixing in ocean circulation models. *J. Phys. Oceanogr.*, 20:150–155, 1990.

[17] P.R. Gent, J. Willebrand, T.J. McDougall, and J.C. McWilliams. Parameterizing eddy-induced tracer transports in ocean circulation models. *J. Phys. Oceanogr.*, 25:463–474, 1995.

[18] R. Giering. Tangent linear and adjoint model compiler. users manual 1.4 (tamc version 5.2). Report , Massachusetts Institute of Technology, MIT/EAPS; 77 Massachusetts Ave.; Cambridge (MA) 02139; USA, 1999. http://puddle.mit.edu/∼ralf/tamc/tamc.html.

[19] R. Giering. Tangent linear and adjoint biogeochemical models. In P. Kasibhatla, M. Heimann, P. Rayner, N. Mahowald, R.G. Prinn, and D.E. Hartley, editors, *Inverse methods in global biogeochemical cycles*, volume 114 of *Geophysical Monograph*, pages 33–48. American Geophysical Union, Washington, DC, 2000.

[20] R. Giering and T. Kaminski. Recipes for adjoint code construction. *ACM Transactions on Mathematical Software*, 24:437–474, 1998.

[21] J.C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-newton algorithms. *Math. Programming*, 45:407–435, 1989.

[22] A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse Automatic Differentiation. *Optimization Methods and Software*, 1:35–54, 1992.

[23] A. Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*, volume 19 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 2000.

[24] M. Griffies, S. and W. Hallberg, R. Biharmonic friction with a smagorinsky-like viscosity for use in large-scale eddy-permitting ocean models. *MWR*, 128(8):2935–2946, 2000.

[25] R.L. Haney. Surface thermal boundary conditions for ocean circulation models. *J. Phys. Oceanogr.*, 1:241–248, 1971.

[26] I.M. Held and M.J. Suarez. A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models. *Bulletin of the American Meteorological Society*, 75(10):1825–1830, 1994.

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

[28] W. D. Hibler, III. Modeling a variable thickness sea ice cover. *Mon. Wea. Rev.*, 1:1943–1973, 1980.

[29] C. Hill, M. Follows, V. Bugnion, and J. Marshall. Spatial and temporal impacts of ocean general circulation on carbon sequestration. *Global Biogeochemical Cycles*, 2002. submitted.

[30] C. Hill and J. Marshall. Application of a parallel navier-stokes model to ocean circulation in parallel computational fluid dynamics. In N. Satofuka A. Ecer, J. Periaux and S. Taylor, editors, *Implementations and Results Using Parallel Computers*, pages 545–552. Elsevier Science B.V.: New York, 1995.

[31] W.R. Holland and L. B. Lin. On the origin of mesoscale eddies and their contribution to the general circulation of the ocean. i. a preliminary numerical experiment. *J. Phys. Oceanogr.*, 5:642–657, 1975a.

[32] E. C. Hunke and J. K. Dukowicz. An elastic-viscous-plastic model for sea ice dynamics. *J. Phys. Oceanogr.*, 27:1849–1867, 1997.

[33] David R. Jackett and Trevor J. McDougall. Minimal adjustment of hydrographic profiles to achieve static stability. *J. Atmos. Ocean. Technol.*, 12(4):381–389, 1995.

[34] Shi Jiang, Peter H. Stone, and Paola Malanotte-Rizzoli. An assessment of the Geophysical Fluid Dynamics Laboratory ocean model with coarse resolution: Annual-mean climatology. *J. Geophys. Res.*, 104(C11):25,623–25,645, 1999.

[35] W.G. Large, G. Danabasoglu, S.C. Doney, and J.C. McWilliams. Sensitivity to surface forcing and boundary layer mixing in a global ocean model: Annual-mean climatology. *JPO*, 27(11):2418–2447, 1997.

[36] S. Levitus and T.P.Boyer. World Ocean Atlas 1994 Volume 4: Temperature. Technical report, NOAA Atlas NESDIS 4, 1994.

[37] J. Marotzke, R. Giering, K.Q. Zhang, D. Stammer, C. Hill, and T. Lee. Construction of the adjoint mit ocean general circulation model and application to atlantic heat transport variability. *J. Geophys. Res.*, 104, C12:29,529–29,547, 1999.

[38] J. Marshall, A. Adcroft, C. Hill, L. Perelman, and C. Heisey. A finite-volume, incompressible navier stokes model for studies of the ocean on parallel computers. *JGR*, 102(C3):5753–5766, 1997.

[39] J. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling. *JGR*, 102(C3):5733–5752, 1997.

[40] J. Marshall, H. Jones, and C. Hill. Efficient ocean modeling using non-hydrostatic algorithms. *JMR*, 18:115–134, 1998.

[41] Trevor J. McDougall, David R. Jackett, Daniel G. Wright, and Rainer Feistel. Accurate and computationally efficient algorithms for potential temperature and density of seawater. *J. Atmos. Ocean. Technol.*, 2003.

[42] Gerdes R., Koberle C., Beckmann A., Herrmann P., and Willebrand J. Mechanisms for spreading of mediterranean water in coarse-resolution numerical models. *JPO*, 29(8):1682–1700, 1999.

[43] J.M. Restrepo, G.K. Leaf, and A. Griewank. Circumventing storage limitations in variational data assimilation studies. *SIAM J. Sci. Comput.*, 19:1586–1605, 1998.

[44] R. K. Rew, G. P. Davis, S. Emmerson, and H. Davies. NetCDF User's Guide for C, FORTRAN 77, and FORTRAN 90, an interface for data access, version 3. Report, Unidata Program Center, Boulder, Colorado, 1997. http://www.unidata.ucar.edu/packages/netcdf/.

[45] P.L. Roe. Some contributions to the modelling of discontinuous flows. In B.E. Engquist, S. Osher, and R.C.J. Somerville, editors, *Large-Scale Computations in Fluid Mechanics*, volume 22 of *Lectures in Applied Mathematics*, pages 163–193. American Mathematical Society, Providence, RI, 1985.

[46] Albert J. Semtner, Jr. A model for the thermodynamic growth of sea ice in numerical investigations of climate. *J. Phys. Oceanogr.*, 6:379–389, 1976.

[47] D. Stammer, C. Wunsch, R. Giering, C. Eckert, P. Heimbach, J. Marotzke, A. Adcroft, C. Hill, and J. Marshall. The global ocean circulation during 1992 – 1997, estimated from ocean observations and a general circulation model. *J. Geophys. Res.*, 2001. in press.

[48] D. Stammer, C. Wunsch, R. Giering, Q. Zhang, J. Marotzke, J. Marshall, and C. Hill. The global ocean circulation estimated from topex/poseidon altimetry and a general circulation model. Technical Report 49, Center for

Global Change Science, Massachusetts Institute of Technology, Cambridge (MA), USA, 1997.

[49] H. Stommel. The western intensification of wind-driven ocean currents. *Trans. Am. Geophys. Union*, 29:206, 1948.

[50] K. M. Trenberth, J. Olson, and W. G. Large. The mean annual cycle in Global Ocean wind stress. *J. Phys. Oceanogr.*, 20:1742–1760, 1990.

[51] K. M. Trenberth, J. Olson, and W. G. Large. Atmospheric simulations using a gcm with simplified physical parametrization, i model climatology and variability in multidecadal experiments. *Clim. Dynamics*, 20:175–191, 2003.

[52] R. Wajsowicz. A consistent formulation of the anisotropic stress tensor for use in models of the large-scale ocean circulation. *JCP*, 105(2):333–338, 1993.

[53] M. Winton. A reformulated three-layer sea ice model. *J. Atmos. Ocean. Technol.*, 17:525–531, 2000.

[54] J. Zhang, W. D. Hibler, III, M. Steele, and D. A. Rothrock. Arctic ice-ocean modeling with and without climate restoring. *J. Phys. Oceanogr.*, 28:191–217, 1998.

[55] Jinlun Zhang and Drew Rothrock. Modeling arctic sea ice with an efficient plastic solution. *J. Geophys. Res.*, 105:3325–3338, 2000.