Automatic Generation of Efficient Adjoint Code for a Parallel Navier-Stokes Solver

Patrick Heimbach[1,⋆], Chris Hill[1,⋆], and Ralf Giering[2,⋆]

[1] Department of Earth, Atmospheric, and Planetary Sciences
Massachusetts Institute of Technology, Cambridge, MA 02139, USA
{heimbach, cnh}@mit.edu
http://mitgcm.org
[2] FastOpt, Martinistr. 21, D–20251 Hamburg, Germany
ralf.giering@fastopt.de
http://www.fastopt.de

Abstract. We describe key computational aspects of automatic differentiation applied to the global ocean state estimation problem. The task of minimizing a cost function measuring the ocean simulation vs. observation misfit is achieved through efficient calculation of the cost gradient w.r.t. a set of controls via the adjoint technique. The adjoint code of the parallel MIT general circulation model is generated using TAMC. To achieve a tractable problem in both CPU and memory requirements, despite the control flow reversal, the adjoint code relies heavily on the balancing of storing vs. recomputation via the checkpointing method. Further savings are achieved by exploiting self-adjointedness of part of the computation. To retain scalability of the domain decomposition, hand-written adjoint routines are provided which complement routines of the parallel support package (such as inter-processor communications, global operations, active variable I/O) to perform corresponding operations in reverse mode. The size of the problem is illustrated for the global ocean estimation problem and results are given by way of example.

1 Introduction

In one of the most complex Earth science inverse modeling initiatives ever attempted, the *Estimation of the Circulation and Climate of the Ocean* (ECCO) project is developing greatly improved estimates of the three-dimensional, time-evolving state of the global oceans. To this end, the project is applying advanced, mathematically rigorous, techniques to constrain a state-of-the-art parallel general circulation model (MITgcm) [1–3] with a diverse mix of observations. What emerges is an optimization problem that must be solved to estimate and monitor the state or "climate" of the ocean. Ocean climate is characterized by patterns of planetary scale circulation, the Gulf Stream current for example, and by large scale distributions of temperature and salinity. These quantities can be observed, but only partially, using satellites and oceanographic instruments. Combining,

⋆ On behalf of the ECCO Consortium, http://www.ecco-group.org

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2330, pp. 1019–1028, 2002.
© Springer-Verlag Berlin Heidelberg 2002

through a formal optimization procedure, the fragmentary observations with a numerical model, which is an a priori expression of the laws of physics and fluid mechanics that govern the ocean behavior, produces a more complete picture of the ocean climate. The ECCO optimization problem proceeds by expressing the difference between a model and observation from the actual ocean in terms of a scalar cost, $\mathcal{J}$, thus,

$$\mathcal{J} = \sum_{i=1}^{n} (M_i - O_i) W_i (M_i - O_i) \ , \tag{1}$$

where M_i refers to a simulated quantity projected onto the i^{th} observational data point O_i, with W_i the associated a priori error estimate. Denoting certain parameters and model state variables as adjustable "controls" C, the simulated state $M(C)$ can be optimized to minimize $\mathcal{J}$ over the n observation points. The optimized controls, C_{opt}, render a numerically simulated ocean state, $M(C_{opt})$, that is spatially and temporally complete and also consistent with observations.

The size of the ECCO optimization problem is formidable. In recent years, with increasing observational and computational capabilities, naturally occurring changes and shifts operating on time-scales of years, for example El Niño–Southern Oscillation (ENSO) [4], and decades, for example the North Atlantic Oscillation (NAO) [5], have become widely appreciated. There is every reason to expect that longer time-scale behaviors are also present in the World oceans. The optimization must, therefore, encompass processes spanning decades to centuries, on global scales; simulating them at spatial and temporal resolutions sufficient to yield state estimates with skill.

Our "smallest" current configuration is characterized by a cost function that spans nine years of planetary-scale ocean simulation and observation. The cost function operates on 10^8 elements and is optimized by corrections to a control vector, C, of size 1.5×10^8. The full Jacobian for this system contains more than 10^{16} elements ($10^8 \cdot 1.5 \times 10^8$) which, even allowing for some sparsity, is fundamentally impractical. Therefore, the reverse mode of automatic differentiation (AD), which allows the computation of the product of the Jacobian and a vector without explicitly representing the Jacobian, plays a central role.

Minimizing $\mathcal{J}$, under the side condition of fulfilling the model equations, leads to a constrained optimization problem for which the gradient

$$\boldsymbol{\nabla}_C \, \mathcal{J}(C, M(C)) \tag{2}$$

is used to reduce $\mathcal{J}$ iteratively. The constrained problem may be transformed into an unconstrained one by incorporating the model equations into the cost function (1) via the method of Lagrange multipliers. Alternatively and equivalently, the gradient may be obtained through application of the chain rule to (1).

AD [6] exploits this fact in a rigorous manner to produce, from a given model code, its corresponding tangent linear (forward mode) or adjoint (reverse mode) model; see [7]. The adjoint model enables the gradient (2) to be computed in a single integration. The reverse mode approach is extremely efficient for scalar-valued cost functions for which it is matrix free. In practical terms, we are able to

develop a system that can numerically evaluate (2), for any scalar $\mathcal{J}$, in roughly four times the compute cost of evaluating $\mathcal{J}$. At this cost, reverse mode AD provides a powerful tool that is being increasingly used for oceanographic and other geophysical fluids applications.

The results that are emerging from the application of reverse mode AD to the ocean circulation problem are of immense scientific value. However, here our focus is on the techniques that we employ to render a computationally viable system, and on providing examples of the calculations that are made possible with a competitive, automatic system for adjoint model development and integration.

The MITgcm algorithm and its software implementation in a parallel computing environment are described in Sect. 2. Section 3 discusses the implications for an AD tool and the attributes of an efficient, scalable reverse mode on a variety of parallel architectures for rendering the calculation computationally tractable. Illustrative applications are presented in Sect. 4 with an emphasis on computational aspects, rather than implications for oceanography or climate. An outlook is given in Sect. 5. For discussion of the scientific aspects of this work refer to reports and data along with animations available at [8].

2 The MIT General Circulation Model

The M.I.T General Circulation Model (MITgcm) is rooted in a general purpose grid-point algorithm that solves the Boussinesq form of the Navier-Stokes equations for an incompressible fluid in a curvilinear framework. The algorithm is described in [2, 3]; see [9] for online documentation. The work presented here uses the model's hydrostatic mode, to integrate forward equations for the potential temperature θ, salinity S, velocity vector $\underline{v}$, and pressure p of the ocean using a two phase approach at each time-step.

A skeletal outline of the iterative time-stepping procedure that is used to step forward the simulated fluid state is illustrated in Fig. 1. The two phases **PS** and **DS** are both implemented using a finite volume approach. Discrete forms of the continuous equations are deduced by integrating over the volumes and making use of Gauss' theorem. The terms in **PS** are computed explicitly from information within a local region. **DS** terms are diagnostic and involve an iterative preconditioned conjugate gradient solver.

Finite-volumes provide a natural model for parallelism. Figure 2(a) shows schematically a decomposition into sub-domains that can be computed concurrently. The implementation of the MITgcm code is such that the **PS** phase for a single timestep can be computed entirely by on processor operations. At the end of **PS** communication operations are performed. This communication and **DS** must complete before the next time-step **PS** can start. The implicit step, **DS**, tightly mixes computation and communication. Performance critical communications in MITgcm employ a communication layer in a custom software library called WRAPPER. The performance critical primitives in the WRAPPER layer are illustrated in Fig. 2(b). The operations are all linear combination and permutations of distributed data.

INITIALIZE. Define geometry, initial flow and tracer distributions
FOR each time step n DO
PS
Active I/O.
Step forward state. $\underline{v}^n = \underline{v}^{n-1} + \Delta t(\underline{G}_v^{n-\frac{1}{2}} - \nabla p^{n-\frac{1}{2}})$
Get time derivatives. $\underline{G}_v^{n+\frac{1}{2}} = \mathbf{gv}(\underline{v}, b)$
Get hydrostatic p. $p_{hy}^{n+\frac{1}{2}} = \mathbf{hy}(b)$
DS
Solve for pressure. $\nabla_h . H \nabla_h p_s^{n+\frac{1}{2}} = \nabla_h . \overline{\underline{G}_{v_h}^{n+\frac{1}{2}}}^H - \nabla_h . \overline{\nabla_h p_{hy}^{n+\frac{1}{2}}}^H$
END FOR

Fig. 1. The MITgcm algorithm iterates over a loop, with two blocks, **PS** and **DS**. A simulation may entail millions of iterations. In **PS**, time tendencies (G terms) are calculated from the state at previous time levels ($^{n,n-1,\ \cdots}$). **DS** involves finding a two-dimensional pressure field p_s to ensure the flow $\underline{v}$ at the next time level satisfies continuity. G term calculations for θ (temperature) and S (salinity) have been left out. These have a similar form to the **gv**() function and yield the buoyancy, b

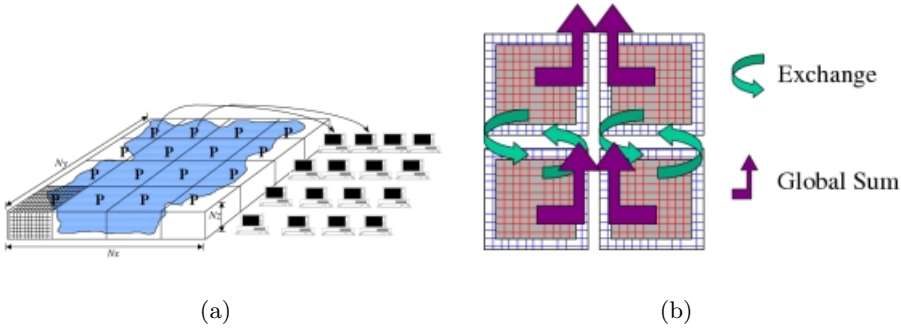

Fig. 2. Panel (a) shows a hypothetical domain of total size $N_x N_y N_z$. The domain is decomposed in two-dimensions along the N_x and N_y directions. Whenever a processor wishes to transfer data between tiles or communicate with other processors it calls a special function in a WRAPPER support layer. Two performance critical parallel primitives are provided by the WRAPPER (b). By maintaining transpose forms of these primitives we can efficiently accommodate parallel adjoint computations

3 The Adjoint of MITgcm

MITgcm has been adapted for use with the Tangent linear and Adjoint Model Compiler (TAMC) [7, 10], developed by Ralf Giering. TAMC is a source-to-source transformation tool. It exploits the chain rule for computing the deriva-

tive 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 (forward mode) or adjoint code (reverse mode), then, may be thought of as the composition in forward or reverse order, respectively, of the Jacobian matrices of the full forward code's compositional elements. The processed MITgcm code and the adjoint code have about 100k executable lines.

While the reverse mode proves extremely efficient in computing gradients with respect to a scalar cost function, a major challenge is the fact that the control flow of the original code has to be reversed. In the following we discuss some computational implications of the flow reversal, as well as issues regarding the generation of efficient, scalable adjoint code on a variety of parallel architectures.

3.1 Storing vs. Recomputation in Reverse Mode

This is a central issue upon which hinges the overall feasibility of the adjoint approach in the present context of large-scale simulation optimization and sensitivity studies. The combination of four related elements:

- the reverse nature of the adjoint calculation,
- the local character of the gradient evaluations, on which the adjoint operations are performed, for this class of time evolving problem,
- the nonlinear character of the model equations such as the equation of state and the momentum advection terms, and
- the conditional code execution (`IF ... ELSE IF ... END IF`).

Thus, it is necessary to make available the intermediate model state in reverse sequence. In principle this could be achieved by either storing the intermediate states of the computation or by successive recomputing of the forward trajectory, throughout the reverse sequence computation. Either approach, in its pure form, is prohibitive; storing of the full trajectory is limited by available fast-access, storage media, recomputation is limited by CPU resource requirements which scale as the square of the number of intermediate steps.

TAMC provides two crucial features to balance recomputation and storage. First, TAMC generates recomputations of intermediate values by the Efficient Recomputation Algorithm (ERA) [11]. Secondly, TAMC generates code to store and read intermediate values, if appropriate directives have been inserted into the code. This enables the user to choose between storing and recomputation on every code level in a very flexible way. At the time-stepping level the directives allow for checkpointing that hierarchically splits the time-stepping loop; cf. also [12, 13]. For the MITgcm, a three-level checkpointing scheme, illustrated in Fig. 3, has been adopted as follows:

lev3 The model trajectory is first subdivided into n^{lev3} subsections. The model is then integrated along the full trajectory, and the state stored at every k_i^{lev3}-th timestep.

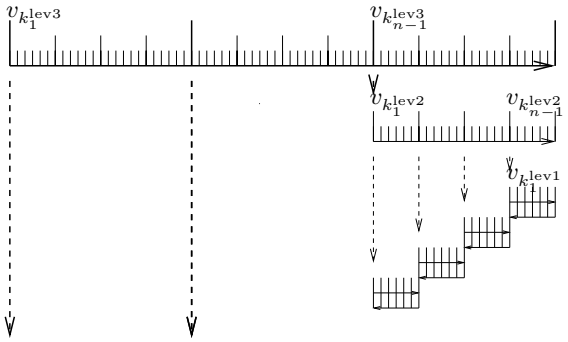

Fig. 3. Schematic view of intermediate dump and restart for 3-level checkpointing

lev2 In a second step, each "lev3"–subsection is divided into n^{lev2} subsections. The model picks up the last "lev3"–saved state $v_{k_{n-1}^{\mathrm{lev3}}}$ and is integrated forward along the last "lev2"–subsection, now storing the state at every k_i^{lev2}-th timestep.

lev1 Finally, the model picks up at the last intermediate saved state $v_{k_{n-1}^{\mathrm{lev2}}}$ and is integrated forward in time along the last "lev1"–subsection. Within this subsection only, the model state is stored at every timestep. Thus, the final state $v_n = v_{k_n^{\mathrm{lev1}}}$ is reached and the model state of all preceding timesteps along the last "lev1"–subsection are available. The adjoint can then be computed back to subsection k_{n-1}^{lev2}.

This procedure is repeated consecutively for each previous subsection carrying the adjoint computation back to initial time k_1^{lev3}.

The 3-level checkpointing requires a total of 3 forward and one adjoint integration, with the latter taking about 2.5 times a forward integration. Thus, a forward/adjoint sweep requires a total of roughly 5.5 times a forward integration. For a given decomposition of the total number of time steps $n_{\mathrm{timeSteps}} = 77,760$ (corresponding to a 9 year integration at an hourly timestep) into a hierarchy of 3 levels of sub-intervals $n_1 = 24$, $n_2 = 30$, $n_3 = 108$ with $n_{\mathrm{timeSteps}} = n_1 \cdot n_2 \cdot n_3$, the storing amount is drastically reduced from $n_{\mathrm{timeSteps}}$ to $n_1 + n_2 + n_3 = 162$ steps. Pure recomputation would incur a computation cost of $77,760^2$.

The two outer loops are stored to disks, while the innermost loop is stored to memory to avoid I/O in this phase of computation. For the outer loops, the storing consists of the model state required to perform the **PS** and **DS** phases of computation. Insertions of store directives at the innermost loop are more intricate and require detailed knowledge of the code. Typical places are near nonlinear expressions. Furthermore, directives may have to be accompanied (or may be avoided), occasionally, by additional measures to break artificial data dependency flows. For instance, a `DO`-loop which contains multiple consecutive and state-dependent assignments of a variable may be broken into several loops, to enable intermediate results to be stored before further state-dependent calculations are performed.

The character of **DS** has important implications for tangent-linear and adjoint computations. The equation solved in **DS** is self-adjoint. Exploiting this fact in reverse mode derivative calculations provides substantial computing cost savings. By providing a directive, TAMC then uses the original code where needed in the adjoint calculation.

3.2 Parallel Implementation

A number of issues required substantial intervention into the original code to enable correct adjoint code generation which is scalable and both memory and CPU efficient.

• *Exchanges between neighboring tiles* Domain decomposition is at the heart of the MITgcm's parallel implementation. Each compositional unit (tile) representing a virtual processor consists of an interior domain, truly owned by the tile and an overlap region, owned by a neighboring tile, but needed for the computational stencil within a given computational phase. The computational phase within which no communication is required can thus be considerable. Periodically, between computational phases, processors will make calls to WRAPPER functions which can use MPI, OpenMP, or combinations thereof to communicate data between tiles (so-called exchanges), in order to keep the overlap regions up-to-date. The separation into extensive, uninterrupted computational phases and minimum communication phases controlled by the WRAPPER is an important design feature for efficient parallel adjoint code generation. Corresponding adjoint WRAPPER functions were written by hand. TAMC recognizes when and where to include these routines by means of directives.

• *Global arithmetic primitives* Operations within the communication phase, for which a processor requires data outside of the overlap region of neighboring processors, must use communication libraries, such as MPI or OpenMP. So far, all global operations could be decomposed into arithmetic elements involving the global sum as the only global operation (major applications in the context of the MITgcm are the pressure inversion phase and global averages). WRAPPER routines exist, which adapt the specific form of the global sum primitive to a given platform. Corresponding adjoint routines were written by hand and directives to use these are given to TAMC.

• *Active file handling on parallel architectures* Figure 1 also shows an isolated I/O phase that deals with external data inputs that affect the calculation of (1) and (2). This isolation of "active" I/O simplifies AD code transformations. Read and write operations in forward mode are accompanied by corresponding write and read operations, respectively, in adjoint mode required for active variables. MITgcm possesses a sophisticated I/O handling package to enable a suite of global or local (tile- or processor based) I/O operations consistent with its parallel implementation. Adjoint support routines were written to retain compatibility in adjoint mode with both distributed memory and shared memory parallel operation, as implemented in the I/O package of the WRAPPER.

4 Applications

• *Global Ocean State Estimation* The estimation problem currently under way iteratively reduces the model vs. data misfit (1), by successive modification to the controls, C. To infer updates in the controls, the cost gradient (2) is subject to a quasi-Newton variable storage line search algorithm [14]. The updated controls serve as improved initial and boundary conditions in consecutive forward/adjoint calculations. Thus, $\boldsymbol{\nabla}_C \mathcal{J}$, the outcome of the adjoint calculation, is a central ingredient for the optimization problem. By way of example, Fig. 4, taken from [15], depicts the surface heat flux correction estimated from the optimization. The mean changes of the flux relative to the NCEP [16] input fields are large over the area of the Gulf Stream and in the Eastern tropical Pacific. The heat flux corrections inferred here were shown to agree with independent studies of the NCEP heat flux analyses.

• *Sensitivity Analysis* Complementary to the estimation problems, sensitivity studies have been undertaken with the MITgcm and its adjoint which aim at interpreting the adjoint or dual solution of the model state [17]. As an example, Fig. 5 depicts the sensitivity of the North Atlantic heat transport at 24^oN to changes in surface temperature.

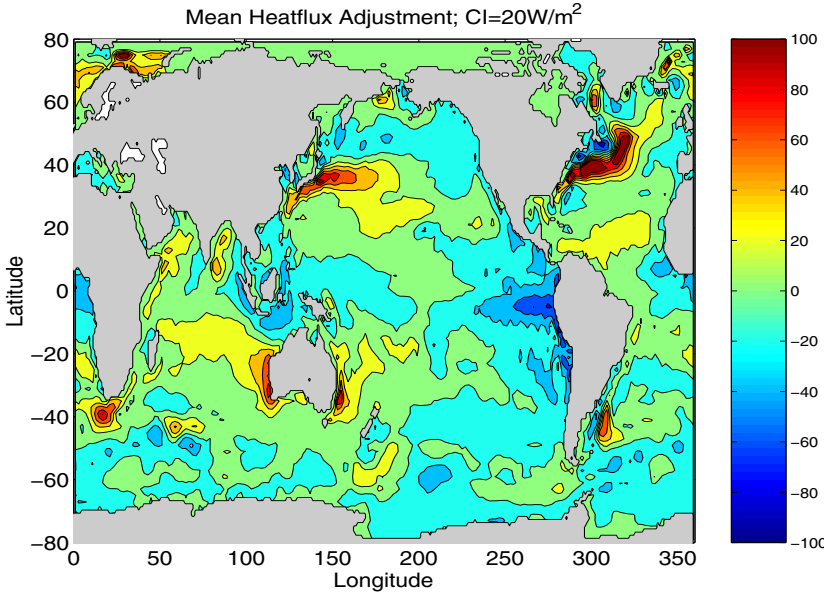

Fig. 4. Mean changes in heat flux relative to the NCEP first guess fields

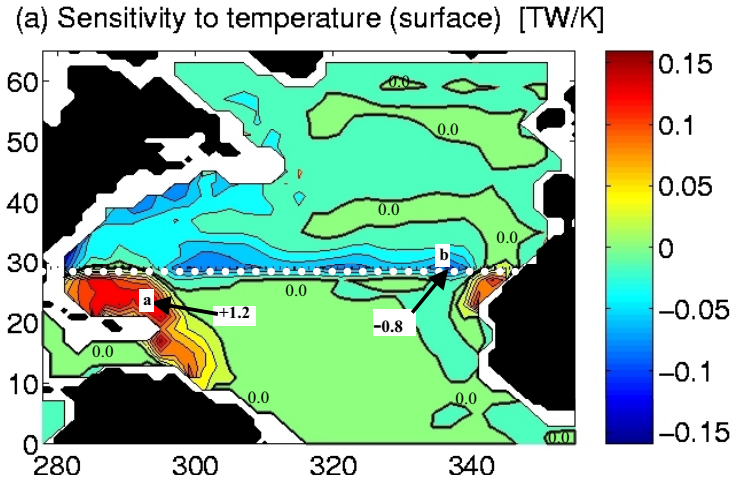

Fig. 5. The sensitivity, $\frac{\partial \mathcal{J}}{\partial \theta}$, of the North Atlantic heat transport at 24°N, $\mathcal{J}$, to changes in temperature, θ, at the ocean surface. At point "a" a persistent, 1 year, unit change in θ will produce a 1.2×10^{12} W increase in annual mean poleward heat transport. The same change at point "b" produces a 0.8×10^{12} W decrease (from [17])

5 Summary and Outlook

Reverse mode AD applications are emerging as powerful tools in oceanographic research. Essential technical features are efficient recomputation algorithms and checkpointing. Scalability of the adjoint code, maintained by hand-written adjoint functions, complements parallel support functions of the forward code. This renders computationally tractable code. AD has been successfully used in solving an unprecedented optimization problem. In addition, a host of physical quantities can be efficiently and rigorously investigated in terms of their sensitivities by means of the dual solution provided by the cost gradient, thus providing novel insight into physical mechanisms. Further reverse mode applications are being pursued using MITgcm's adjoint. They include optimal perturbation/singular vector analyses in the context of investigating atmosphere-ocean coupling. A natural extension of the state estimation problem is the inclusion of estimates of the errors of the optimal controls. The computation of the full error covariance remains prohibitive, but dominant structures may well be extracted from the Hessian matrix. As ambitions grow the ECCO group has recently switched to the TAF tool (Transformation of Algorithms in Fortan), the successor of TAMC, which has enhanced features. ECCO is contributing to the Adjoint Compiler Technology & Standards (ACTS) project, an initiative to increase accessibility to and development of AD algorithms by a larger community through the definition of a common intermediate algorithmic platform, within which AD algorithms can be easily shared among different developers and tools.

Acknowledgments

This is paper is a contribution to the ECCO project, supported by NOPP, and with funding from NASA, NSF and ONR.

References

1. Hill, C., Marshall, J.: Application of a parallel Navier-Stokes model to ocean circulation in parallel computational fluid dynamics. In: Proceedings of Parallel Computational Fluid Dynamics, New York, Elsevier Science (1995) 545–552
2. Marshall, J., Hill, C., Perelman, L., Adcroft, A.: Hydrostatic, quasi-hydrostatic and nonhydrostatic ocean modeling. J. Geophys. Res. **102, C3** (1997) 5,733–5,752
3. Marshall, J., Adcroft, A., Hill, C., Perelman, L., Heisey, C.: Hydrostatic, quasi-hydrostatic and nonhydrostatic ocean modeling. J. Geophys. Res. **102, C3** (1997) 5,753–5,766
4. Neelin, J., Battisti, D., Hirst, A., Jin, F., Wakata, Y., Yamagata, T., Zebiak, S.: ENSO theory. J. Geophys. Res. **103** (1998) 14,261–14,290
5. Marshall, J., Kushnir, Y., Battisti, D., Chang, P., Czaja, A., Hurrell, J., McCartney, M., Saravanan, R., Visbeck, M.: Atlantic climate variability. Int. J. Climatology (2002) (to appear).
6. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia (2000)
7. Giering, R., Kaminski, T.: Recipes for adjoint code construction. ACM Transactions on Mathematical Software **24** (1998) 437–474
8. ECCO: `http://ecco-group.org/`. (URL)
9. MITgcm: `http://mitgcm.org/sealion/`. (URL)
10. Giering, R.: Tangent linear and Adjoint Model Compiler. Users manual 1.4 (TAMC Version 5.2). Technical report, MIT, MIT/EAPS, Cambridge (MA), USA (1999) http://puddle.mit.edu/∼ralf/tamc/tamc.html.
11. Giering, R., Kaminski, T.: Generating recomputations in reverse mode AD. In Corliss, G., Faure, C., Griewank, A., Hascoët, L., Naumann, U., eds.: Automatic Differentiation of Algorithms: From Simulation to Optimization. Springer (2002) (to appear).
12. Griewank, A.: Achieving logarithmic growth of temporal and spatial complexity in reverse Automatic Differentiation. Optimization Methods and Software **1** (1992) 35–54
13. Restrepo, J., Leaf, G., Griewank, A.: Circumvening storage limitations in variational data assimilation studies. SIAM J. Sci. Comput. **19** (1998) 1586–1605
14. Gilbert, J., Lemaréchal, C.: Some numerical experiments with variable-storage quasi-Newton algorithms. Math. Programming **45** (1989) 407–435
15. Stammer, D., Wunsch, C., Giering, R., Eckert, C., Heimbach, P., Marotzke, J., Adcroft, A., Hill, C., Marshall, J.: The global ocean circulation and transports during 1992–1997, estimated from ocean observations and a general circulation model. (2002) (to appear).
16. NCEP: `http://www.ncep.noaa.gov/`. (URL)
17. Marotzke, J., Giering, R., Zhang, K., Stammer, D., Hill, C., Lee, T.: Construction of the adjoint MIT ocean general circulation model and application to atlantic heat transport variability. J. Geophys. Res. **104, C12** (1999) 29,529–29,547