Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 21 (2005) 1356–1371
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

An efficient exact adjoint of the parallel MIT General Circulation Model, generated via automatic differentiation

Patrick Heimbach[a,*], Chris Hill[a], Ralf Giering[b]

[a] *Department of Earth, Atmospheric and Planetary Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[b] *FastOpt, Schanzenstr. 36, 20357 Hamburg, Germany*

Available online 28 December 2004

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 versus 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 versus recomputation via the checkpointing method. Further savings are achieved by exploiting self-adjointness 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 the dumping of 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. Strategies to test the correctness of the adjoint-generated gradient are presented. The size of a typical adjoint application is illustrated for the case of the global ocean state estimation problem undertaken by the SIO-JPL-MIT Consortium "*Estimating the Circulation and Climate of the Ocean*" (ECCO). Results are given by way of example.

Keywords: Ocean/climate modeling; Ocean state estimation/data assimilation; Automatic differentiation; Adjoint/reverse mode of AD; Parallel/high performance computing

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

* Corresponding author. Fax: +1 617 253 4464.
E-mail addresses: heimbach@mit.edu (P. Heimbach), cnh@mit.edu (Chris Hill), ralf.giering@fastopt.de (R. Giering).
URLs: http://mitgcm.org, http://www.fastopt.de.

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2004.11.010

end, the project is applying advanced, mathematically rigorous, inverse techniques [3] to constrain a state-of-the-art parallel general circulation model, MITgcm [4–6], 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 $\mathcal{M}(\vec{u})$ and a set of observations $\vec{d}$ from the actual ocean in terms of a scalar cost, $\mathcal{J}$, thus,

$$\begin{aligned}\mathcal{J}(\mathcal{M}(\vec{u})) &= (\vec{\mathcal{H}}(\vec{v}) - \vec{d})^{\mathrm{T}}\mathbf{W}^{\mathrm{obs}}(\vec{\mathcal{H}}(\vec{v}) - \vec{d}) \\ &\quad + (\vec{C}^{\mathrm{opt}} - \vec{C}^{(0)})^{\mathrm{T}}\mathbf{W}^{\mathrm{C}}(\vec{C}^{\mathrm{opt}} - \vec{C}^{(0)}) \\ &= \sum_{i=1}^{n_{\mathrm{obs}}}(\mathcal{H}_i(\vec{v}) - d_i)W_{ii}^{\mathrm{obs}}(\mathcal{H}_i(\vec{v}) - d_i) \\ &\quad + \sum_{j=1}^{n_{\mathrm{c}}}(C_j^{\mathrm{opt}} - C_j^{(0)})W_{jj}^{\mathrm{C}}(C_j^{\mathrm{opt}} - C_j^{(0)}).\end{aligned} \tag{1}$$

Here, $\vec{u}$ represents the full set of independent variables (initial and time-dependent boundary conditions), $\vec{v} = \mathcal{M}(\vec{u})$ the model state at each time step, $\mathcal{H}_i$ the operator projecting the model state onto the ith observation d_i, $\mathbf{W}^{\mathrm{obs}}$ and $\mathbf{W}^{\mathrm{C}}$ the data error and a priori control error variance matrices, respectively. Denoting a subset of the independent variables as adjustable "controls" $\vec{C}$, the simulated state $\vec{v} = \mathcal{M}(\vec{C})$, can be optimized to minimize $\mathcal{J}$ over the n_{obs} observation points. The optimized controls, $\vec{C}^{\mathrm{opt}}$, render a numerically simulated ocean state, $\mathcal{M}(\vec{C}^{\mathrm{opt}})$, 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 occurring intrinsic oceanic and coupled atmosphere–ocean–cryosphere variability have become widely appreciated, operating on many timescales, 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 characterized by a cost function $\mathcal{J}$ that spans 9 years of planetary scale ocean simulations and observations, consisting of $n_{\mathrm{obs}} = 10^8$ observational elements. Major 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 2–3-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.

In addition to the observational elements, $\mathcal{J}$ contains penalty terms for each control variable which account for the a priori error estimate of the controls (the a posteriori estimate of which would be obtained through the Hessian, i.e. the second derivative of $\mathcal{J}$ with respect to the controls). The a priori error was derived from climatological variances of the respective quantities, and taking into account the scales which the model resolves versus those which have to be considered as noise.

The set of independent or control variables which are varied to optimize $\mathcal{J}$ consists of three-dimensional fields for the initial temperature and salinity, as well as two-dimensional daily-varying surface forcing fields of

heat, freshwater, and momentum (zonal and meridional wind stress) fluxes. Compressed to a one-dimensional vector C which contains wet points only, the size of the control space is $n_c = 1.5 \times 10^8$.

The model state, i.e. the set of prognostic variables which are stepped forward in time by the underlying system of differential equations consists of 17 three-dimensional and 2 two-dimensional fields. A global configuration at a $1^\circ \times 1^\circ$ horizontal resolution with 23 vertical layers yields a model state of dimension 2×10^7.

Thus, with a gradient, $\frac{\partial \mathcal{J}}{\partial C}$ of dimension 1.5×10^8, and a model Jacobian (i.e. the derivative of the mapping of the control space to the model state space, $\frac{\partial \mathcal{M}}{\partial C}$) of dimension $2 \times 10^7 \cdot 1.5 \times 10^8 = 3 \times 10^{15}$, even allowing for some sparsity, the computation of the full model Jacobian 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

Minimizing $\mathcal{J}$, 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,

$$\min_{C} \mathcal{J}(\vec{C}) \quad \Rightarrow \quad \vec{\nabla}_C \, \mathcal{J}(\vec{C}, \vec{v}(\vec{C}))\Big|_{\vec{C}=\vec{C}^{\mathrm{opt}}} = 0. \qquad (2)$$

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 Eq. (1). For the state estimation cost function (1) the gradient assumes the general form

$$\vec{\nabla}_C \mathcal{J}^{\mathrm{T}} = 2M^{\mathrm{T}} \cdot H^{\mathrm{T}} \cdot W \cdot (\mathcal{H}(\vec{v}) - \vec{d}). \qquad (3)$$

Here, M denotes the model Jacobian with M^{T} its adjoint, and H the differential of the projection operator with H^{T} its adjoint. Thus, the gradient is proportional to the adjoint of the model Jacobian times the adjoint of the projection operator differential times the model versus data misfit.

Automatic differentiation (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, 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 five to six 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. Note, that the model code is formulated in such a way that the projection operator (i.e. its specific realization for each data type, both altimetric and hydrographic) is part of the differentiated code, and thus intrinsically tied to the automatic adjoint code generation.

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 model's software implementation in a parallel computing environment are described in Section 2. 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 MIT General Circulation Model (MITgcm) is rooted in a general purpose grid-point algorithm

INITIALIZE. Define geometry, initial flow and tracer distributions
FOR each time step n DO
PS
Active I/O.
Step forward state. $\vec{v}^n = \vec{v}^{n-1} + \Delta t(\vec{G}_v^{n-\frac{1}{2}} - \vec{\nabla} p^{n-\frac{1}{2}})$
Get time derivatives. $\vec{G}_v^{n+\frac{1}{2}} = \mathbf{gv}(\vec{v}, b)$
Get hydrostatic p. $p_{hy}^{n+\frac{1}{2}} = \mathbf{hy}(b)$
Update halo (exchange).
DS
Solve for pressure. $\vec{\nabla}_h \cdot H\vec{\nabla}_h p_s^{n+\frac{1}{2}} = \vec{\nabla}_h \cdot \overline{\vec{G}_{v_h}^{n+\frac{1}{2}}}^H - \vec{\nabla}_h \cdot \overline{\vec{\nabla}_h p_{hy}^{n+\frac{1}{2}}}^H$
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 ($n, n-1, \ldots$). **DS** involves finding a two-dimensional pressure field p_s ensuring the flow $\vec{v}$ 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 **gv**() function and yield the buoyancy, b.

that solves the Boussinesq form of the Navier–Stokes equations for an incompressible fluid, hydrostatic or fully non-hydrostatic, in a curvilinear framework in the present context on a three-dimensional longitude (λ), latitude (ϕ), depth (r) grid. 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 discretized fields of potential temperature θ, salinity S, velocity vector $\vec{v} = (u, v, w)$, 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 Fig. 1. The two phases **PS** and **DS** within each timestep 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 involve iteration for which an iterative preconditioned conjugate gradient scheme is used.

2.2. Parallelism via domain decomposition

Finite-volumes provide a natural model for parallelism. Fig. 2a shows schematically a decomposition into sub-domains (tiles) 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. To avoid communication during this computational phase of values from neighboring tiles required by the computational stencil at each grid point within a tile, an overlap region (halo) is associated to each tile. The size of the halo is determined by the computational stencil of the differential operators encountered in the model equations. 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 that 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 communication layer in a custom software library called WRAPPER [22]. The performance critical primitives in the WRAPPER communication layer are illustrated in Fig. 2b. Significantly, for AD, the WRAPPER operations are all linear combination and permutations of distributed data.

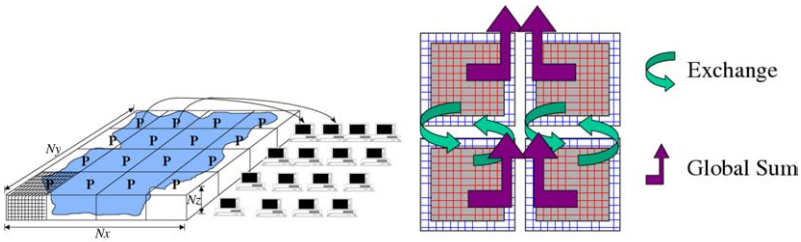

Fig. 2. Panel (a) shows a hypothetical domain of total size $N_x N_y N_z$. The domain is decomposed in two dimensions along the N_x and N_y directions. Whenever a processor wishes to transfer data between tiles or communicate with other processors it calls a special function in a WRAPPER support layer. Three 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

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 Fortran) developed by 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 model 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 elementary Jacobian matrices of the full code's compositional elements. The processed MITgcm code has about 40,000 lines and the adjoint code about 37,000 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 theoretically extremely efficient in computing gradients of 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 versus 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 parameterization schemes),
- conditional code execution and switches 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 model 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; orders of magnitude of required recomputations are very difficult to estimate since they may involve, depending on the type and occurrence of nonlinearities or switches, full array recomputations within three-dimensional array loops, and/or full trajectory recomputations within each of the intermediate checkpointing loops, and/or combinations thereof.

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):

$$F^x = \overline{U}_y^j \overline{v}_x^i, \tag{4}$$

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 $\mathrm{m}\,\mathrm{s}^{-1}$, 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 versus recomputation by TAMC

TAMC provides two crucial features to balance the amount of recomputation versus 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 appropriate 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

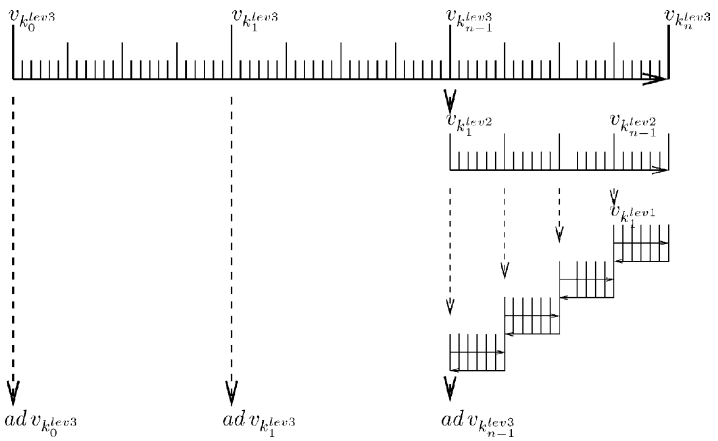

Fig. 3. Schematic view of intermediate dump and restart for 3-level checkpointing.

three-level checkpointing scheme, illustrated in Fig. 3, has been adopted.

(lev3) The model trajectory is first subdivided into n^{lev3} subsections with limits indexed $k_0^{\text{lev3}}, \ldots, k_n^{\text{lev3}}$. The model is integrated along the full trajectory, and the state stored at every k_i^{lev3}th timestep up to k_{n-1}^{lev3}. The integration to the final state $v_{k_n^{\text{lev3}}}$ is performed to evaluate the final cost function.

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

(lev1) Finally, the model picks up at the last intermediate saved state $v_{k_{n-1}^{\text{lev2}}}$ and is integrated forward in time along the last lev2-subsection. Within this subsection only, the model state is stored at every timestep. Thus, the final state $v_n = v_{k_n^{\text{lev1}}}$ is reached and the model state of all preceding timesteps along the last "lev2" 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_0^{lev3}.

The 3-level checkpointing requires a total of three forward and one adjoint integrations, 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 $n_{\text{timeSteps}} = 77{,}760$ (corresponding to a 9-year integration at an hourly timestep) into a hierarchy of three levels of sub-intervals $n_1 = 24$, $n_2 = 30$, $n_3 = 108$ with $n_{\text{timeSteps}} = n_1 \cdot n_2 \cdot n_3$, the storing amount is drastically reduced. Pure recomputation would incur a computation cost of $n_{\text{timeSteps}} = 77{,}760^2$.

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. Firstly, required fields are stored to memory rather than to file to avoid I/O in this phase of computation. Secondly, insertions of store directives at the innermost loop are more intricate and presume detailed knowledge of the code. Rather than storing the model state at each timestep, only those variables are stored, which are required, i.e. 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 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:

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

Level3, Level2 (file)	
Basic model state	14
INCLUDE_CD_CODE	7
EXACT_CONSERV	2
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

(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 overhead 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 adjoint execution. Taking advantage of its checkpointing algorithm, the outermost checkpoints are at the same time 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 $\Delta = k_i^{\mathrm{lev3}} - k_{i-1}^{\mathrm{lev3}}$. In addition to dumping the model state $v_{k_i^{\mathrm{lev3}}}$ and the adjoint state $\mathrm{adv}_{k_i^{\mathrm{lev3}}}$, 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 $v_{k_{i-1}^{\mathrm{lev3}}}$ over one outer checkpoint interval to k_i^{lev3}. Then the adjoint state $\mathrm{adv}_{k_i^{\mathrm{lev3}}}$ is read, and the accumulation of adjoint sensitivities resumes up to k_{i-1}^{lev3}, at which place $\mathrm{adv}_{k_{i-1}^{\mathrm{lev3}}}$ is dumped.

The divided adjoint capability is of crucial practical importance, since it enables the fitting of long-term, large-scale adjoint ocean and climate model calculations 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 two-dimensional, global elliptic problem. An explicit adjoint of this routine would have required substantial storing of required variables for each intermediate solver iteration (typically 50–200).

3.4. Parallel implementation

In the following we discuss 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 model code for the adjoint.

3.4.1. Adjoints of parallel support routines

In order to generate efficient scalable adjoint code, substantial intervention into the original code was required. Table 2 summarizes the main parallel support primitives and their corresponding adjoint.

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

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

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 support libraries for portable parallel programming such as Message Passing Interface (MPI), OpenMP, or combination thereof 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 compute-intensive **PS** phase and communication-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.

3.4.1.2. Global arithmetic 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 and reduced to 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. Fig. 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 model code are accompanied 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. 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,

[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.

T_{PS}, to complete a **PS** cycle

$$T_{\mathbf{PS}} = \underbrace{\bar{\tau}_{\mathrm{flops}} N_{\mathrm{PSops}} N_{\mathrm{r}} N_{\mathrm{h}}}_{T_{\mathrm{PScomp}}} + \underbrace{\tau_{\mathrm{PSexch}} N_{\mathrm{PSexch}} N_{\mathrm{r}} O_{\mathrm{halo}} L_{\mathrm{halo}}}_{T_{\mathrm{PSexch}}}, \quad (5)$$

where $\bar{\tau}_{\mathrm{flops}}$ is the per grid-point single arithmetic operation time $\bar{\tau}_{\mathrm{flops}} = \bar{n}\tau_{\mathrm{flops}}$; N_{PSops} the number of lines of arithmetic operations $N_{\mathrm{PSops}} \sim 2000$; $\bar{n}$ the average number of flops per code line; N_{r} the number of vertical layers; N_{h} the number of points in horizontal tile, including halo $N_{\mathrm{h}} = \left(\frac{\tilde{N}_x}{P_x} + 2O_{\mathrm{halo}}\right)\left(\frac{\tilde{N}_y}{P_y} + 2O_{\mathrm{halo}}\right)$; τ_{PSexch} the time of per grid-point exchange operation; N_{PSexch} the number of exchanged fields during **PS**; O_{halo} the width of halo region; L_{halo} the 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 ensuing 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 $N_{\mathrm{r}}N_{\mathrm{h}}$. 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 a global connection is established through a dot product and adjusting for the fact that the hydrostatic case we are considering only entails a two-dimensional Laplacian, yield formulae for the timing and scaling of the **DS** stage:

$$T_{\mathbf{DS}} = N_{\mathrm{DSiter}}\left[\underbrace{\bar{\tau}_{\mathrm{flops}} N_{\mathrm{DSops}} N_{\mathrm{h}}}_{T_{\mathrm{DScomp}}} + \underbrace{\tau_{\mathrm{DSexch}} N_{\mathrm{DSexch}} O_{\mathrm{halo}} L_{\mathrm{halo}}}_{T_{\mathrm{DSexch}}} + \underbrace{\tau_{\mathrm{DSsum}} N_{\mathrm{DSsum}} \log_2(N_{\mathrm{P}})}_{T_{\mathrm{DSsum}}}\right]. \quad (6)$$

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}. The global sum is assumed to scale as $\log_2$ of the number of processors N_{P}.

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 occurrence 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

$$\mathrm{ad}\, T = \mathrm{ad}\, T_{\mathbf{PS}} + \mathrm{ad}\, T_{\mathbf{DS}}.$$

We note that in view of the self-adjointness of the elliptic solver, the estimate for ad $T_{\mathbf{DS}}$ is identical to that of $T_{\mathbf{DS}}$. We can thus limit our analysis to the term ad $T_{\mathbf{PS}}$. It is difficult to give some generally valid precise estimate that would be based on model code parameters only. This is because the complexity of the adjoint statement depends on the operations involved in the

model 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:

$$\mathrm{ad}T_{\mathbf{PS}} = T_{\mathrm{PScomp}}\left\{1+\gamma_{\mathrm{accum}}+\gamma_{\mathrm{reset}}+\gamma_{\mathrm{recomp}} +\gamma_{\mathrm{nonlin}}\right\}+T_{\mathrm{store}}+T_{\mathrm{PSexch}}. \quad (7)$$

The three 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 versus scatter, or a send versus receive expression).

We first consider the term T_{store}, the time spent in the innermost checkpointing loop to store/restore required variables to/from common blocks. In keeping with the previous approach, an estimate may be given by

$$T_{\mathrm{store}} = 2\tau_{\mathrm{flops}} N_{\mathrm{store}} N_{\mathrm{r}} N_{\mathrm{h}}.$$

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:

original $\quad y = cx$(c passive)

adjoint $\quad \mathrm{ad}\,x = \mathrm{ad}\,x + c\,\mathrm{ad}\,y.$

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

original $\quad y(i,j) = c_1\,x(i,j) + c_2\,x(i,j+1)$

adjoint $\quad \mathrm{ad}\,x(i,j+1) = \mathrm{ad}\,x(i,j+1) + c_2\,\mathrm{ad}\,y(i,j),$

$\mathrm{ad}\,x(i,j) = \mathrm{ad}\,x(i,j) + c_1\,\mathrm{ad}\,y(i,j).$

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 $\gamma_{\mathrm{accum}} = 2$, 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

$$\mathrm{ad}\,y(i,j) = 0.$$

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 $\gamma_{\mathrm{reset}} = 1/\bar{n}$, 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 available (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 $N_{\mathrm{store}} \sim 40$–$100$ or, in the worst case, when recomputations occur within loops over the domain, to $N_{\mathrm{h}}^2 \sim (N_x \cdot N_y)^2$.

(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 $N_{\text{nonlin}}/N_{\text{ops}}$. With N_{nonlin} being on the order of the number of required store directives in the innermost checkpointing loop, $N_{\text{nonlin}} \sim N_{\text{store}}$, which is between 40 and 100, we obtain $\gamma_{\text{nonlin}} \sim 1/20$.

An approximate factor of 5.5 of a full exact adjoint calculation over a forward calculation of the nonlinear parent model has been found for a variety of MITgcm setups involving different configurations and packages. This is important since the inclusion of a specific package, e.g. the parameterization of vertical mixing (computation of viscosity and diffusivity coefficients) may incur additional complexities in the reverse flow dependency and the storing/recomputation requirements due to additional nonlinear expressions and switches. Thus, each addition of code requires careful analysis and possibly update in user-defined directives to retain the adjoint code's efficiency and storing versus recomputation balance.

3.5. Correctness of the adjoint and gradient checks

A crucial element in adjoint code development and maintenance of adjoint code along with the ongoing development of the full model is to ensure correctness of the adjoint. We achieve this by comparing components of the adjoint-generated gradient to components of a gradient obtained (i) via finite differences, or (ii) via the tangent linear model. We note that in this context we prefer the notion of "correctness" rather than "exactness" of the gradient. As described in [18], a basic feature of AD is to obtain, in principle, derivatives which are "exact" up to machine precision, in contrast to finite difference derivatives, whose precision may depend substantially on the finite difference scheme and magnitude of perturbation chosen.

- *Finite difference gradient checks*

  For a given component C_i (or a set of components) of the control vector we compute the centered-finite difference of the perturbed cost function

$$G_i^{\text{fd}} = \frac{\mathcal{J}(C_i + \epsilon) - \mathcal{J}(C_i - \epsilon)}{2\epsilon} \tag{8}$$

  and compare it to the adjoint-generated gradient G_i^{ad} by considering the deviation of the ratio $G_i^{\text{fd}}/G_i^{\text{ad}}$ from 1,

$$R_i^{\text{fd}} = 1 - \frac{G_i^{\text{fd}}}{G_i^{\text{ad}}}. \tag{9}$$

  This requires two full model runs for each component C_i to evaluate the perturbed cost functions $\mathcal{J}(C_i \pm \epsilon)$.
- *Tangent linear gradient checks*

  Alternatively or additionally, we may explore the availability of the tangent linear model, also obtained via TAF. However, in contrast to the reverse mode, which yields the full gradient $\vec{\nabla}_C \mathcal{J}^{\text{T}}$ with one adjoint integration, the forward mode only yields one component per tangent linear integration. It is obtained by setting the ith component $\delta C_i = 1$ and all other components $\delta C_j = 0,\ j \neq i$,

$$G_i^{\text{tl}} = \vec{\nabla}_C \mathcal{J} \cdot \delta\vec{C} = (\vec{\nabla}_C \mathcal{J})_i. \tag{10}$$

  Thus, one tangent linear integration is required for each component C_i whose gradient is to be tested. Again, we consider the ratio

$$R_i^{\text{tl}} = 1 - \frac{G_i^{\text{tl}}}{G_i^{\text{ad}}}. \tag{11}$$

For the fully fledged global model setup values of R_i^{fd} were found to be on the order of 10^{-3} or less. Values for R_i^{tl} were consistently smaller (10^{-5} or less).

4. Applications

4.1. Global ocean state estimation

The model state of the underlying global estimation problem consists of 17 three- and two-dimensional 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 $1/10^\circ \times 1/10^\circ$ as well as

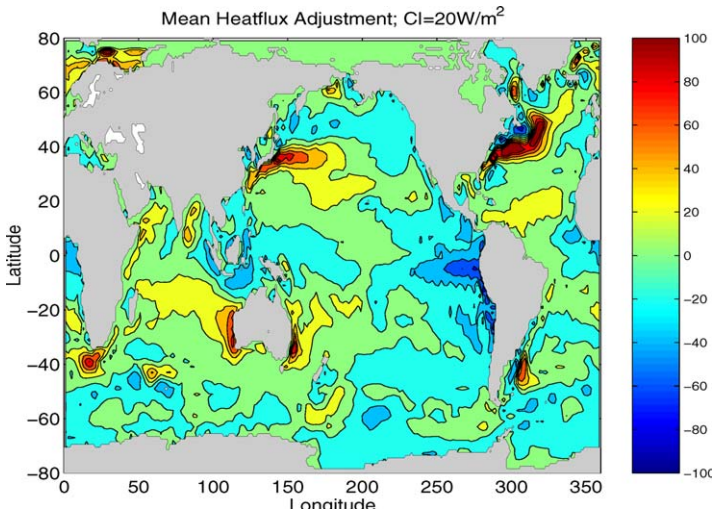

Fig. 4. Mean changes in heat flux relative to the NCEP first guess fields (taken from [1]).

higher vertical resolution, and, to remain numerically stable, a shorter model time step). The model is run over a 9-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 versus data misfit (1), by successive modification to the controls, C, which consist of three-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 initial 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, Fig. 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 tropical Pacific. The heat flux corrections inferred here were shown to agree with independent studies of the NCEP heat flux analyses.

4.2. Sensitivity analysis

Complementary to the estimation problem, the first adjoint sensitivity studies with a full general circulation model and its adjoint generated by means of AD have been performed with the MITgcm [31], aiming at interpreting the adjoint or dual solution of the model state. In this context the cost function is a much simpler one. Instead of a least square misfit between model and data over a large amount of data points, a scalar quantity is considered, usually diagnosed after a model integration. As an example, Fig. 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, thus,

$$\mathcal{J} = \frac{1}{\tau} c_p \rho_0 \int\int\int v T \, \mathrm{d}\lambda \, \mathrm{d}r \, \mathrm{d}t$$

which assumes a value of 1.2 PW (Peta Watt) in the Atlantic at 29°N. The control variable is the initial sea surface temperature (SST) distribution.

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 sensi-

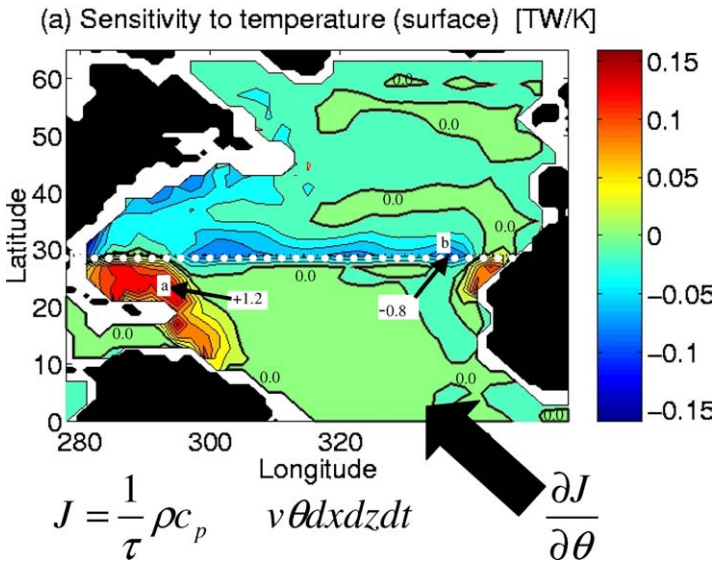

Fig. 5. The sensitivity, $\frac{\partial \mathcal{J}}{\partial \theta}$, of the annual mean North Atlantic heat transport at 29°N, $\mathcal{J}$, to changes in temperature, θ, at the ocean surface. At point "a" a persistent change in θ of $+1^{\circ}$ will produce a 1.2×10^{12} W increase in annual mean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

tivities 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 model 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 an equally important role in oceanographic research, and are being pursued using the MITgcm and its 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, 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 paper is a contribution to the ECCO project, supported by NOPP, and with funding from NASA, NSF and ONR. Many members of the ECCO Consortium under the lead of D. Stammer have contributed.

Valuable comments by two anonymous reviewers are gratefully acknowledged.

Appendix A. TAF: a short description

A.1. Overview

TAF is a source-to-source translator for Fortran 77-95 code, i.e. TAF accepts Fortran 77-95 code as input, applies a semantic transformation, and generates Fortran 77-95 code as output. TAF supports several semantic transformations. The most important one is Automatic Differentiation (AD), i.e. generation of code for evaluation of the first-order derivative (Jacobian matrix). This generated code can operate in forward or reverse mode (tangent linear or adjoint model). TAF can generate code to evaluate Jacobian times vector products or the full Jacobian. Higher order derivative code is generated by applying TAF multiple times.

Another TAF transformation is Automatic Sparsity Detection (ASD), i.e. efficient determination of the sparsity structure of the Jacobian matrix. This transformation is important, because the Jacobian's sparsity pattern can be exploited to render the evaluation of the Jacobian more efficient.

A.2. Special features

A.2.1. Analyses

TAF normalises the code and applies a control flow analysis. TAF applies an intraprocedural data dependence and an interprocedural data flow analysis. Given the independent and dependent variables of the specified top-level routine, TAF determines all active routines and variables and produces derivative code only for those.

A.2.2. Directives

TAF accepts several kinds of directives. Using the reverse mode automatic storing/reading of required values is triggered by directives. Multi level checkpointing can be generated by splitting a loop and inserting directives. Generating memory efficient adjoint code for iterative solvers can be triggered by inserting a directive. Black box (library) routines are handled by specifying flow information via directives.

A.2.3. Parallelization

TAF offers basic support of OpenMP and MPI.

A.2.4. Readability

TAF generated code is structured an well readable. The derived structures are closely inherited from the original code, as are names (variables, subroutines).

Details on the structure of directives and TAF-generated code can be found in [19].

References

[1] D. Stammer, C. Wunsch, R. Giering, C. Eckert, P. Heimbach, J. Marotzke, A. Adcroft, C. Hill, J. Marshall, The global ocean circulation and transports during 1992–1997, estimated from ocean observations and a general circulation model, J. Geophys. Res. 107 (C9) (2002) 3118.
[2] D. Stammer, C. Wunsch, R. Giering, C. Eckert, P. Heimbach, J. Marotzke, A. Adcroft, C. Hill, J. Marshall, 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. 108 (C1) (2003) 3007.
[3] C. Wunsch, The ocean Circulation Inverse Problem, Cambridge University Press, Cambridge, UK, 1996.
[4] C. Hill, J. Marshall, Application of a parallel Navier–Stokes model to ocean circulation in parallel computational fluid dynamics, in: Proceedings of Parallel Computational Fluid Dynamics, Elsevier Science, New York, 1995, pp. 545–552.
[5] J. Marshall, C. Hill, L. Perelman, A. Adcroft, Hydrostatic, quasi-hydrostatic and non-hydrostatic ocean modeling, J. Geophys. Res. 102 (C3) (1997) 5733–5752.
[6] J. Marshall, A. Adcroft, C. Hill, L. Perelman, C. Heisey, Hydrostatic, quasi-hydrostatic and non-hydrostatic ocean modeling, J. Geophys. Res. 102 (C3) (1997) 5753–5766.
[7] D. Stammer, C. Wunsch, R. Ponte, De-aliasing of global high-frequency barotropic motions in altimetric observations, Geophys. Res. Lett. 27 (2000) 1175–1178.
[8] C. Wunsch, Where do ocean eddy heat fluxes matter?, J. Geophys. Res. 104 (C6) (1999) 13235–13249.
[9] J. Neelin, D. Battisti, A. Hirst, F. Jin, Y. Wakata, T. Yamagata, S. Zebiak, ENSO theory, J. Geophys. Res. 103 (1998) 14261–14290.
[10] J. Marshall, Y. Kushnir, D. Battisti, P. Chang, A. Czaja, J. Hurrell, M. McCartney, R. Saravanan, M. Visbeck, Atlantic climate variability, Int. J. Climatol. 21 (2002) 1863–1898.
[11] P. Clark, N. Pisias, T. Stocker, A. Weaver, The role of the thermohaline circulation in abrupt climate change, Nature 415 (2002) 863–869.
[12] TOPEX/Poseidon. http://topex-www.jpl.nasa.gov/.
[13] ERS-1/2. http://earth.esa.int/ers/.
[14] C. Wunsch, D. Stammer, 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/.
[16] Jason-1. http://sealevel.jpl.nasa.gov/mission/jason-1.html/.
[17] ARGO, Voyage of the argonauts, Nature 415 (2002) 954–955.
[18] A. Griewank, Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation, vol. 19 of Frontiers in Applied Mathematics, SIAM, Philadelphia, 2000.
[19] R. Giering, T. Kaminski, Recipes for adjoint code construction, ACM Trans. Math. Softw. 24 (1998) 437–474.
[20] ECCO. http://www.ecco-group.org/.
[21] MITgcm. http://mitgcm.org/.
[22] J. Hoe, C. Hill, A. Adcroft, A personal supercomputer for climate research, in: Proceedings of Supercomputing 1999, Portland, OR, USA, 1999, 15 pp.
[23] R. Giering, Tangent linear and Adjoint Model Compiler, Users Manual 1.4 (TAMC Version 5.2), MIT/JPL/FastOpt, 1999. http://www.autodiff.com/tamc/.
[24] TAF. http://www.fastopt.com/.
[25] A. Arakawa, V. Lamb, Computational design of the basic dynamical processes of the UCLA general circulation model, in: Methods in Computational Physics, vol. 17, Academic Press, 1977, pp. 174–267.
[26] R. Giering, T. Kaminski, Generating recomputations in reverse mode AD, in: G. Corliss, A. Griewank, C. Fauré, L. Hascoet, U. Naumann (Eds.), Automatic Differentiation of Algorithms: From Simulation to Optimization, Springer Verlag, 2002, pp. 283–291.
[27] A. Griewank, Achieving logarithmic growth of temporal and spatial complexity in reverse Automatic Differentiation, Optim. Meth. Softw. 1 (1992) 35–54.
[28] J. Restrepo, G. Leaf, A. Griewank, Circumventing storage limitations in variational data assimilation studies, SIAM J. Sci. Comput. 19 (1998) 1586–1605.
[29] NCEP. http://www.ncep.noaa.gov/.
[30] J. Gilbert, C. Lemaréchal, Some numerical experiments with variable-storage quasi-Newton algorithms, Math. Programm. 45 (1989) 407–435.
[31] J. Marotzke, R. Giering, K. Zhang, D. Stammer, C. Hill, T. Lee, Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport variability, J. Geophys. Res. 104 (C12) (1999) 29529–29547.

Patrick Heimbach joined the physical oceanography team at the Massachusetts Institute of Technology after finishing his Ph.D. at the Max-Planck Institute for Meteorology, Hamburg, Germany, in 1998. His research focuses on applying mathematical rigorous techniques to the modeling of the complex atmosphere–ocean climate system and the quantification of its uncertainties. He has been involved in the first dynamical consistent decadal inverse modeling study by the ECCO project.

Chris Hill is a researcher at the Massachusetts Institute of Technology, Department of Earth Atmospheric and Planetary Sciences. His current research interests center on the application of advanced computational technologies to challenging Earth science problems. Contact him at cnh@mit.edu.

Ralf Giering is co-owner and manager of FastOpt. He received both diploma in physics (1990) and Ph.D. (1995) in oceanography from University of Hamburg. His current research interests include automatic differentiation and optimisation/data assimilation.