An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation

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, hill}@mit.edu
http://mitgcm.org

[2] FastOpt, Martinistr. 21, 20251 Hamburg, Germany,
ralf.giering@fastopt.de
http://www.fastopt.de

June 12, 2002

Contents

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

Abstract

We describe computational aspects of automatic differentiation applied to global ocean circulation modeling and state estimation. 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 or its successor TAF. To achieve a tractable problem in both CPU and memory requirements, in the light of 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 domain decomposition based parallelism, hand-written adjoint routines are provided. These complement routines of the parallel support package to perform corresponding operations in reverse mode. The unique feature of the TAF tool which enables to dump the adjoint state and restart the adjoint integration is exploited to overcome batch execution limitations on HPC machines for large-scale ocean and climate simulations. The size of a typical adjoint application is illustrated for the global ocean state estimation problem. Results are given by way of example.

1 Introduction

In one of the most complex Earth science inverse modeling initiatives, 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 [1, 2]. To this end, the project is applying advanced, mathematically rigorous, inverse techniques [3] to constrain a state-of-the-art parallel general circulation model, MITgcm [4, 5, 6], with a diverse mix of observations. What emerges from the ECCO project goals 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 ocean 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, 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 numerical model trajectory 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 model parameters and state variables (initial/boundary values) 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 consistent with observations within their estimated errors.

1.1 Problem size

The size of the ECCO optimization problem is formidable. In recent years, with increasing observational and computational capabilities, prominent patterns of naturally occuring intrinsic oceanic and coupled atmosphere-ocean-cryosphere variability have become widely appreciated, operating on many time-scales, from days (barotropic motion, e.g. [7]), months (mesoscale eddies, e.g. [8]), years, (e.g. the El Niño–Southern Oscillation, ENSO, [9]), decades, (e.g. the North Atlantic Oscillation, NAO, [10]), to centuries and millennia (thermohaline circulation, THC, e.g. [11]). Ideally, 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 characterised by a cost function that spans nine-years of planetary scale ocean simulation and observation, operating on 10^8 elements, and optimized by corrections to a control vector, C, of size 1.5×10^8. Major observational ingredients include global,

continuous sea surface height (SSH) data obtained from radar altimeters on board the TOPEX/Poseidon [12] and the European Remote Sensing Satellites ERS-1/2 [13] which achieve an accuracy at the 2- to 3-cm level on many spatial scales [14]. In addition, a variety of hydrographic data from the World Ocean Circulation Experiment (WOCE) [15] are used to constrain the model. Currently, the newly available data from the Jason-1 [16] altimetric mission, and depth profiles of temperature and salinity from a series of presently 300 (and potentially up to 3000 by the year 2005) autonomous seagoing floats [17] are being added.

The full Jacobian, $\frac{\partial \mathcal{J}}{\partial C}$, 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.

1.2 Role of the adjoint and AD

Minimising $\mathcal{J}$, under the side condition of fulfilling the model equations, leads to a constrained optimization problem for which the gradient $\vec{\nabla}_C \mathcal{J}$ is used to reduce $\mathcal{J}$ iteratively,

$$\min_C \mathcal{J}(C) \quad \Rightarrow \quad \vec{\nabla}_C \, \mathcal{J}(C, M(C)) \, = \, 0 \tag{2}$$

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 rigorous application of the chain rule to equation (1).

AD [18] 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 e.g. [19]). 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. The full Jacobian is never explicitly calculated and so the computational cost becomes tractable. 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.

1.3 Paper organization

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, applications and the models software implementation in a parallel computing environment are described in section 2.

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. $p_{hy}^{n+\frac{1}{2}} = \mathbf{hy}(b)$
Update halo (exchange).
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

Figure 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 ensuring the flow $\underline{v}$ at the next time level satisfies the continuity. G term calculations for θ (temperature) and S (salinity) have been left out. These have a similar form to the $\mathbf{gv}()$ function and yield the buoyancy, b.

Section 3 discusses the implications for an AD tool and the requirements of an efficient, scalable reverse mode on a variety of parallel architectures for rendering the calculation computationally tractable. Applications are presented in section 4 with an emphasis on computational aspects, rather than implications for oceanography or climate. An outlook is given in section 5. Further discussion of the scientific aspects of this work, is available, along with extensive data sets at the ECCO website [20]).

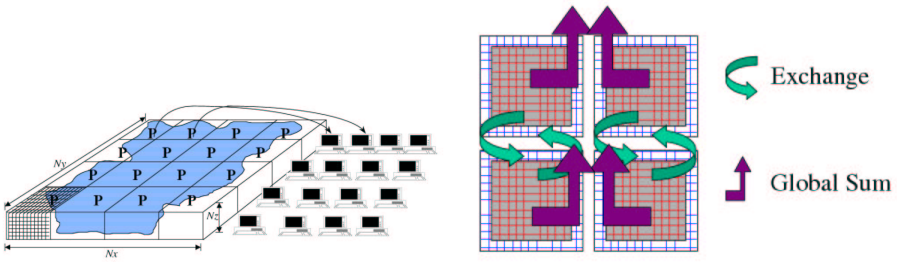
2 The MIT General Circulation Model

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, hydrostatic or fully non-hydrostatic, in a curvilinear framework. The algorithm is described in [5, 6] (for online documentation and access to the model code, see [21]).

2.1 Prognostic and diagnostic computational phases

The work presented here uses the model's hydrostatic mode, to integrate forward equations for, θ (potential temperature), S (salinity), $\underline{v}$ (velocity vector) and p (pressure) 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 Figure 1. The two phases **PS** and **DS** within each timestep are both implemented using



(a) (b)

Figure 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. Three performance critical parallel primitives are provided by the WRAPPER (b) By maintaing transpose forms of these primitives we can efficiently accomodate parallel adjoint computations.

a finite volume approach. Discrete forms of the continuous equations are deduced by integrating over the volumes and making use of Gauss's theorem. The terms in **PS** are computed explicitly from information within a local region. **DS** terms involve iteration for which an iterative preconditioned conjugate gradient scheme is used.

2.2 Parallelism

Finite-volumes provide a natural model for parallelism. Figure 2a shows schematically a decomposition into sub-domains that can be computed on concurrently. The implementation of the MITgcm code is such that the **PS** phase for a single timestep can be computed entirely by local, on processor operations. At the end of **PS** communication operations are performed which update the halo regions. This communication and **DS** must complete before the next time-step **PS** can start. The over-computations in the halo ensure theat the **PS** phase can be extended to a maximum fraction of a full timestep. The implicit step, **DS**, tightly mixes computation and communication. Performance critical communications in MITgcm employ a commuication layer in a custom software library called WRAPPER [22]. The performance critical primitives in the WRAPPER communication layer are illustrated in Figure 2b. Significantly, for AD, the WRAPPER operations are all linear combination and permutations of distributed data.

3 The Adjoint of MITgcm

The MITgcm has been adapted for use with the Tangent linear and Adjoint Model Compiler (TAMC), and recently its successor TAF (Transformation of Algorithms in Fortan) developed by Ralf Giering [19], [23], [24]. TAMC is a source-to-source transformation tool. It exploits the chain rule for computing the 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 (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 code's compositional elements. The processed MITgcm code has about 40k lines and the adjoint code about 37k lines without comments. TAF and TAMC produce code of the same performance, but code generation time by TAF is improved by a factor of approximately 5.

While the reverse mode is theroretically 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 (tangent at a point), 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, the momentum advection, the parametrization schemes)
- conditional code execution involving active variables (`IF ... ELSE IF ... END IF` expressions)

require the intermediate model state to be available 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, would be 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.

3.1.1 Example: zonal advective flux of meridional momentum

As an example of a nonlinear expression, consider the along x-axis advective flux of y-component velocity, one element of one term in the momentum equation (at a given vertical level):

$$F^x = \overline{U}_y^j \overline{v}_x^i \tag{3}$$

where $\overline{U}_y^j$ is the advecting volume transport in $\mathrm{m}^3\mathrm{s}^{-1}$ averaged along the y axis, and $\overline{v}_x^i$ is the y-component velocity in ms^{-1}, advected along the x-axis. On the staggered Arakawa C-grid used here [25], the corresponding code is of the form

```
AdvectFluxUV(i,j) = 0.25 * [ uTrans(i,j) + uTrans(i,j-1) ]
                         * [  vVel(i,j)  +  vVel(i-1,j)  ]
```

The derivative code requires both `uTrans(i,j)` and `vVel(i,j)`:

```
aduTrans(i,j) = aduTrans(i,j) + 0.25 * adAdvectFluxUV(i,j)
                                     * [ vVel(i,j) + vVel(i-1,j) ]

advVel(i,j)  =   advVel(i,j)  + 0.25 * adAdvectFluxUV(i,j)
                                     * [ uTrans(i,j) + uTrans(i,j-1) ]

adAdvectFluxUV(i,j) = 0.
```

(the adjoint code for `aduTrans(i,j-1)` and `advVel(i-1,j)` reads similarly and is omitted here). In the present case, the velocity fields `uVel` and `vVel` are stored prior to computing the momentum equation, whereas the momentum transports `uTrans` and `vTrans` may be readily re-computed from the available velocity fields.

3.1.2 Handling of storing vs. recomputation by TAMC

TAMC provides two crucial features to balance the amount of recomputation vs. storage requirements. First, TAMC generates recomputations of intermediate values by the Efficient Recomputation Algorithm (ERA,[26]). Secondly, TAMC generates code to store and read intermediate values, if appropiate directives have been inserted into the code. This enables the user to choose between storing and recomputation at every code level in a very flexible way. In the above example, storing of the velocity fields is activated by the user through insertion of appropriate TAMC directives. In contrast, ERA recognizes the efficient recomputation of the volume flux fields from the given velocity fields and inserts the corresponding code.

3.1.3 Checkpointing

At the time-stepping level the directives allow for checkpointing that hierarchically splits the time-stepping loop. (cf. also [27], [28]). For the MITgcm, a three-level checkpointing scheme, ilustrated in Figure 3, has been adopted

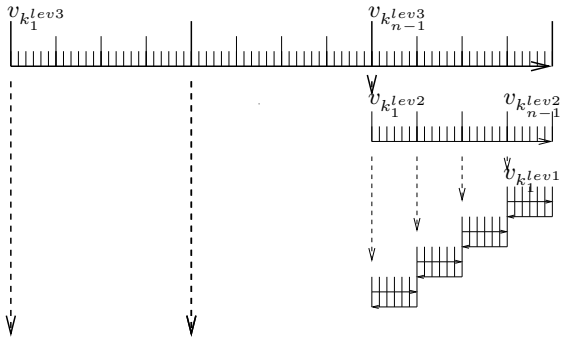


Figure 3: Schematic view of intermediate dump and restart for 3-level checkpointing.

$lev3$ 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 .

$lev2$ In a second step, each "$lev3$" subsection is divided into n^{lev2} subsections. The model picks up the last "$lev3$" saved state $v_{k_{n-1}^{lev3}}$ 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 $v_{k_{n-1}^{lev2}}$ 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 $v_n = v_{k_n^{lev1}}$ is reached and the model state of all preceding timesteps along the last "$lev1$" subsection are available. Thus, the adjoint can 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 $n_{timeSteps} = 77,760$ (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 $n_{timeSteps} = n_1 \cdot n_2 \cdot n_3$, the storing amount is drastically reduced. Pure recomputation would inccur a computation cost of $n_{timeSteps} = 77,760^2$.

The full model state of the two outer loops are stored to disk using an explicit TAMC directive, corresponding to a storing factor of $n_2+n_3 = 138$ times the dimension of the model state. The state here is defined as the set of quantities required to pick up the model integration at an intermediate timestep. The procedure for the innermost checkpointing loop differs from those of the two outer loops. First, required fields are stored to memory rather than to file to avoid I/O in this phase of computation. Second, insertions of store directives at the innermost loop are more intricate and

level3, level2 (file)			
	basic model state	14	
	INCLUDE_CD_CODE	7	
	EXACT_CONSERV	2	
	INCLUDE_EXTERNAL_FORCING_PACKAGE	18	

level1 (common)			
model			41
	calc_phi_hyd	2	
	convective_adjustment	6	
	dynamics	7	
	thermodynamics	26	
KPP			21
	kppmix	8	
	bldepth	6	
	blmix	2	
	dynamics	1	
	thermodynamics	4	
GM/Redi			29
	gmredi_calc_tensor	6	
	gmredi_slope_limit	13	
	gmredi_xtransport	2	
	gmredi_ytransport	2	
	dynamics	-	
	thermodynamics	6	
EXF			22
	exf_mapfields	4	
	the_main_loop	18	

Table 1: Number of fields stored to disk (level3, level2) and held in main memory (level1) for the basic model and additional packages.

require detailed knowledge of the code. Rather than storing the model state at each timestep, only those variables are stored, which appear in nonlinear expressions or state-dependent conditions. Furthermore, storing is invoked only if recomputation of these variables is expensive (`uVel`, `vVel` in the above example), otherwise they are recomputed (`uTrans`, `vTrans`). Note that this approach is more efficient than a storing of all input variables at the subroutine level. In most cases, only one specific variable at a specific place is needed rather than the full subroutine input fields. Note also, that multiple storing of the same variable within a subroutine may be necessary if this variable appears in several nonlinear expressions and changes its value in between. Storing efficiency relies crucially on identifying the right variable and the right place to store. Finally, directives may have to be accompanied (or may be avoided), occasionally, by additional measures to break data dependency flows. For instance, a `DO`-loop which

contains multiple nonlinear 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 [19].

Table 1 gives a summary of the number of arrays stored at the different checkpointing levels.

3.2 Adjoint dump and restart

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. 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, and which are needed in an adjoint restart. 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 is presently unique among existing AD tools to provide these capabilities. Through a simple TAF directive it generates code for divided adjont execution. Taking advantage of its checkpointing algorithm, the outermost checkpoints are also defined as dump/restart points for the adjoint state. Thus, forward/adjoint code execution can be limited to one segment of the length of the outermost checkpointing interval $\Delta = k_i^{lev3} - k_{i-1}^{lev3}$. In addition to dumping the model state $v_{k_i^{lev3}}$ and the adjoint state $adv_{k_i^{lev3}}$, the bookkeeping indices of the outermost checkpoint loop is saved to file for the start and the end of the interval.

In a consecutive adjoint code execution, the model state is recomputed from $v_{k_{i-1}^{lev3}}$ over one outer checkpoint interval to k_i^{lev3}. Then the adjoint state $adv_{k_i^{lev3}}$ is read, and the accumulation of adjoint sensitivities resumes up to k_{i-1}^{lev3}, at which place $adv_{k_{i-1}^{lev3}}$ is dumped.

The divided adjoint capability is of crucial practical importance, since it allows to run large scale adjoint ocean and climate models on HPC machines despite their strict batch execution limitations.

3.3 Exploiting formal self-adjointness

The character of the **DS** computational phase has important implications for adjoint computations. The equation solved in **DS**, is self-adjoint. Exploiting this fact in reverse mode, no derivative code needs to be generated. Instead the original, optimized code is invoked by providing a TAMC directive, thus saving substantial computing cost. Note that an iterative conjugate gradient method is implemented for the 2-dimensional, global elliptic problem. An explicit adjoint of this routine would have required substantial storing of required variables for each intermediate solver iteration (typically 50 to 200).

3.4 Parallel Implementation

In the following we discuss in more detail issues related to the scalability of the model and its adjoint. The approach chosen to generate efficient scalable adjoint code consists of retaining the parallel design of the forward code for the adjoint.

3.4.1 Adjoints of parallel support routines

In order to do so, substantial intervention into the original code was required to enable correct adjoint code generation. Table 2 summarizes the main parallel support primitives and their corresponding adjoint.

3.4.1.1 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 a halo region, owned by a neighboring tile, but needed for the computational stencil within a given computational phase. By means of overcomputing in the halo region the **PS** computational phase within which no communication is required can be extended to a large fraction of the full timestep phase. Following the **PS** phase, a communication intensive **DS** phase ensues during which processors will make calls to WRAPPER functions which can use MPI, OpenMP, or combination of both to communicate data between tiles (so-called exchanges) in order to keep the halo regions up-to-date. Furthermore, a global elliptic problem is solved which invokes global sum operations. 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. The adjoint code maintains the separation between communication-intensive **PS** phase and communictation-intensive **DS** phase (but in reverse order, and with appropriately modified function semantics). In addition, the use of WRAPPER functions is maintained by providing to each function a corresponding hand-written adjoint WRAPPER function. TAMC recognizes when and where to include these routines by means of directives provided by the user.

	operation/primitive	forward		reverse
•	communication (MPI,...):	send & assign	$\longleftrightarrow$	receive & accumulate
•	arithmetic (global sum,...):	gather	$\longleftrightarrow$	scatter
•	active parallel I/O:	read & assign	$\longleftrightarrow$	write & accumulate

Table 2: Summary of parallel support primitives and their hand-written adjoint.

3.4.1.2 Global arithmetric primitives Operations within the **DS** communication phase, for which a processor requires data outside of the overlap region of neighboring processors, communication libraries must be used, 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 involve the elliptic solver, and global averages e.g. when accumulating a sum over least square cost function). 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 routines are provided to TAMC.

3.4.1.3 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 $\mathcal{J}$ and $\frac{\partial \mathcal{J}}{\partial C}$. This isolation of "active" I/O simplifies AD code transformations. Read and write operations in forward mode are acompanied by corresponding write and read operations (plus variable reset), respectively, in adjoint mode required for active variables. The 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.

3.4.2 Analysis of model and adjoint scaling

By considering the algorithm and the details of the scalable formulation described above we can make some estimates of likely scaling for the different problem sizes as a function of key computational parameters.

3.4.2.1 PS phase: We first consider the prognostic phase. If we assume that the time to update the *halo* regions is proportional to their size and that the compute time is proportional to the effective domain size. Then we can write an approximate formula [1] for the time, $T_{\mathbf{PS}}$, to

[1] The formula will only be approximate because there are second order effects that mean that updating large halo regions is more efficient than updating small regions and similarly computing efficiency can vary with tile size and aspect ratio. Nevertheless the formula employed does give reasonable insights into scalability.

complete a **PS** cycle

$$T_{\mathbf{PS}} = \underbrace{\bar{\tau}_{flops} N_{PSops} N_r N_h}_{T_{\mathrm{PScomp}}} + \underbrace{\tau_{PSexch} N_{PSexch} N_r O_{\mathrm{halo}} L_{\mathrm{halo}}}_{T_{\mathrm{PSexch}}} \tag{4}$$

where

$\bar{\tau}_{flops}$	per grid point single arithmetic operation time $\bar{\tau}_{flops} = \bar{n}\,\tau_{flops}$
N_{PSops}	number of lines of arithmetic operations $N_{PSops} \sim 2000$
$\bar{n}$	average number of flops per code line
N_r	number of vertical layers
N_h	number of points in horizontal tile, including halo $N_h = \left(\frac{\tilde{N}_x}{P_x} + 2O_{\mathrm{halo}}\right)\left(\frac{\tilde{N}_y}{P_y} + 2O_{\mathrm{halo}}\right)$
τ_{PSexch}	time of per grid point exchange operation
N_{PSexch}	number of exchanged fields during **PS**
O_{halo}	width of halo region
L_{halo}	tile edge length, including halo $L_{\mathrm{halo}} = 2\left(\frac{\tilde{N}_x}{P_x} + 2O_{\mathrm{halo}}\right) + 2\left(\frac{\tilde{N}_y}{P_y} + 2O_{\mathrm{halo}}\right)$

In essence, $T_{\mathbf{PS}}$ is determined by the computational phase T_{PScomp} and the ensueing communication phase T_{PSexch}. The former is obtained by estimating the number of lines containing arithmetic operations N_{PSops}, the average number of flops per line, and the field dimension $N_r N_h$. The latter is obtained through the number of exchanges N_{PSexch} times the dimension of the exchanged fields and their overlaps (see the online documentation [21] for more details).

3.4.2.2 The DS phase: For **DS**, the diagnostic phase a similar estimation model can be made. This time however we need to account for additional global operations which sum up a scalar over all processors to calculate a dot product needed as part of the conjugate gradient solution procedure. The preconditioned conjugate gradient algorithm is a common procedure in codes designed for parallel computation. In ocean modeling it is widely used to solve for the surface pressure/height field. Accounting for the fact that is global connection established through a dot product and adjusting for the fact that the hydrostatic case we are considering only entails a two-dimensional Laplacian, yields formulae for the timing and scaling of the **DS** stage:

$$\begin{aligned} T_{\mathbf{DS}} = N_{DSiter} \cdot [& \underbrace{\bar{\tau}_{flops} N_{DSops} N_h}_{T_{\mathrm{DScomp}}} \\ & + \underbrace{\tau_{DSexch} N_{DSexch} O_{\mathrm{halo}} L_{\mathrm{halo}}}_{T_{\mathrm{DSexch}}} \\ & + \underbrace{\tau_{DSsum} N_{DSsum} \log_2(N_p)\ ]}_{T_{\mathrm{DSsum}}} \end{aligned} \tag{5}$$

In essence, the estimate is obtained as the sum of computations T_{DScomp}, exchanges T_{DSexch}, and global sums T_{DSsum} for evaluating a dot product

for each iteration of the conjugate gradient solver times the number of iterations required for convergence N_{DSiter}.

3.4.2.3 Adjoint model: The scaling of the adjoint model is to be compared to the total time

$$T = T_{\mathbf{PS}} + T_{\mathbf{DS}}$$

We limit our analysis to the innermost checkpointing loop. We already discussed the occurence of a factor of N incurred by the N-level checkpointing scheme, in our case $N = 3$ (an additional term comes through the I/O of the model state from/to disk by the outer checkpointing loops). Crucially, the adjoint code maintains

- domain decomposition
- the separation between PS and DS phase,

The timing for the adjoint code may thus again be split into

$$adT = adT_{\mathbf{PS}} + adT_{\mathbf{DS}}$$

We note that in view of the self-adjointness of the elliptic solver, the estimate for $adT_{\mathbf{DS}}$ is identical to that of $T_{\mathbf{DS}}$. We can thus limit our analysis to the term $adT_{\mathbf{PS}}$. It is difficult to give some generally valid precise estimate that would be based on forward code parameters only. This is because the complexity of the adjoint statement depends on the operations involved in the forward code statement. An upper bound of achievable scaling for an efficient adjoint may, nevertheless, be given. We first propose an equation and then discuss the contributions:

$$\begin{aligned} adT_{\mathbf{PS}} &= T_{\mathrm{PScomp}} \cdot \{ 1 + \gamma_{\mathrm{accum}} + \gamma_{\mathrm{reset}} + \gamma_{\mathrm{recomp}} + \gamma_{\mathrm{nonlin}} \} \\ &\quad + T_{\mathrm{store}} + T_{\mathrm{PSexch}} \end{aligned} \tag{6}$$

The 3 major contributions thus come from (i) T_{PScomp} itself which is augmented by a sum of factors $\sum_i \gamma_i$, (ii) T_{store}, a new term for the storing required for nonlinear and active variable-dependent conditional expressions, and (iii) T_{PSexch}, which remains unchanged, since the exchange pattern is unaltered and thus does not need to be discussed (we neglect here the slight difference in the number of FLOPs between, e.g. a gather vs. scatter, or a send vs. receive expression).

We first consider the term T_{store}. In keeping with the previous approach, an estimate may be given by

$$T_{\mathrm{store}} = 2\tau_{flops} N_{store} N_r N_h$$

The number of required storing per timestep in the innermost checkpointing loop can be inferred from Table 1 for the basic model and enhanced versions using different parameterization packages.

We next consider the factors γ which contribute to the adjoint computation in excess of the model computation (a comprehensive collection of adjoint code for various operations may be found in [19])

γ_{accum} In many cases, the accumulative character of the adjoint operation leads to an increase of the number of FLOPs of the adjoint statement as compared to the original statement. Two examples illustrate this:

(a) The following operation adds a FLOP to the adjoint statement:

$$\begin{array}{lrcll} \text{original} & y & = & c\,x & , \; c \text{ passive} \\ \text{adjoint} & adx & = & adx + c\,ady & \end{array}$$

(b) The following operation generates 2 lines of adjoint code:

$$\begin{array}{lrcl} \text{original} & y(i,j) & = & c_1\,x(i,j) + c_2\,x(i,j+1) \\ \text{adjoint} & adx(i,j+1) & = & adx(i,j+1) + c_2\,ady(i,j) \\ & adx(i,j) & = & adx(i,j) + c_1\,ady(i,j) \end{array}$$

It should, however, be noted that arithmetic statements which involve passive variables only, can be discarded altogether in the adjoint calculation. In summary, an upper bound would be $\gamma_{\text{accum}} = 2$, but a more realistic estimate would be somewhat less than 1.

γ_{reset} All expressions, except those of recursive form $x = f(x, y, \ldots)$, require the adjoint of the original l.h.s. variable to be reset (new assignments break dependency flow). Thus, in the above example (b), the adjoint statements have to be followed by a reset

$$ady(i,j) = 0.$$

Again, only a fraction of the number of arithmetic operations in the code will need this statement. Furthermore, for each line of arithmetic operations which comprises an average of $\bar{n}$ FLOPs the resetting consists of only one operation. Thus, an upper bound would be $\gamma_{\text{reset}} = 1/\bar{n}$, but a number less than this is likely.

γ_{recomp} This factor refers to efficient recomputations. An example is the volume transport `uTrans` introduced in Section 3.1.1, where it was needed in an adjoint operation. Since the variable `uVel` is availabe (through store/restore), `uTrans` can be readily computed as the product of the velocity field and its areal element, `uTrans = uvel*xa`. We estimate γ_{recomp} to be of the order of 0.2. Note that in the absence of appropriate storing/restoring, this factor could, in the best case go up to $N_{store} \sim 40$ to 100 or, in the worst case, when recomputations occur within loops over the domain, to $N_h^2 \sim (N_x \cdot N_y)^2$.

γ_{nonlin} For nonlinear expressions, the product rule is invoked, increasing the number of arithmetic operation compared to the original code (see Section 3.1.1). If N_{nonlin} refers to the number of lines involving nonlinear expressions and N_{ops} the total number of lines of arithmetic operations, a very rough estimate for γ_{nonlin} would be N_{nonlin}/N_{ops}. With N_{nonlin} being on the order of the number of required store directives in the innermost checkpointing loop, $N_{nonlin} \sim N_{store}$, which is between 40 and 100, we obtain $\gamma_{\text{nonlin}} \sim 1/20$

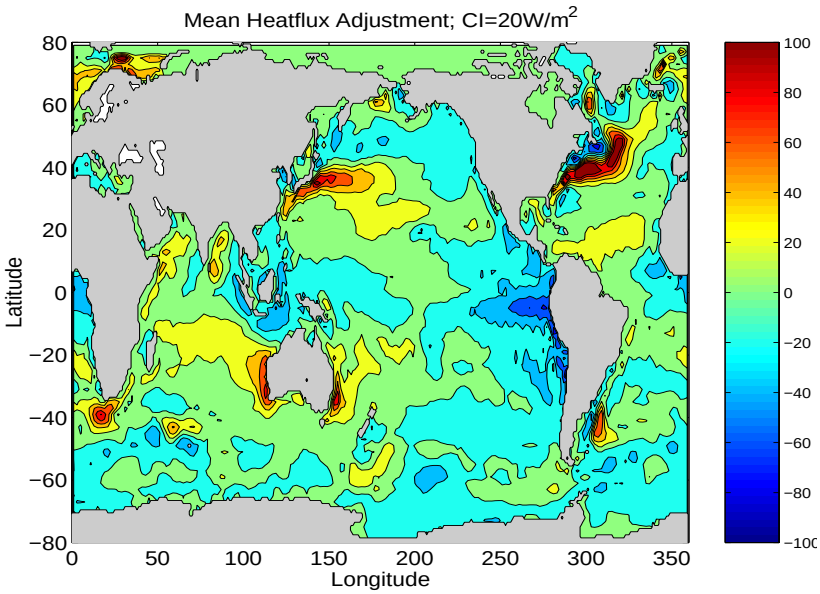


Figure 4: Mean changes in heat flux relative to the NCEP first guess fields (taken from [1]).

4 Applications

4.1 Global Ocean State Estimation

The model state of the underlying global estimation problem consists of 17 3-D and 2-D fields at a $1^\circ \times 1^\circ$ horizontal resolution and 23 vertical layers, yielding a model state of 5,659,200 elements which are updated at each time step (note, that a truly eddy-resolving setup would require a much higher horizontal resolution of about $1/10^\circ \times 1/10^\circ$). The model is run over a nine-year period between 1992 and 2000 at an hourly timestep. It is forced twice daily with realistic air-sea surface fluxes of momentum, heat and freshwater, provided by the National Centers for Environmental Prediction (NCEP) [29]. The inverse method iteratively reduces the model vs. data misfit (1), by successive modification to the controls, C, which consist of 3-dimensional initial temperature and salinity distributions, as well as time-varying surface forcing fields. To infer the update in the control variables, the cost gradient (2), is subject to a quasi-Newton variable storage line search algorithm [30]. The updated control variables serve as improved intial and boundary conditions in a consecutive forward/adjoint calculation. Thus, $\vec{\nabla}_C \mathcal{J}$, the outcome of the adjoint calculation, is a central ingredient for the optimization problem. By way of example, Figure 4, taken from [1], depicts the surface heat flux correction estimated from the optimization. The mean changes of the flux relative to the NCEP input fields are large over the area of the Gulf Stream and in the Eastern

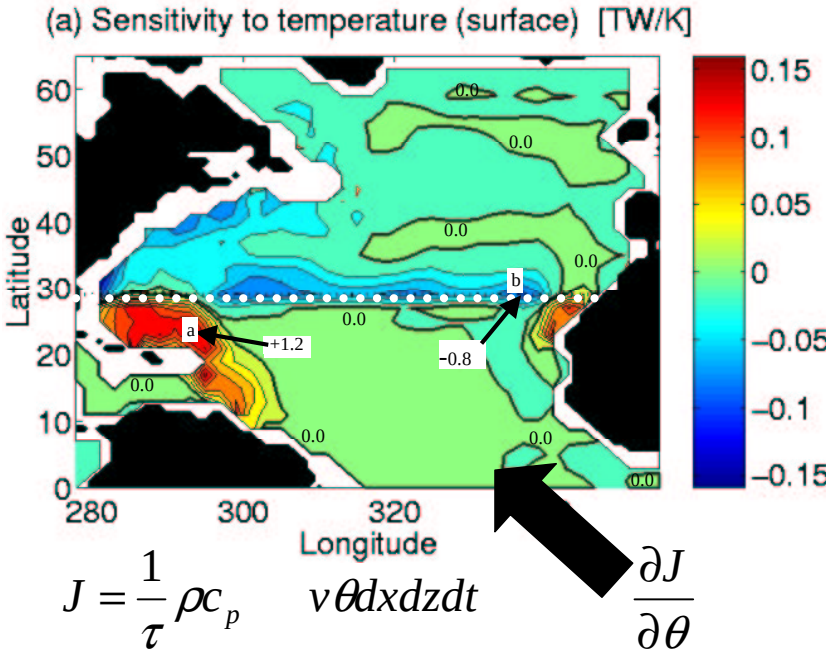


Figure 5: The sensitivity, $\frac{\partial \mathcal{J}}{\partial \theta}$, of the annual mean North Atlantic heat transport at 29°N, $\mathcal{J}$, to changes in temperature, θ, at the ocean surface. At point "a" a persistent change in θ of $+1°$ will produce a $1.2 \cdot 10^{12}$ Watt increase in annual mean poleward heat transport. The same change at point "b" produces a $0.8 \cdot 10^{12}$ Watt decrease (taken from [31]).

tropical Pacific. The heat flux corrections inferred here were shown to agree with independent studies of the NCEP heat flux analyses.

4.2 Sensitvity Analysis

Complementary to the estimation problems, the first adjoint sensitivity studies with a full general circulation model have been performed with the MITgcm [31], aiming at interpreting the adjoint or dual solution of the model state. As an example, Figure 5 depicts the sensitivity of the North Atlantic annual mean heat transport at 29°N to changes in surface temperature over a 1 year integration period, starting January 1, 1993.

The kinematic effect of advection of temperature anomalies in the western boundary current is readily apparent from the large upstream sensitivity pattern. Over the 1 year integration period a temperature anomaly can be carried by a 10 cm/s zonal velocity field over a 3000 km distance. Explaining the sensitivities in the interior ocean and off the African coast is more subtle, requiring the consideration of the dynamical effect of temperature and salinity anomalies on the density field and the corresponding changes in sea level.

5 Summary and Outlook

Reverse mode AD applications emerge as a powerful tool to address a suite of ocean science issues. Crucial 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. These features render computationally tractable adjoint code, despite the flow reversal in adjoint mode. Applied to the global time-dependent ocean circulation estimation problem, the code has been successfully used to solve a gigantic optimization problem. Complementary, 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 kinematical and dynamical mechanisms. Further reverse mode applications play a an equally important role in oceanographic research, and are being pursued using the MITgcm and its adjoint. They include optimal perturbation/singluar 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, the successor of TAMC which has enhanced features. Furthermore, ECCO supports efforts of the Adjoint Compiler Technology & Standards (ACTS) project 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.

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

References

[1] 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. J. Geophys. Res. (2000a) in press.

[2] Stammer, D., Wunsch, C., Giering, R., Eckert, C., Heimbach, P., Marotzke, J., Adcroft, A., Hill, C., Marshall, J.: Volume, heat and freshwater transports of the global ocean circulation 1993 –2000, estimated from a general circulation model constrained by WOCE data. J. Geophys. Res. (2000b) in press.

[3] Wunsch, C.: The ocean circulation inverse problem. Cambridge University Press, Cambridge (UK) (1996)

[4] 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

[5] Marshall, J., Hill, C., Perelman, L., Adcroft, A.: Hydrostatic, quasi-hydrostatic and nonhydrostatic ocean modeling. J. Geophys. Res. **102, C3** (1997a) 5,733–5,752

[6] Marshall, J., Adcroft, A., Hill, C., Perelman, L., Heisey, C.: Hydrostatic, quasi-hydrostatic and nonhydrostatic ocean modeling. J. Geophys. Res. **102, C3** (1997b) 5,753–5,766

[7] Stammer, D., Wunsch, C., Ponte, R.: De-aliasing of global high-frequency barotropic motions in altimetric observations. Geophys. Res. Lett. **27** (2000d) 1175–1178

[8] Wunsch, C.: Where do ocean eddy heat fluxes matter? J. Geophys. Res. **104, C6** (1999b) 13,235–13,249

[9] 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

[10] Marshall, J., Kushnir, Y., Battisti, D., Chang, P., Czaja, A., Hurrell, J., McCartney, M., Saravanan, R., Visbeck, M.: Atlantic climate variability. Int. J. Climatology **21** (2002) 1863–1898

[11] Clark, P., Pisias, N., Stocker, T., Weaver, A.: The role of the thermohaline circulation in abrubt climate change. Nature **415** (2002) 863–869

[12] TOPEX/Poseidon: `http://topex-www.jpl.nasa.gov/`. (URL)

[13] ERS-1/2: `http://earth.esa.int/ers/`. (URL)

[14] Wunsch, C., Stammer, D.: Satellite altimetry, the marine geoid, and the oceanic general circulation. Annu. Rev. Earth Planet. Sci. **26** (1998) 219–253

[15] WOCE: `http://www.woce.org/`. (URL)

[16] Jason-1: `http://sealevel.jpl.nasa.gov/mission/jason-1.html`. (URL)

[17] ARGO: Voyage of the argonauts. Nature **415** (2002) 954–955

[18] Griewank, A.: Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation. Volume 19 of Frontiers in Applied Mathematics. SIAM, Philadelphia (2000)

[19] Giering, R., Kaminski, T.: Recipes for adjoint code construction. ACM Transactions on Mathematical Software **24** (1998) 437–474

[20] ECCO: `http://ecco-group.org/`. (URL)

[21] MITgcm: `http://mitgcm.org/sealion/`. (URL)

[22] Hoe, J., Hill, C., A.Adcroft: A personal supercomputer for climate research. In: Proceedings of Supercomputing 1999, Portland (Oregon), USA. (1999) 15pp.

[23] Giering, R.: Tangent linear and adjoint model compiler. users manual 1.4 (TAMC version 5.2). Technical report (1999) http://www.autodiff.com/tamc/.

[24] TAF: `http://www.fastopt.com/`. (URL)

[25] Arakawa, A., Lamb, V.: Computational design of the basic dynamical processes of the ucla general circulation model. In: Methods in Computational Physics. Volume 17. Academic Press (1977) 174–267

[26] Giering, R., Kaminski, T.: Generating recomputations in reverse mode AD. In Corliss, G., Faure, C., eds.: Automatic Differentiation 2000: From Simulation to Optimization, Springer (2000c) in press.

[27] Griewank, A.: Achieving logarithmic growth of temporal and spatial complexity in reverse Automatic Differentiation. Optimization Methods and Software **1** (1992) 35–54

[28] Restrepo, J., Leaf, G., Griewank, A.: Circumvening storage limitations in variational data assimilation studies. SIAM J. Sci. Comput. **19** (1998) 1586–1605

[29] NCEP: `http://www.ncep.noaa.gov/`. (URL)

[30] Gilbert, J., Lemaréchal, C.: Some numerical experiments with variable-storage quasi-newton algorithms. Math. Programming **45** (1989) 407–435

[31] 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