A Scalable and Adaptable Solution Framework within Components of the Community Climate System Model

Katherine J. Evans[1], Damian W.I. Rouson[2], Andrew G. Salinger[3], Mark A. Taylor[3], Wilbert Weijer[4], and James B. White III[1]

[1] Oak Ridge National Laboratory, Oak Ridge, TN 37831
[2] Sandia National Laboratory, Livermore, CA 94551
[3] Sandia National Laboratory, Albuquerque, NM 87185
[4] Los Alamos National Laboratory, Los Alamos, NM 87545

Abstract. A framework for a fully implicit solution method is implemented into (1) the High Order Methods Modeling Environment (HOMME), which is a spectral element dynamical core option in the Community Atmosphere Model (CAM), and (2) the Parallel Ocean Program (POP) model of the global ocean. Both of these models are components of the Community Climate System Model (CCSM). HOMME is a development version of CAM and provides a scalable alternative when run with an explicit time integrator. However, it suffers the typical time step size limit to maintain stability. POP uses a time-split semi-implicit time integrator that allows larger time steps but less accuracy when used with scale interacting physics. A fully implicit solution framework allows larger time step sizes and additional climate analysis capability such as model steady state and spin-up efficiency gains without a loss in scalability. This framework is implemented into HOMME and POP using a new Fortran interface to the Trilinos solver library, ForTrilinos, which leverages several new capabilities in the current Fortran standard to maximize robustness and speed. The ForTrilinos solution template was also designed for interchangeability; other solution methods and capability improvements can be more easily implemented into the models as they are developed without severely interacting with the code structure. The utility of this approach is illustrated with a test case for each of the climate component models.

1 Introduction

Climate simulation will not grow to the ultrascale without new algorithms to overcome the scalability barriers blocking existing implementations. Until recently, climate simulations concentrated on the question of whether the climate is changing. The emphasis is now shifting to impact assessments, mitigation and adaptation strategies, and regional details. Such studies will require significant increases in spatial resolution and model complexity while maintaining adequate throughput. The barrier to progress is the resulting decrease in time step without increasing single-process performance [1].

G. Allen et al. (Eds.): ICCS 2009, Part II, LNCS 5545, pp. 332–341, 2009.

To be able to run higher resolution global climate models, several different approaches have been taken to minimize computation time. For dynamical cores with good scalability, the whole system is solved explicitly, and increasing processor counts are utilized as grids are refined [2]. More commonly, the time integration of the climate system is integrated semi-implicitly; the relatively slower physics is solved explicitly with a larger time step size and the faster scale physics is either subcycled or solved implicitly. Inherent limitations arise with semi-implicit integration as newly resolvable physics on finer grids is included.

Fully implicit (FI) numerical frameworks provide several potential benefits to large scale model development [3]. FI methods allow longer time steps or, conversely, finer resolution at a particular time step. The time step is chosen to resolve the physical processes of interest, which for climate models ranges from over a thousand years in the deep ocean to parameterized physics and chemistry occurring on the order of seconds. Enhanced accuracy is possible because all the terms in the model equations are solved coherently, and the time discretization determines the accuracy as the model is refined and time step increased. Also, the FI framework creates a pathway to determine model sensitivities through the exploration of parameter space.

Using a solver package that is under active development and maintenance allows access to a suite of mature solvers and has maximized portability and performance for large-scale multiphysics applications [4,5]. The recent development of ForTrilinos, an interface between the C++ solver code and Fortran within the Trilinos solver package, allows for a systematic implementation of new solver and analysis capability with the Fortran-based global climate modeling community. Climate models are large, complex, multiscale, multi-institutional efforts that maintain scores of developers working in tandem, so implementing a tested and benchmarked solver package with seamless interactions between the climate code is a clear benefit.

Until recently, accessing C++ from Fortran required considerable compiler- and platform-specific knowledge, including the chosen compilers' name mangling conventions and the hardware representation of each language's data types. Achieving portability further required "flattening" interfaces by exporting C++ member functions as external procedures and flattening data structures into one-dimensional lists, the elements of which had to be chosen from a small set of types sharing common bit representations in Fortran and C++ [6]. ForTrilinos capitalizes on recently expanded compiler support for Fortran 2003 [7], which simplifies the above process and increases its portability. The commonly supported features include object-orientation, the ability to bind C structs and function prototypes to their Fortran counterparts, and the ability to declare and manipulate C types natively in Fortran.

In this brief paper, we will demonstrate the virtually seamless presence of ForTrilinos with comparable results and scaling in atmospheric (HOMME) and ocean (POP) component climate models (sections 3 and 4) of the Community Climate System Model (CCSM) [8].

2 Interchangeable Solution Framework for a Global Climate Model

2.1 Nonlinearly Consistent Solution Algorithm

The Jacobian-Free Newton-Krylov (JFNK) fully implicit solver technique is used to integrate the model equations and illustrate the utility of the ForTrilinos framework. Future additions to the JFNK algorithm, including a suite of high performance preconditioners and analysis tools mentioned in section 2.2, as well as an extension to more realistic climate test cases, are ongoing within ForTrilinos. An extensive overview of the JFNK method is provided in Knoll and Keyes (2004) [9] so only a brief explanation is provided here.

The nonlinear partial differential equation (PDE) set can be written in residual form as a nonlinear operator, $\mathbf{F}$, on the vector of dependent variables, $\mathbf{x}$, of the problem to be solved. $\mathbf{F}(\mathbf{x})$ is evaluated at a time level and then with updates for $\mathbf{x}$ at some future time, given $\delta\mathbf{x}$,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \delta\mathbf{x}^k, \tag{1}$$

until the residual decrease has reached a specified tolerance. The nonlinear iteration index k of the update is generated by solving a linearized version of the problem, a first-order Taylor series expansion of $\mathbf{F}$ about $\mathbf{x}^k$. The resulting Jacobian vector product is approximated with a finite difference approximation, rendering the operation "Jacobian-Free". The linear solution is found with the Generalized Minimum Residual Method (GMRES). The $\delta\mathbf{x}$ that satisfies the linear tolerance criterion, which is set constant to $\eta_l = 1 \times 10^{-6}$ for this study, is then sent to equation (1). The parameters used in this application of the JFNK solver in ForTrilinos are set to maximize efficiency while maintaining accuracy for the linear test case. The ability of JFNK to provide a scalable, efficient solution within multiscale codes depends on a quality, scalable preconditioner, a point illustrated in sections 3 and 4 below. To be consistent with the existing explicit and semi-implicit formulations in the component models, the explicit and fully implicit JFNK method are discretized in time with a second order method using leap-frog with an Asselin filter and Crank-Nicolson, respectively.

2.2 The Trilinos Framework of Scalable Solvers

The Trilinos framework of scalable solution algorithms [10] has a large set of mature solvers under active development, several of which have the potential to impact climate modeling. In terms of current capabilities, most Trilinos linear algebra algorithms are based on the "Epetra" data structure for vector and sparse matrix operations on distributed-memory parallel architectures. Several iterative linear solver algorithms, such as common variants of GMRES and CG solvers, can be invoked on Epetra data structures.

Above the linear algebra layer is the embedded analysis tool layer that includes (1) LOCA, a continuation package for parameter studies and bifurcation

analysis, (2) a time integration package, and (3) an iterative eigensolver for stability analysis. The JFNK algorithm described in section 2.1 has been implemented through the use of the nonlinear solve package NOX within LOCA. Exchanging layers to access these capabilities requires only small extensions to the interface for the nonlinear solver, and all of them make use of the Trilinos linear solvers underneath as needed.

The implementation of NOX within a climate model component provides not only an opportunity for improved accuracy and efficiency, but the enabling of a framework for the use of sophisticated analysis tools, including sensitivity and stability analysis, continuation and bifurcation studies, and the use of optimization algorithms to calibrate model parameters to data. Further, long term modeling development could benefit from several ongoing algorithm development efforts in Trilinos. Development of fast linear algebra kernels for multi-core and other modern computer architectures and work on new aggregation techniques for multi-level preconditioners that improve performance for non-symmetric systems that arise in highly-convective flows is ongoing. Also under development is a capability for solving for periodic orbits, which includes automatic discretization and parallelism over the time domain, which can be used, for example, to resolve steady annual cycles of ocean models.

2.3 Interfacing a Climate Component Code to Trilinos

Trilinos has been successfully connected with two component climate models. This process confronts two main hurdles: (1) Trilinos is written in C++, while climate codes are written in Fortran and (2) the solvers need to control the application. The first issue further breaks down into two subtopics: the language differences and the related programming paradigm differences. The component models employ procedural programming whereas Trilinos uses object-oriented programming in C++. For present purposes, the intersection between the climate and solver codes (one procedure call on either side) proved insufficient to justify simultaneously confronting the language and paradigm disparities. Hence, we focus on language interoperability and defer the discussion of object-orientation in Fortran 2003.

A general-purpose driver, `doloca`, facilitates invocation of the NOX nonlinear solver within the LOCA package described in section 2.2 in a procedural fashion. Portability derives from embedding a Fortran 2003 interface block of the form:

```
interface
  subroutine doloca(vector_size,vector,comm, &
                    vector_container,residual_calculator) &
  bind(C,name='doloca')
    use iso_c_binding, only : c_int,c_double,c_ptr,c_funptr
    integer(c_int)               :: vector_size,comm
    real(c_double), dimension(*) :: vector
    type(c_ptr)                  :: vector_container
    type(c_funptr), value        :: residual_calculator
```

```
    end subroutine
  end interface
```

This interface uses type parameters from the Fortran 2003 intrinsic module `iso_c_binding`. These parameters facilitate the use of similarly named C primitive types, including C pointers (`c_ptr`) and function pointers (`c_funptr`). Furthermore, the `value` attribute enables passing the corresponding argument by value instead of the Fortran default of passing by reference. Finally, the `bind(C,name='doloca')` construct binds the Fortran interface body to a corresponding C prototype and preserves case sensitivity on the C side via the `name` argument. As of December 2008, the compilers supporting these features include gfortran, g95 and those from Intel, IBM, Cray, Portland Group, and Pathscale.

The C++ code exports `doloca` via an `extern "C"` construct to suppress any C++-specific name mangling. (The aforementioned `bind` construct automatically handles any remaining name mangling.) The calling Fortran code encapsulates its solution vector and associated arguments needed to pass through the residual calculator subroutine in a Fortran derived type object. It then passes the size of the vector and the raw vector as the first and second argument to `doloca` respectively; the MPI communicator as the third argument; a C pointer to the derived type object as the fourth argument; and a C function pointer to a residual calculation procedure as the fifth argument. Inside `doloca`, NOX perturbs the raw vector and calls the residual calculator, passing it the vector and the derived type object, which is otherwise treated as a black box on the C++ side. Referencing the residual calculator via a C function pointer decouples the climate and solver source codes by obviating the need to hardwire the residual procedure name in the C++ source. Neither side dictates procedure names to the other, thus easing any later transition to a different solver or application.

The algorithmic issues of interfacing the C++ Trilinos framework and the Fortran coding are fairly simple. The vector object of the Epetra distributed parallel data structure can be constructed from an MPI Communicator, an integer length N_p of the vector on the current processor, and a double precision vector. These are all available in the Fortran code. Secondly, iterative implicit solver algorithms need to be able to call the code which provides the residual evaluation, $F(\mathbf{x})$, as a stateless subroutine (function call), given a solution vector. Unlike the implicit version, the explicit code has the evaluation of the PDE, time integration, and the update of the solution vector to the next time step all mixed together in the same code.

Presently, the climate codes are rewritten to formulate the residual of the function evaluation directly in terms of the solution vector and the time derivative of the solution vector. In POP, which is semi-implicit, this required creating a right hand side of the governing equations (refer to section 4). Both codes had their build system adapted to include the Trilinos libraries and compilation of multiple code languages.

3 JFNK Integration in HOMME Shallow Water Test Case

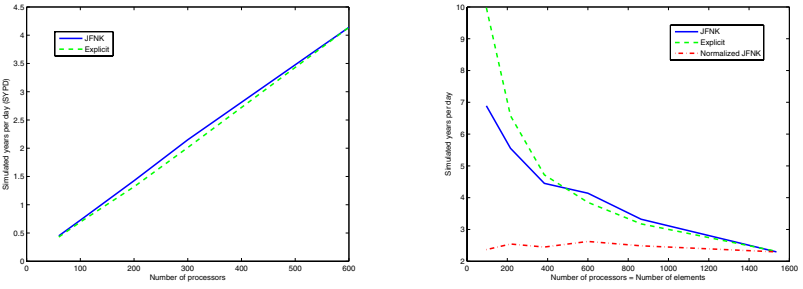
The HOMME atmospheric component model option of the CCSM uses a spectral element spatial discretization scheme, with a full description given elsewhere [11,12]. The cubed sphere uses a tiled, inscribed cube mapped to the sphere, which avoids the very disparate horizontal grid sizes near the poles that exist within a traditional latitude/longitude discretization. Ideally, the spectral element discretization on the cubed sphere retains the scalability and geometric flexibility of finite elements combined with superior accuracy and exponential convergence within each element, whereby the $O(N^3)$ cost associated with the Legendre transforms is mitigated with the use of a lower order discretization relative to a full sphere. This balance of accuracy and expense provides a scalable dynamic core option up to $O(100K)$ processors [2]. The discretization notation used here follows earlier convention, MxNxN, where M is the total number of elements on the sphere and N is the polynomial degree.

A suite of tests exist to evaluate the quality of numerical methods within a global scale atmospheric climate model [13], and the most basic test case is presented here to illustrate the utility of the ForTrilinos solver framework. The following linear example, named TC1, demonstrates the ability of HOMME to solve the shallow water equations with a fixed velocity profile. The test case specifies an initial cosine bell anomaly and the error is assessed after one rotation around the globe. In advective and residual form, the height equation is

$$0 = \frac{\partial h}{\partial t} + \nabla \cdot (h\mathbf{v}); \qquad \mathbf{v} = \{u, v\}^T, \tag{2}$$

where h is the depth of the fluid above the topography features in the model and the two-dimensional velocity field, $\mathbf{v}$, is specified in Williamson et al. (1992) in equations (75)-(76). Although only h is solved here, the full nonlinear solution framework outlined in section 2 is fully implemented. The specific parameters are matched to earlier TC1 runs with HOMME using the fully explicit [11] and semi-implicit methods [12], whereby the anomaly is advected over the corners of the cubed sphere edges using a fixed zonal rotation at an angle $\pi/4$ from the Earth's equatorial axis and the spatial grid is set to 96x16x16, which corresponds to 24576 grid points and an average resolution of about 167 km (minimum 38 km and corresponding CFL limit of 36 s).

The net time to solution for a 12 day simulation (throughout, day 1 is not included) of the explicit and JFNK method with a 30 s time step size on the Jaguar Cray XT4 on 96 processors is 4 min 53 s and 102 min 49 s with a final L_2 norm of error of 3 and 1.7×10^{-3} respectively. Because the JFNK method requires multiple function evaluations of the residual at each time step, it is more expensive than an explicit integration procedure using the same time step size. However, the time step size using JFNK is not limited by the time scale



(a) Strong Scaling for 600x16x16 (b) Scaling for Mx16x16, 1 element/proc

Fig. 1. Strong (a) and weak (b) scaling of TC1 benchmark problem for the explicit (green dashed line) and implicit (blue solid line) JFNK solution methods. SYPD is the simulated years per day. For (b), the explicit time step for 96x16x16 matches [12] and is reduced proportionally with grid refinement to maintain the same level below the stability limit. The red dashed line represents the simulation time for the normalized JFNK (see text), respectively. M is the number of elements on the sphere.

of the fastest waves in the system. When run with a 12 minute (720 s) time step size, the error using JFNK is larger than explicit (8.5×10^{-3}), however the JFNK simulation time is significantly reduced to 7 min 1 s. Note that the JFNK method is currently not preconditioned, which would reduce the simulation time and number of function evaluations further with no loss in accuracy.

The long term goal is to design a scalable method for finer resolution climate modeling studies. Figure 1a shows the strong scaling of the ForTrilinos implementation of JFNK in HOMME in units of simulated years per day (SYPD), and it is clear the method scales well up to the limits of the decomposition, one element per processor for a finer 600x16x16 grid (avg. spacing of 66.8 km). At this resolution and a 720 s time step, the JFNK method takes about the same time to complete the simulation as the explicit method. Figure 1b displays the SYPD for range of grids decomposed at 1 element per processor. This is a weak scaling without an increase in the domain, so a necessary refinement of the grid requires a smaller time step size for the explicit method and a corresponding increase in simulation time, or decreased SYPD (green dashed line). Using the JFNK method, the simulations can be completed using the same time step size with grid refinement, however they experience the same increase in simulation time (blue solid line). This is associated with the increased number of linear iterations by the Krylov-based linear solver within the outer nonlinear iteration [14]. To illustrate this effect, the SYPD for each run is normalized to the finest grid using a ratio of average number of iterations at the current and finest grid. The weak scaling of the normalized JFNK simulations (red dot-dashed line) is mostly flat with grid refinement. The action of a preconditioner is to flatten the blue JFNK line at some level above the dashed red line.

4 JFNK Integration in POP Channel Test Case

The JFNK implementation within HOMME has also been implemented in the Parallel Ocean Program (POP), a state-of-the-art Ocean General Circulation Model that is routinely run on a global domain at 0.1° spatial resolution [15], and is the ocean component of the CCSM. With implicit time stepping being implemented simultaneously in both the atmosphere and ocean components of CCSM, we are working towards a coupled system where potential efficiency gains can be obtained in its two most expensive components. Currently in POP, the (fast) barotropic mode is split off, and solved implicitly. The remaining baroclinic system is solved using an explicit leap-frog scheme. Still, increased resolution puts a severe limit on the time step that can be taken, even when no additional physical scales are included in the model.

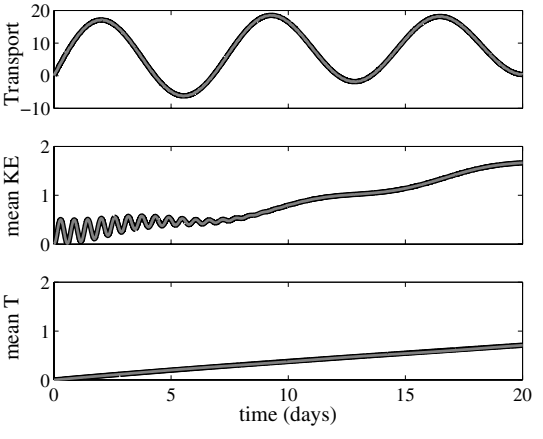


Fig. 2. Channel transport (in $10^6\,\mathrm{m}^3\,\mathrm{s}^{-1}$), mean kinetic energy ($\mathrm{kg}\,\mathrm{m}^{-1}\,\mathrm{s}^{-2}$), and mean temperature (in $10^{-3}K$), for a simulation of the POP model in a reentrant channel configuration. The JFNK scheme (gray) matches the conventional POP time stepping scheme (black).

In implementing implicit methods in POP, like HOMME, our main goals were to invade the core code as little as possible, make it user-friendly, and make it work for most, if not all, of the available physics options. Our implementation consisted of 2 major developments. In the first phase, infrastructure was added to allow for arbitrary time-stepping schemes without mode splitting, like 4th-order Runge Kutta. This required adding capability to calculate tendencies of the prognostic variables given a state vector, and to construct appropriate right-hand side, or $F(\mathbf{x})$, similar to [16]. Some extra routines and runtime switches were added, but required only minor modifications to the core POP code. The second phase of the development was specific to implicit stepping. It consisted of coupling POP to the ForTrilinos framework as outlined in section 2.3 to access the JFNK solution algorithms in Trilinos.

To test the implementation of the implicit solver in POP, we used a simple test case of the reentrant channel with an undersea bump [17]. The model is forced by a wind stress, as well as a restoring of Sea-Surface Temperature (SST) to a prescribed profile that changes linearly over the width of the channel. Figure 2 shows that the JFNK method with Crank-Nicolson implicit time stepping produces the same solution as with the leap-frog time stepping. Thus using the Trilinos solver package to integrate implicitly in POP can be achieved with little change to the underlying climate component models. The JFNK method, like HOMME, will not outperform the conventional method explicit method in terms of runtime without preconditioning.

5 Summary

The implementation of the ForTrilinos interface to the Trilinos solver package allows two Fortran-based climate components of the CCSM to take advantage of a suite of high fidelity solution methods. ForTrilinos successfully reproduces solutions to the corresponding test cases without significant alteration of the code structure or degradation of the scaling behavior. Note also that the JFNK method is called within the Trilinos LOCA package, so it is now possible to perform analyses such as parameter continuation on the model with no additional solver coding or computational expense. Also, because the JFNK method allows larger time step sizes, refined versions of the grids used for a simple benchmark study completes the simulation in the same time as the explicit method. Weak scaling of HOMME using the JFNK method performs like explicit, with increased simulation time upon grid refinement, but for different reasons. Unlike the explicit method, the JFNK method has a way around this issue, which is to reduce the number of linear iterations using a scalable preconditioner. With a good preconditioner, previous work has shown minimal growth of linear iterations with problem size [18]. This is the focus of future work using the JFNK method in component climate models within the CCSM.

Acknowledgments

The authors would like to thank Mike Heroux and Roger Pawlowski for their enabling contributions. This project was funded by the Office of Science within the U.S. Department of Energy (DOE-OS) through a combination of the Oak Ridge National Laboratory Laboratory Directed Research and Development program (Evans and White), the Climate Change Prediction Program (Weijer), and the TOPS II project within SciDAC (Rouson and Salinger), a Division of the Office of Advanced Scientific Computing Research in DOE-OS. We also acknowledge the Core Development Group of the HOMME project, which is part of the National Science Foundation funded National Center for Atmospheric Research and the POP developers. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by DOE-OS under Contract DE-AC05-00OR22725.

References

1. Simon, H., Bader, D.A. (eds.): Software Deisgn for Petascale Climate Science. In: Petascale Computing: Algorithms and Applications, ch. 7. Chapman and Hall/CRC, Boca Raton (2008)
2. Taylor, M.A., Edwards, J., St-Cyr, A.: Petascale atmospheric models for the community climate system model: new developments and evaluation of scalable dynamic cores. J. Phys. Conf. Series 125, 012023 (2008)
3. Keyes, D., Reynolds, D., Woodward, C.: Implicit solvers for large-scale nonlinear problems. J. Phys. Conf. Series 46, 433–442 (2006)
4. Lasater, M., Kelley, C., Salinger, A., Wollard, D., Zhao, P.: Parallel parameter study of the Wigner-Poisson equations for RTD's. Comp. Math. App. 51, 1677–1688 (2006)
5. Chacon, L.: Parallel implicit solvers for 3D magnetohydrodynamics. J. Phys. Conf. Series 125, 012041 (2008)
6. Gray, M.G., Roberts, R.M., Evans, T.M.: Shadow-object interface between Fortran 95 and C++. Comp. Sci. Eng. 1(2), 63–70 (1999)
7. Chivers, I.D., Sleightholme, J.: Compiler support for the Fortran 2003 standard. ACM Fortran Forum 27(2) (2008)
8. Collins, W.D., et al.: The Community Climate System Model Version 3 (CCSM3). J. Climate 19, 2122–2143 (2006)
9. Knoll, D., Keyes, D.: Jacobian-free Newton-Krylov methods: A survey of approaches and applications. J. Comput. Phys. 193, 357–397 (2004)
10. Heroux, M.A., Bartlett, R.A., Howe, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A.: An overview of the Trilinos project. ACM Trans. Math. Soft. 31(3), 397–423 (2005)
11. Taylor, M.A., Tribbia, J., Iskandarani, M.: The spectral element method for the shallow water equations on the sphere. J. Comput. Phys. 130, 92–108 (1997)
12. Thomas, S.J., Loft, R.D.: Semi-implicit spectral element model. SIAM J. Sci. Comput. 17, 339–350 (2002)
13. Williamson, D.L., Drake, J.B., Hack, J.J., Jakob, R.J., Swarztrauber, P.: A standard test set for numerical approximations to the shallow water equations in spherical geometry. J. Comput. Phys. 102, 211–224 (1992)
14. Iskandarani, M., Haidvogel, D.B., Boyd, J.P.: A staggered spectral element model with application to the oceanic shallow water equations. Internat. J. Numer. Methods Fluids, 393–414 (1995)
15. Maltrud, M.E., McClean, J.L.: An eddy-resolving global $1/10^{o}$ ocean simulation. Ocean Modeling 8, 31–54 (2005)
16. Nadiga, B.T., Taylor, M.A., Lorenz, J.: Ocean modelling for climate studies: Eliminating short time scales in long-term, high-resolution studies of ocean circulation. Math. Comp. Mod. 44, 870–886 (2006)
17. Petersen, M.R., Hecht, M.W., Wingate, B.A.: Efficient form of the lans-α turbulence model in a primitive-equation ocean model. J. Comput. Phys. 227, 5717–5735 (2008)
18. Evans, K., Knoll, D.A., Pernice, M.A.: Enhanced algorithm efficiency using a multigrid preconditioner and SIMPLE based smoother. J. Comput. Phys. 223, 121–126 (2007)