Tangent Linear and Adjoint Versions of NASA/GMAO's Fortran-90 Global Weather Forecast Model

Ralf Giering[1], Thomas Kaminski[1], Ricardo Todling[2], Ronald Errico[2], Ronald Gelaro[2], and Nathan Winslow[2]

[1]FastOpt, Hamburg, Germany (`http://www.FastOpt.com`)
[2]Global Modeling and Assimilation Office, NASA/GSFC, Greenbelt, Maryland, USA (`http://gmao.gsfc.nasa.gov`)

Summary. The NASA finite-volume General Circulation Model (fvGCM) is a three-dimensional Navier-Stokes solver that is being used for quasi-operational weather forecasting at NASA/GMAO. By means of the automatic differentiation tool TAF, efficient tangent linear and adjoint versions are generated from the Fortran-90 source code of fvGCM's dynamical core. fvGCM's parallelisation capabilities based on OpenMP and MPI have been transferred to the tangent linear and adjoint codes. For OpenMP, TAF automatically inserts corresponding OpenMP directives in the derivative code. For MPI, TAF generates interfaces to hand-written tangent linear and adjoint wrapper routines. TAF also generates a scheme that allows the tangent linear and adjoint models to linearise around an external trajectory of the model state. The generation procedure is set up in an automated way, allowing quick updates of the derivative codes after modifications of fvGCM.

Keywords: automatic differentiation, tangent linear model, adjoint model, source-to-source translation, Fortran-90

1 Introduction

Many applications in dynamic meteorology rely on derivative information. Talagrand [28] and Errico [9] describe the use of tangent linear (TLM) and adjoint (ADM) models for sensitivity analysis, stability (singular vector) analysis, variational data assimilation, and observation targeting. The use of second order derivative information for sensitivity analysis in the presence of observations, uncertainty analysis, and (Hessian) singular vector analysis is reviewed by Le Dimet et. al. [18].

Despite recent progress in Automatic Differentiation (AD) and first successful applications of AD-tools in the support of TLM and ADM coding [2, 8, 7, 24, 30, 31, 32], hand-coding of TLMs and ADMs is still common in dynamic meteorology. This is in contrast to oceanography, where fully automated generation of TLMs and ADMs of GCMs is becoming standard [12, 17, 23, 27, 29]. The purpose of the present study is to demonstrate the (after initial code preparations) automated generation of the TLM and ADM of a state of the art GCM by means of an AD tool.

The remainder of the present paper is organised as follows. The next section introduces the GCM, followed by a section describing the TLM and ADM generation. Sections 4 and 5 each address a particular challenge in the AD process: preserving the GCM's parallelisation capabilities and linearising around an externally provided trajectory. Section 6 discusses the TLM and ADM performance, and section 7 shows an application example. Conclusions are drawn in the final section.

2 Finite-volume General Circulation Model

The NASA finite-volume General Circulation Model (fvGCM) [20, 21, 22] is a three-dimensional Navier-Stokes solver. The GCM has been developed at NASA's Data Assimilation Office (DAO, now Global Modeling and Assimilation Office, GMAO) for quasi-operational weather forecasting. The model has various configurations. For the current study, we use two resolutions: the b55 production configuration, which runs on a regular horizontal grid of about 2 by 2.5 degree resolution (144×91 grid cells) and 55 vertical layers as well as the coarse a18 development configuration, with roughly 4 by 5 degree horizontal resolution (72×46 grid cells) and 18 vertical layers.

The time step is 30 minutes, and typical integration periods vary between 6 and 48 hours depending on applications. The state of the model comprises three-dimensional fields of 5 prognostic variables, namely two horizontal wind components, pressure difference, potential temperature, and moisture.

3 Applying TAF to fvGCM

For GMAO's retrospective Data Assimilation System (GEOS-DAS, [31]), TLM and ADM versions of fvGCM's dynamical core are needed. Both the TLM and the ADM refer to the Jacobian of the mapping of the initial state onto the final state. While the TLM evaluates the product of the Jacobian times a vector of initial state perturbations in forward mode, the ADM evaluates the product of a (transposed) final state perturbation vector with the Jacobian in reverse mode. Further applications at GMAO such as sensitivity analysis, stability (singular vector) analysis, or chemical data assimilation also require TLMs and ADMs.

Transformation of Algorithms in Fortran (TAF) [13, 14] is a source-to-source transformation AD tool for programs written in Fortran 77-95, i.e. TAF generates a TLM or ADM from the source code of a given model. As the above applications differ in their sets of dependent and independent variables, TAF generates a TLM/ADM pair for each of the applications and in addition two pairs for finite difference tests (rough and detailed). fvGCM is implemented in Fortran 90 and contains about 87000 lines of source code excluding comments. It makes use of Fortran-90 features such as *free source form*, *modules*, *derived types* and *allocatable arrays*.

Our standard approach to render a given code TAF-compliant consists of combining modifications to the model code with TAF enhancements. For instance, at two places fvGCM's control flow structure has been simplified. Generation of an efficient store/read scheme for providing required values [13] (often denoted by *trajectory*) has been triggered by 41 TAF init directives and 75 TAF store directives. The ADM can be generated with and without a checkpointing scheme [14]. To support TAF's data dependence analysis, 11 TAF loop directives have been used to mark parallel loops. In total 204 TAF flow directives have been inserted to trigger generation of specified calling sequences [14]. For instance, TAF flow directives allow one to use the Fast Fourier Transformation (FFT) and its inverse in the TLM and ADM, respectively, which is more efficient than using a generated FFT derivative code [4, 14, 28]. In some subroutines, variables were allocated and/or initialised during the first call. This introduces a data flow dependence between the first and later calls which forces TAF to generate proper but inefficient recomputations. In order to avoid these recomputations we have moved the allocations and initialisations into extra module procedures which are not differentiated.

As a result of these initial modifications, the generation procedure for the derivative code is now fully automated. This is important to allow quick updates of the derivative code to future changes in the underlying model code. After each code change, the updated TLM and ADM have to be verified. Depending on the nature and the extent of the change, additional modifications may be necessary to keep the generated derivative code correct and efficient. Unfortunately there is a bug in the SGI-compiler (version 7.4.0) on the production machine (Origin 2000) which requires switching off the compiler optimisation for two files and reducing the optimisation level (-O2 instead of -O3) for six more files in the ADM. It turns out that the wrong compiler optimisation causes an inaccuracy of only a few percent, which is acceptable for many applications.

4 Parallelisation

Regarding parallelisation, fvGCM can run on both shared and distributed memory architectures. On shared memory machines the model parallelises over vertical levels using OpenMP [25, 26] directives, and on distributed memory machines it parallelises over latitude bands using calls to the MPI library (MPI-1 [15] or MPI-2). There are even architectures that allow one to combine OpenMP and MPI, e.g. the so-called non-uniform memory access (NUMA) systems. Our production machine, an SGI Origin 2000, belongs to this class.

The challenge for AD consists in transferring these parallelisation capabilities to the TLM and ADM. This task is not to be confused with AD applications based on sequential function codes, where parallelisation is restricted to the derivative code (see, e.g., [1, 3]).

For OpenMP, the model arranges all its parallelisation by repeated use of the `parallel do` directive. Analysis of such parallel loops is discussed in [13, 16]. The loop analyses in TAF have been extended to evaluate the `parallel do` directive. For each parallelised loop of the model, i.e. each loop furnished with an OpenMP directive, TAF can automatically generate the proper parallelisable TLM and ADM versions, including their OpenMP directives. By specifying either **-omp** or **-omp2** as command line options, the user selects the OpenMP standard to which the generated code conforms. Without either command line option, the code generation ignores OpenMP directives in the model code altogether, i.e. TAF produces sequential code.

For MPI-communication, rather than including calls to the library routines directly into the main code of the GCM, there is an additional layer of routines in between. These wrapper routines are called from the main code of the model and arrange all MPI-communication internally. All wrappers plus a few utility routines form a Fortran-90 module (named `mod_comm`). As an example, File 1 shows the MPI-1 version of a wrapper routine that exchanges a three-dimensional field across boundaries of latitude bands. It does all the necessary bookkeeping for indices, and the packing of the relevant section of the field `q` using the utility routine `BufferPack3d`, which is also part of `mod_comm`.

For a subset of MPI-1, Faure and Dutto [10, 11] address handling in forward and reverse mode AD, respectively. Carle and Fagan [6] as well as Bischof and Hovland [5] address handling of MPI-1 in forward mode AD. In forward mode, TAF handles most relevant MPI calls. Among the MPI library routines used by fvGCM, the only one missing is *MPI_Allreduce* with *MPI_MAX* as reduction operation. As MPI versions of both the TLM and ADM are needed, we chose to handle MPI via a different approach (see also [27]): Adjoints of all wrappers have been hand-coded. In the TLM, most of the model wrappers can be reused; only a single TLM wrapper had to be hand-coded. As the actual MPI library calls are carried out within the wrappers, this approach is working independently of the MPI standard (MPI-1 or MPI-2). The ADM version of the wrapper in File 1 is shown in File 2.

Inclusion of the proper calling sequences for TLM and ADM versions of the wrappers into the generated TLM and ADM is triggered by TAF flow directives. Specifying TAF flow directives for a routine makes TAF analyses ignore the code of the routine and instead use the information provided by the directives. The flow directives for the wrapper in File 1 are shown in File 3. The first word, `!$taf`, is a keyword indicating a directive to TAF. The

```
      subroutine mp_send3d_ns(im, jm, jfirst, jlast, kfirst, klast, &
                               ng_s, ng_n, q, iq)
       implicit none
       integer im, jm
       integer jfirst, jlast
       integer kfirst, klast
       integer ng_s      ! southern zones to ghost
       integer ng_n      ! noruthern zones to ghost
       real q(im,jfirst-ng_s:jlast+ng_n,kfirst:klast)
       integer iq
 ! Local:
       integer i,j,k
       integer src, dest
       integer qsize
       integer recv_tag, send_tag
       ncall_s = ncall_s + 1
 ! Send to south
       if ( jfirst > 1 ) then
         src = gid - 1
         recv_tag = src
         qsize = im*ng_s*(klast-kfirst+1)
         nrecv = nrecv + 1
         tdisp = igonorth*idimsize + (ncall_s-1)*idimsize*nbuf
         call mpi_irecv(buff_r(tdisp+1), qsize, MPI_DOUBLE_PRECISION, src, &
                        recv_tag, commglobal, rqest(nrecv), ierror)
         dest = gid - 1
         qsize = im*ng_n*(klast-kfirst+1)
         tdisp = igosouth*idimsize + (ncall_s-1)*idimsize*nbuf
         call BufferPack3d(q, 1, im, jfirst-ng_s, jlast+ng_n, kfirst, klast, &
                              1, im, jfirst, jfirst+ng_n-1, kfirst, klast, &
                              buff_s(tdisp+1))
         send_tag = gid
         nsend = nsend + 1
         call mpi_isend(buff_s(tdisp+1), qsize, MPI_DOUBLE_PRECISION, dest, &
                        send_tag, commglobal, sqest(nsend), ierror)
       endif
 ! Send to north
       if ( jlast < jm ) then
 ...
       endif
       end subroutine mp_send3d_ns
```

File 1: Example of a wrapper routine for MPI-communication. To save space the kernel of the lower `if-then-endif` construct (indicated by the dots) is not displayed. It works analogously to the kernel of the upper `if-then-endif` construct.

leading “`!`” makes the Fortran compiler ignore the directive. The next words, `module mod_comm subroutine mp_send3d_ns`, indicate the module and the routine to which the flow directives refer. The first two directives indicate the input and output arguments of the subroutine. The numbers refer to the position of an argument in the argument list, i.e. arguments 1 to 10 (in fact all arguments) are input, and none is output. For the current routine the output directive may also be omitted as the empty set is the default for this type of directives. The next two directives indicate active and required arguments [13]. The next two directives indicate the names of the TLM and the ADM versions of the routine. Since the name of the TLM version corresponds to the name of the original routine, TAF recognises that the routine is linear.

```
!=========================================================================
      subroutine mp_send3d_ns_ad(im, jm, jfirst, jlast, kfirst, klast, &
                                 ng_s, ng_n, q, iq)
!=========================================================================
      implicit none
      integer im, jm
      integer jfirst, jlast
      integer kfirst, klast
      integer ng_s      ! southern zones to ghost
      integer ng_n      ! noruthern zones to ghost
      integer iq        ! Counter
      real    q(im,jfirst-ng_s:jlast+ng_n,kfirst:klast)
! Local:
      integer i,j,k
      integer src
      integer recv_tag
      ncall_r = ncall_r + 1
! Recv from south
      if ( jfirst > 1 ) then
        nread = nread + 1
        call mpi_wait(rqest(nread), Status, ierror)
        tdisp = igonorth*idimsize + (ncall_r-1)*idimsize*nbuf
        call BufferUnPack3dx(q, 1, im, jfirst-ng_s, jlast +ng_n  , kfirst, klast, &
                                1, im, jfirst     , jfirst+ng_n-1, kfirst, klast, &
                                buff_r(tdisp+1))
      endif
! Recv from north
      if ( jlast < jm ) then
...
      endif
      if (ncall_r == ncall_s) then
        call mpi_waitall(nsend, sqest, Stats, ierror)
        nrecv = 0
        nread = 0
        nsend = 0
        ncall_s = 0
        ncall_r = 0
      endif
      end subroutine mp_send3d_ns_ad
```

File 2: ADM version of the wrapper in File 1. Again the kernel of the second `if-then-endif` block not displayed (indicated by the dots), to save space.

```
!*********************************************************************
! mp_send3d_ns
!*********************************************************************
!$taf module mod_comm subroutine mp_send3d_ns   input = 1,2,3,4,5,6,7,8,9,10
!$taf module mod_comm subroutine mp_send3d_ns  output =
!$taf module mod_comm subroutine mp_send3d_ns  active = 9
!$taf module mod_comm subroutine mp_send3d_ns  depend = 1,2,3,4,5,6,7,8  ,10
!$taf module mod_comm subroutine mp_send3d_ns  adname = mod_comm::mp_send3d_ns_ad
!$taf module mod_comm subroutine mp_send3d_ns ftlname = mod_comm::mp_send3d_ns
!$taf module mod_comm subroutine mp_send3d_ns common mp_3d_ns output = 1
!$taf module mod_comm subroutine mp_send3d_ns common mp_3d_ns active = 1
```

File 3: TAF flow directives triggering generation of calls to TLM and ADM versions of File 1.

The last two directives refer to elements of the common block `mp_3d_ns`, their syntax is similar to that of the directives for the argument list.

With the flow directives the generated TLM and ADM can be linked to the hand-written wrappers. Hence, the generation procedure can be fully automated and produces four TLM/ADM versions, one for each of the combinations OpenMP on/off and MPI on/off.

5 Linearising around an External Trajectory

In dynamic meteorology it is typical to run the TLM/ADM on a coarser resolution than the forecast model. Also a set of processes called physics (e.g. parametrisations of clouds and rain, surface drag, or vertical diffusion) is usually not included in the derivative code. This reduces the computational demand and avoids potential problems arising from numerical instability. To compensate for this approximation, one linearises along an external trajectory of the state computed by the complete high resolution model. Technically this is achieved by making the (coarse grid) TLM/ADM integration periodically read in a regridded version of the external state and overwrite the internal state. However, to an AD tool, overwriting makes the final model state appear independent from the initial model state, as the data flow is interrupted. Straightforward use of AD would result in an erroneous TLM/ADM. To solve this problem, we exploit TAF's flexibility in setting up a store/read scheme for providing required variables. A combination of TAF init, store, and flow directives essentially hides the overwriting from TAF analyses and includes proper calls to the routines that provide the external trajectory. The resulting trajectory versions of the TLM and ADM are no longer proper linearisations of the coarse resolution model, unless they are run with an external trajectory provided by the model itself.

6 Performance of Generated Code

The performance of the sequential TLM and ADM versions has been tested on a Linux PC (P4 3GHz Processor, 2 GByte memory, Intel Fortran Compiler 8.0) and on a SGI Origin 2000 (Compiler version 7.4.0). On the Linux PC we could only run the coarser a18 configuration (see section 2), because of memory limitations. On the SGI we have done separate tests for OpenMP-1 (8 threads) and for MPI-1 (8 processors). We ran the ADM in both the inaccurate version with full compiler optimisation and the accurate version with reduced compiler optimisation. The integration period was six hours. We ran the configuration without check-pointing and without reading an external trajectory, i.e. both the TLM and ADM integration include a model integration. All required variables were stored in memory.

The performance numbers are listed in Table 1. It is common to quantify the CPU time of the derivative code in multiples of the CPU time of a function evaluation (model integration). For the ADM performance, it is striking that

Table 1. TLM and ADM run time in multiples of model run time.

Platform/Setup	resolution	TLM	ADM	ADM-noopt
Linux Intel 4	a18	1.5	7.0	-
SGI-OpenMP-1/8 threads	b55	1.5	10.8	20.6
SGI-MPI-1/8 threads	b55	1.5	3.9	12.6

the OpenMP version performs so much worse than the MPI version. We think that this is due to many critical sections, which have to be generated, because the OpenMP 1.1 standard does not support array reductions. As the OpenMP 2.0 standard [26] allows array reduction, TAF generated code conforming to OpenMP 2.0 does avoid critical sections. We were, unfortunately, not able to run that version of the generated code on SGI, due to a problem in the compiler's handling of OpenMP 2.0. It is also remarkable that the ADM value for MPI on SGI is much better than on Linux. We attribute this to the difference in grid resolution between the two configurations, which affects the ratio of memory accesses to computations. While both resolutions need to access the same number of arrays in memory (albeit of different sizes), the coarse a18 resolution on Linux does fewer operations. This conjecture is supported by initial ADM tests of the MPI-1 version in a18 resolution on SGI, which show a performance ratio close to the one for Linux. Finally, SGI values for the ADM with reduced optimisation are considerably slower than those with full optimisation for both OpenMP and MPI.

Fig. 1 shows the speedup for OpenMP-1, for the model itself, the TLM and the ADM for 2, 4, 6, and 8 threads. The speedup for n threads is defined by the quotient of the run times for 1 and n threads. The ideal speedup ignoring communication overhead is also indicated. While the TLM speedup is almost as good as that of the GCM, the ADM speedup lags behind. As mentioned above, this is presumably due to the critical sections, and OpenMP-2 is expected to yield a better speedup.

In the MPI case (Fig. 2) the TLM and ADM speedup is similar to that of the GCM code, with the ADM speedup being slightly better.

7 Application Example

As an example for an application of both the TLM and ADM, the leading singular vectors of a 24 hour integration have been computed. A singular vector is an Eigenvalue of A^*A, where A is the Jacobian of the function mapping the initial state onto the final state, and A^* its adjoint. Via the definition of the adjoint, the singular vector depends on the norms in state space at initial and final times. The leading singular vector is the initial time perturbation that amplifies the most (in the sense defined by the pair of norms). For the singular vector computation the automatically generated TLM and ADM have been

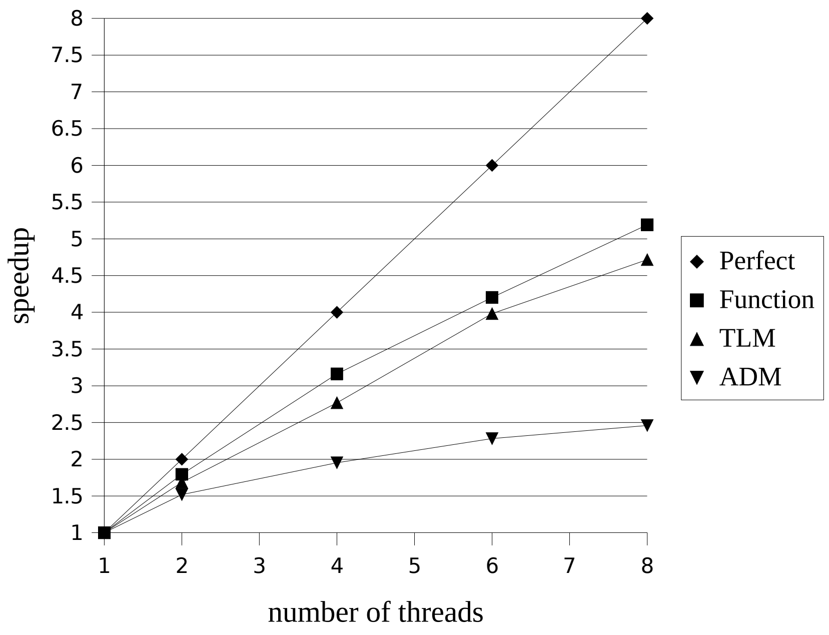


Fig. 1. Speedup for OpenMP-1.1 configuration

extended by hand-coded TLM and ADM versions of formulations for simple drag and vertical mixing (important damping processes). The coarse a18 resolution in the OpenMP version with reduced optimisation for the ADM has been used. ARPACK [19] is used to solve the Eigenvalue problem.

Fig. 3 shows the dominant singular vector for total energy norms at initial and final times. Grid points outside the target area indicated by the light black rectangle on the upper panel and in the top five vertical layers do not contribute to the norm at final time. The upper panel shows a horizontal view on the 500 hPa pressure level and the lower panel a vertical cross section averaged over the latitude band from 45 to 55 North. The shaded areas show the initial temperature component of the singular vector (temperature perturbation in K), and the thin black contours the background temperature (with 0.2 degree contour interval in the top panel and 0.1 in the bottom panel). The bold black contours show the evolved perturbation in the northward wind speed in m/s. As expected for a midlatitude perturbation, it evolves from a small scale, tilted structure at initial time to a larger scale structure at final time.

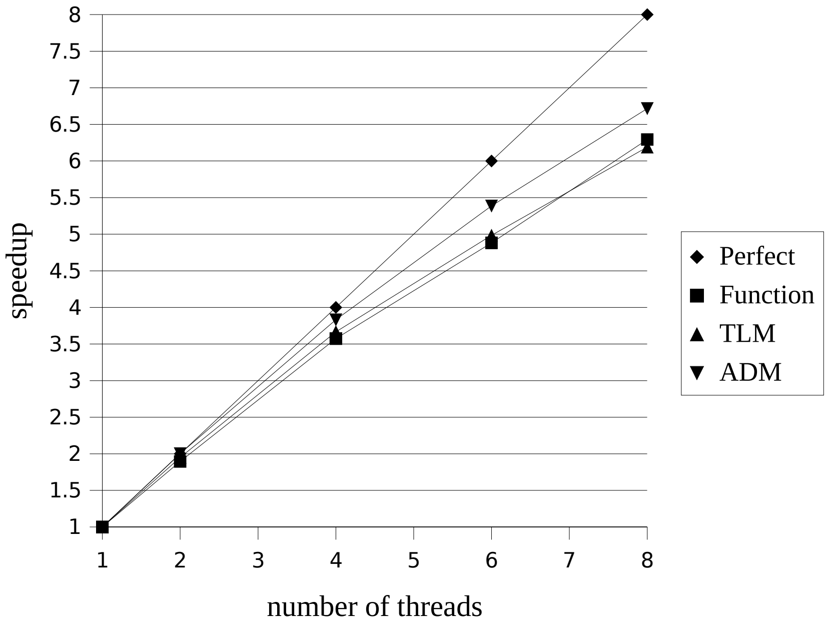


Fig. 2. Speedup for MPI-1 configuration

8 Conclusions

We have presented the generation of TLM and ADM versions of the dynamical core of fvGCM by means of TAF. After initial preparations, the generation process is fully automated. This automation is important, as it simplifies adaptation of the TLM and ADM to future changes of and extensions to the GCM code.

A TLM integration takes the run time of about 1.5 model integrations. For the ADM that number varies with the configuration of the GCM in terms of resolution and parallelisation strategy. In the most favourable case (fine resolution, MPI-1, no check-pointing, problems with the Fortran compiler ignored) a factor of 3.9 is achieved.

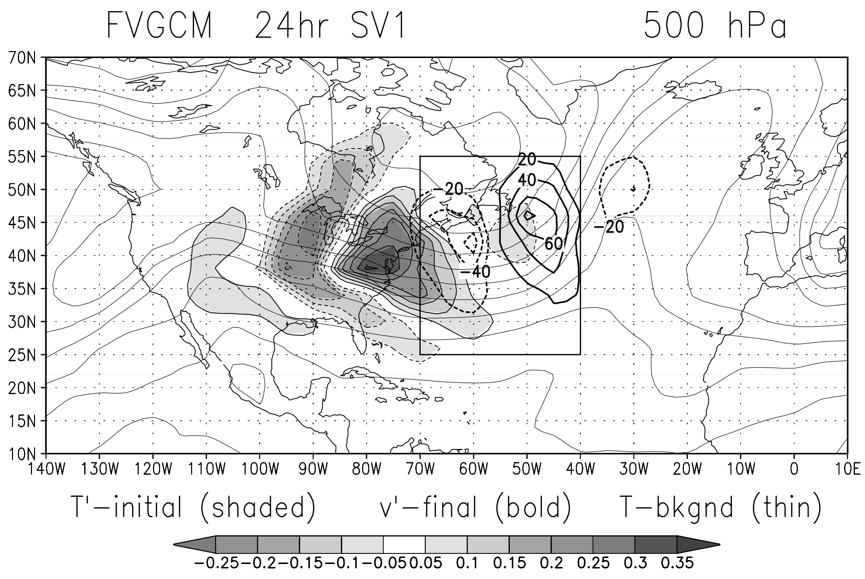
Challenges such as transferring the model's parallelisation capabilities to the TLM and ADM, or linearising around an external trajectory have been overcome. We cannot think of any fundamental obstacle that could seriously hamper automatic generation of TLMs, ADMs, and even Hessian codes of models in dynamic meteorology.

References

1. Christian H. Bischof. Issues in parallel automatic differentiation. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 100–113. SIAM, Philadelphia, Penn., 1991.
2. S. Blessing. Development and applications of an adjoint GCM. Master's thesis, University of Hamburg, Germany, 2000.
3. H. M. Bücker, B. Lang, D. an Mey, and C. H. Bischof. Bringing Together Automatic Differentiation and OpenMP. In *Proceedings of the 15th ACM International Conference on Supercomputing, Sorrento, Italy, June 17–21, 2001*, pages 246–251, New York, 2001. ACM Press.
4. H. M. Bücker, B. Lang, A. Rasch, and C. H. Bischof. Automatic Parallelism in Differentiation of Fourier Transforms. In *Proceedings of the 18th ACM Symposium on Applied Computing, Melbourne, Florida, USA, March 9–12, 2003*, pages 148–152, New York, 2003. ACM Press.
5. C. H. Bischof and P. D. Hovland. Automatic Differentiation: Parallel Computation. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, volume I, pages 102–108. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001.
6. Alan Carle and Mike Fagan. Automatically differentiating MPI-1 datatypes: The complete story. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation: From Simulation to Optimization*, Computer and Information Science, chapter 25, pages 215–222. Springer, New York, 2001.
7. I. Charpentier. Checkpointing schemes or adjoint codes: Application to the meteorological model Meso-NH. *SIAM J. Sci. Comput.*, 22:2135–2151, 2001.
8. I. Charpentier and M. Ghemires. Efficient adjoint derivatives: Application to the meteorological model Meso-NH. *Optim. Methods Software*, 13:35–63, 2000.
9. R. Errico. What is an Adjoint Model. *Bull. Am. Met. Soc.*, 78(11):2577–2591, 1997.
10. Christèle Faure and Patrick Dutto. Extension of Odyssée to the MPI library - Direct mode. Technical Report 3715, INRIA, June 1999.
11. Christèle Faure and Patrick Dutto. Extension of Odyssée to the MPI library - Reverse mode. Technical Report 3774, INRIA, October 1999.
12. E. Galanti, E. Tziperman, M. Harrison, A. Rosati, R. Giering, and Z. Sirkes. The equatorial thermocline outcropping - a seasonal control on the tropical pacific ocean-atmosphere instability. *J. Climate*, 15(19):2721–2739, 2002.
13. R. Giering and T. Kaminski. Recipes for Adjoint Code Construction. *ACM Trans. Math. Software*, 24(4):437–474, 1998.
14. R. Giering, T. Kaminski, and T. Slawig. Applying TAF to a Navier-Stokes solver that simulates an Euler flow around an airfoil. *To appear in Future Generation Computer Systems*, 2005.
15. William D. Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI – Portable Parallel Programming with the Message Passing Interface.* MIT Press, Cambridge, 1994.
16. Laurent Hascoët, Stefka Fidanova, and Christophe Held. Adjoining independent computations. In George Corliss, Christèle Faure, Andreas Griewank, Laurent

Hascoët, and Uwe Naumann, editors, *Automatic Differentiation: From Simulation to Optimization*, Computer and Information Science, chapter 35, pages 299–304. Springer, New York, 2001.

17. M. M. Junge and T.W.N. Haine. Mechanisms of North Atlantic wintertime Sea Surface Temperature Anomalies. *J. Climate*, 14:4560–4572, 2001.
18. F.-X. LeDimet, I.M. Navon, and D.N. Daescu. Second order information in data assimilation. *Mon. Wea. Rev.*, 130(3):629–648, 2002.
19. R. Lehoucq, D. Sorensen, and C. Yang. ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, 1997.
20. S.-J. Lin. A finite volume integration method for computing pressure gradient force in general vertical coordinates. *Quart. J. Roy. Meteor. Soc.*, 123:1749–1762, 1997.
21. S.-J. Lin and R. B. Rood. Multidimensional flux-form semi-Lagrangian transport scheme. *Mon. Wea. Rev.*, 124(9):2046–2070, 1996.
22. S.-J. Lin and R. B. Rood. An explicit flux-form semi-Lagrangian shallow-water model on the sphere. *Quart. J. Roy. Meteor. Soc.*, 123:2477–2498, 1997.
23. J. Marotzke, R. Giering, Q. K. Zhang, D. Stammer, C. N. Hill, and T. Lee. Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *J. Geophys. Res.*, 104:29,529 – 29,548, 1999.
24. T. Nehrkorn, G. D. Modica, M. Cemiglia, F. H. Ruggiero, J. G. Michalakes, and X. Zou. MM5 adjoint development using TAMC: Experiences with an automatic code generator. In *Proceedings of 14th Conference on Numerical Weather Prediction*, pages 481–484. American Meteorological Society, 2001.
25. OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface, Version 1.1. http://www.openmp.org, 1999.
26. OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface, Version 2.0. http://www.openmp.org, 2000.
27. P. Heimbach and C. Hill and R. Giering. An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation. *To appear in Future Generation Computer Systems*, 2005.
28. Oliver Talagrand. The use of adjoint equations in numerical modelling of the atmospheric circulation. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 169–180. SIAM, Philadelphia, Penn., 1991.
29. G.J. van Oldenborgh, G. Burgers, S. Venzke, C. Eckert, and R. Giering. Tracking down the delayed ENSO oscillator with an adjoint OGCM. *Mon. Wea. Rev.*, 127:1477–1495, 1999.
30. Y. Xiao, M. Xue, W. Martin, and J. Gao. Development of an adjoint for a complex atmospheric model, the ARPS, using TAF. In H. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Tools*, Lecture Notes in Computational Science and Engineering. Springer, 2005.
31. Y.Q. Zhu, R. Todling, J. Guo, S.E. Cohn, I.M. Navon, and Y. Yang. The geos-3 retrospective data assimilation system: The 6-hour lag case. *Mon. Wea. Rev.*, 131(9):2129–2150, 2003.
32. M. Zupanski, D. Zupanski, D.F. Parrish, E. Rogers, and G. Dimego. Four-dimensional variational data assimilation for the blizzard of 2000. *Mon. Wea. Rev.*, 130(8):1967–1988, 2002.



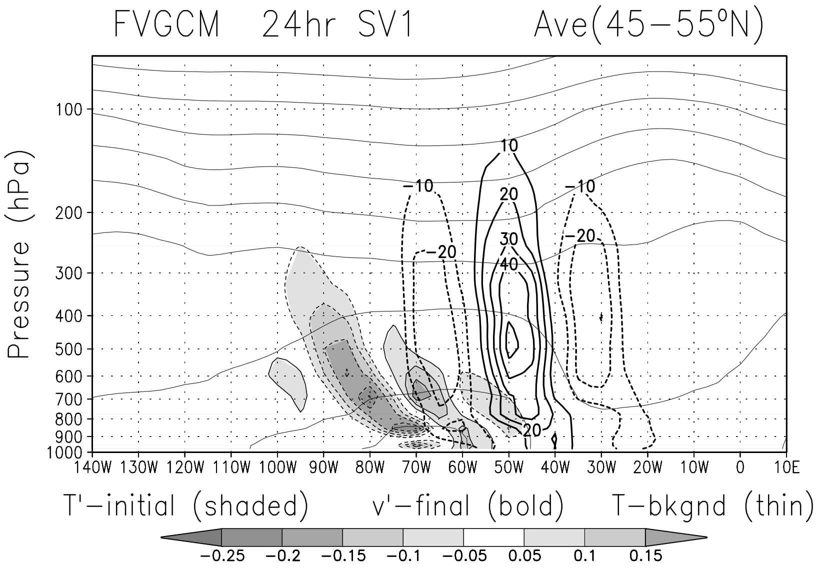


Fig. 3. Leading singular vector for 24 hour integration. See text for details.