On the Performance of Derivative Code Generated by TAMC

Ralf Giering

FastOpt, Hamburg, Germany

e-mail: Ralf.Giering@FastOpt.de

Thomas Kaminski

Max-Planck-Institut für Meteorologie, Hamburg, Germany

e-mail: kaminski@dkrz.de

submitted to Optimization Methods and Software

April 25, 2000

Abstract

The Tangent linear and Adjoint Model Compiler (TAMC) is a source-to-source translator for Fortran programs that generates code to evaluate first or second order derivatives. For first order derivatives, code operating in forward (tangent linear code) or reverse mode (adjoint code) mode can be generated. TAMC's key features for generating efficient adjoint code are briefly described.

The performance of TAMC generated adjoint and second order derivative code is compared to that of hand coded counterparts for scalar valued functions in the Minpack-2 collection. The run times for the automatically generated and hand written codes are similar. With increasing degree of optimization by the Fortan compiler, the speedup for the generated codes is larger than that for the hand written codes. TAMC has generated adjoint models for large scale applications in dynamic meteorology and oceanography. For some of those the performance is discussed: The run time of the adjoint codes is only about a factor of 3 to 6 higher than that of the respective function evaluations. This, however, includes the time for 2 or 3 function evaluations, which are required by the necessary check-pointing schemes.

Keywords: Automatic differentiation, optimization, adjoint model, adjoint code

1 Introduction

Adjoint models are increasingly being used in computational fluid dynamics (CFD), in particular in meteorology, oceanography, and climate research. Typical applications are data assimilation, model tuning, and sensitivity analysis. Both data assimilation and model tuning derive a set of control variables that achieves an optimal degree of consistency between simulated and observed quantities. This degree of consistency is quantified by a scalar valued misfit or cost function, which is defined by the (usually large and complex) numerical model of the system under consideration. If first order derivatives can be provided, powerful iterative gradient algorithms (see, e.g, 9) can be employed to minimize the cost function by variation of the control parameters.

Applying the reverse mode of automatic differentiation (AD), adjoint code evaluates this first order derivative or gradient. To analyze the uncertainties in the inferred optimal values of the control variables, second order derivatives of the scalar valued cost function are of interest. Since, usually, the number of control variables is large, evaluation of the full second order derivative, i.e. the Hessian matrix, is prohibitively expensive. However, Hessian times vector products are relatively cheap and provide a module to compute certain properties of the Hessian matrix. For example the best constrained directions are the leading eigenvectors of the Hessian matrix and can be determined iteratively by Lanczos type algorithms (22).

In practice, many of the abovementioned adjoint applications are based on models that have been previously developed and applied for simulation of the system under consideration, i.e. the designers of these models did not necessarily have adjoint applications in mind. Typically these models are written in Fortran, more precisely some Fortran dialect in between FORTRAN-77 and Fortran-90, with a recent tendency towards Fortran-90. These models typically run on super computers close to the limit of resources in terms of both memory and CPU time. Since the abovementioned applications (except for sensitivity analysis) require multiple runs of the adjoint models, it is obvious that efficient use of computer resources by the adjoint code is a necessary condition for executing the generated adjoint models.

In the eighties and early nineties, adjoints of CFD models have been hand coded. This task, however, is extremely error prone and time consuming. Furthermore, the strategies that have been used made the adjoint code inflexible to changes in the model code. As a consequence, development of adjoint models was rare and usually limited to simplified models (32; 29). This also holds for numerical weather prediction models: despite an enormous effort in hand coding of adjoints, typically, only for a fraction of the code the adjoints are available. For evaluation of second order derivatives, the situation is even worse: As a consequence of its even larger degree of complexity, for large scale applications, there exists no hand written code (4).

Since a few years, a number of AD tools are being developed, some of which are capable of generating adjoint code (Odyssee (28), TAMC (7)). Further tools operating in reverse mode are employing operator overloading capabilities of C^{++} or Fortran-90 (ADOL-C, AD01, ADOL-F, IMAS, OPTIMA90) (3) or do explicit operator overloading in Fortran-77 by replacing arithmetic operations by subroutine calls (PADRE2 (21), GRESS (16)). In contrast to hand coding, these tools allow fast and safe generation of adjoint code. Therefore adaptation of the derivative code to changes in the model code is fast, too. So far, however, two essential disadvantages of these tools have hampered their wider use: (i) the need of extensive preparation of the model code (only a subset of the language could be handled, and information about the structure of the code had to be provided), and (ii) the low computational efficiency of the generated code, which is most severe for operator overloading. Meanwhile, the development of the Tangent linear and Adjoint Model Compiler (TAMC, (7)) has reached a state at which both of these disadvantages are overcome: For a number of large and complex CFD models, after minor preparation of their code, TAMC was able to generate efficient derivative computing code. This paper quantifies and discusses the performance of the adjoint codes for some of these models, and the second derivative computing code for one of the models.

Although, in theory, these derivative computing codes also could have been hand written, in practice, without AD none of these applications would have been performed. This means in particular that there exist no hand coded counterparts to compare the automatically generated code to in terms of efficiency. To get at least a flavor of how such a comparison would look like, we employed the Minpack-2 test problem collection (2). For a number of small to medium size problems this collection contains hand written code to evaluate a scalar valued function, its gradient, and the product of its Hessian times a vector.

The remainder of the paper is organized as follows: Section 2 gives a brief description of TAMC. Section 3 gives a comparison of TAMC generated derivative code to hand written derivative code for some functions of the Minpack-2 collection. The performance of complex CFD codes will be quantified in section 4. Section 5 gives a summary and conclusions.

2 TAMC

The aim of this section is to give a brief description of TAMC, with a focus on those features that are essential for generating efficient derivative code. A more elaborate introduction to TAMC is given by (8), users should consult also (7).

TAMC is a source-to-source translator for Fortran programs to generate derivative computing code operating in forward or reverse mode. The internal algorithms are based on a few principles suggested e.g. by Talagrand (31). These principles can be derived from the chain rule of differentiation. TAMC applies a number of analyses and code normalizations similar to those applied by optimizing compilers (constant propagation, index variable substitution, data dependence analysis). In addition, given the top-level routine to be differentiated and the independent and dependent variables, by applying a forward/reverse data flow analysis TAMC detects all variables that depend on the independent variables and influence the dependent variables (active variables). This is in contrast to operator overloading based tools, where the user has to determine active variables and to declare them to be of a specific data type. TAMC can handle all but very few relevant FORTRAN-77 and Fortran-90 statements.

A mayor challenge of adjoint code is providing intermediate results that are required, e.g. to evaluate derivatives of non linear operations. Efficient adjoint code uses a combination of recalculating and restoring from a tape written previously; both strategies can be applied by TAMC, the latter is invoked by inserting directives in the code to be differentiated. The tapes are realized in core memory or on disks. Note that the trade off between recomputation and storing has to be made by the user. In this respect generation of *most efficient* adjoint code works still semi-automatically. For generation of recomputations a reverse data flow analysis is applied, and, as far as possible, only statements being absolutely necessary are included. Concerning this key issue for generation of efficient derivative code, TAMC is unique among the AD tools.

For the large applications discussed in section 4 recomputing all values would be prohibitively time consuming. On the other hand, the tape space needed to store all required values exceeds the available resources by orders of magnitude. The checkpointing technique suggested by (10) solves this problem. It allows to use the available resources for storing intermediate results more efficiently, at the cost of additional model runs and is indispensable for large applications. The idea is to partition the entire evaluation of the function by defining a number of check-points. During an initial evaluation of the function (highest level of check-pointing), at any check-point snapshots of the state of the model are taken, i.e. all values that are needed to redo the evaluation for the following portion are stored. Next, working through the sequence of partitions in reverse order, for the current partition, first the function code and then the derivative code are executed. The evaluation of the partition of the function code (lowest level of check-pointing) serves to provide all required values for the corresponding adjoint code. Note that for a time integration model, the space per time step needed to store all those values often exceeds the space used to store a snapshot. For some applications even a three level check-pointing scheme was necessary to actually allow executing of the derivative code. For some of the derivative codes described in section 4, the check-pointing scheme was generated semi automatically by TAMC. For functions stepping through a main loop, e.g. integrations in time, this main loop has to be splitted into nested loops, one loop for each level of check-pointing. In addition to the directives for storing intermediate results of the inner loop, similar directives have to be inserted in the outer loops to generate code for storing the snapshots.

The strategy for setting the check-points depends on the platform used. Usually access to core memory is faster than to disk. In this case the check-points should be set in such a way that the innermost loop (the lowest level) uses all available core memory, because this tape is accessed most frequently. This advantage of the two tape strategy is one of the reasons why TAMC does not apply the sophisticated original algorithm of (10), which, although needing only resources in proportion to the logarithm of the number of required intermediated results, uses only a single tape. The other reason is that the algorithm implemented in TAMC is much simpler. For an n level check-pointing, TAMC can achieve a growth of the resources in proportion to the $n-$th root of the number of time steps. In cases with equally fast tapes the TAMC check-pointing algorithm is most memory efficient for tapes of equal size.

Reapplying TAMC to the first order derivative code with appropiate options, it generates code to evaluate second order derivatives. This works for any combination of forward and reverse modes for the two successive applications of TAMC. The most efficient variant operates in the so-called forward over reverse mode (FOR), i.e. the first order derivative is computed in reverse mode and the second order derivative in forward mode. The constructed code evaluates Hessian times vector products, Hessian times matrix products, or the full Hessian. Other tools use the forward over forward mode (FOF) or Taylor series expansion (TSE) (1). For scalar valued functions FOR is much faster, and the run time ratio for derivative to function code is independent of the number of control variables, while the cost of FOR and TSE increases with this number. In theory (11), a run time ratio below 10 should be attainable.

3 Comparison to Hand Written Derivative Code

For a number of small to medium size problems the Minpack-2 collection (2) contains hand written code to evaluate a scalar valued function, its gradient, and the product

name	lines	short description
ept	51	elastic-plastic torsion
ssc	54	steady state combustion
pjb	61	pressure distribution in a journal bearing
gl1	70	Ginzburg-Landau (1-dimensional) superconductivity
msa	90	minimal surface area
gl2	111	Ginzburg-Landau (2-dimensional) superconductivity

Table 1: Names of Minpack-2 problems and their number of code lines

of its Hessian times a vector. The number of independent variables can be chosen arbitrarily. To compare the performance of TAMC generated derivative code with, we selected six of these problems, which are representative of small to medium scale optimization problems arising from applications in superconductivity, optimal design, combustion, and lubrication. Table 1 gives the list of problems and their number of Fortran code lines.

The code for function evaluation has been differentiated by TAMC to generate code for evaluation of the gradient (adjoint code). The comparison has been carried out on two machines, a Sun Ultra-1 and a Cray C90. To allow a fair comparison on the Cray C90, the performance of the hand written code has been improved by inserting vectorization directives and moving conditional statements out of the inner most loop. The codes have been compiled by the vendors Fortran compiler with the precision and compiler options given in Table 2.

The results for evaluation of the gradient codes are depicted in Fig. 1 for Sun

platform	precision	Fortran command line
Sun Ultra-1	double precision	f90 -O2
Cray C90	double precision	f90 -O inline3,scalar3,vector3,task0

Table 2: Precision and compiler options used on platforms.

Ultra-1 and in Fig. 2 for Cray C90. For every test problem the relative run time, i.e. the run time of the gradient code compared to the run time of the function code, has been computed for different numbers of independent variables. On Sun Ultra-1 the hand written code is in four cases slower than the TAMC generated code (GL2,SSC,GL1,EPT). However, a remarkable difference can only be seen for the GL2 problem, while in all other cases differences are small. For GL2, a nested loop in the function computing code is split into three loops in the hand written gradient code: one for interior points of the domain and two for boundary points. This is common practice in hand written adjoint codes. In contrary, TAMC does not split the loop. Instead, interior and boundary points are handled simultaneously as is implied by strict application of the rules TAMC is based on (8). In all cases, the changes of the relative run time with the dimension of the problems (the number of independent variables) are very small. On a Sun Ultra-1 performance is degraded by cache misses. Their number depends mainly on the memory needed for all variables in a loop compared to the cache size. For non-linear operators, this ratio is different for function and gradient code. This explains the spikes at certain problem sizes.

The differences in relative run time are also small on a Cray C90, except again for the GL2 problem. Here, in most cases, the relative run time increases slightly with

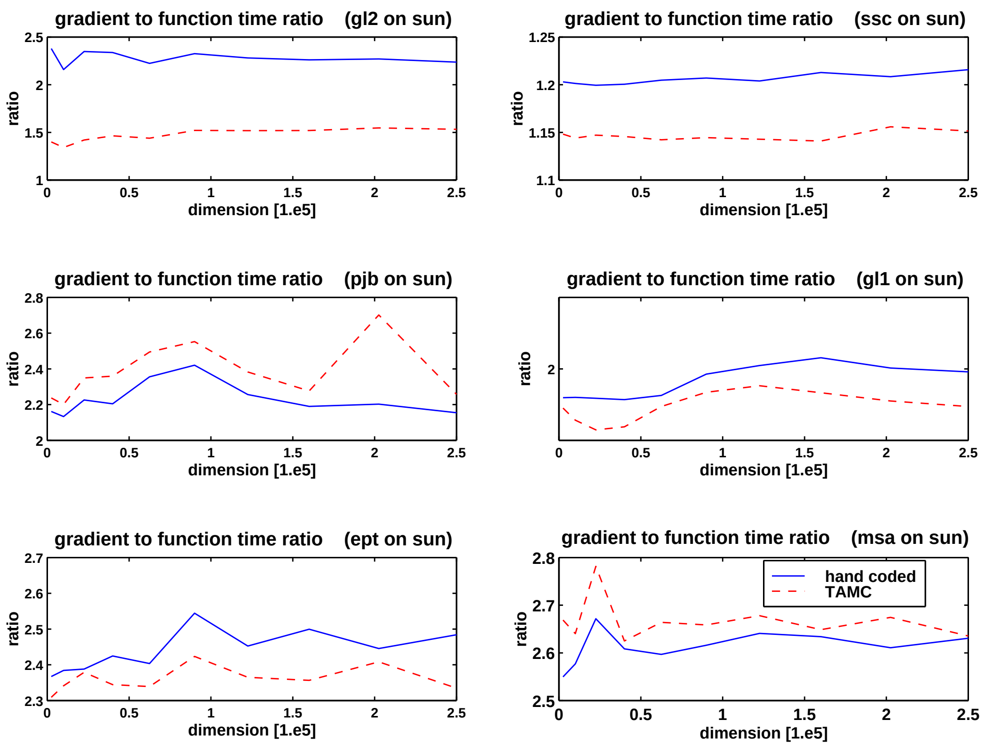

Figure 1: Relative run time of gradient code on Sun Ultra-1 (x-axis is the number of control variables).

the problem size.

Some recomputations in the adjoint code are independent of the problem size. If they constitute a mayor fraction of all computations, as is the case for small problem sizes, the ratio is almost one. For large sizes the run time of the adjoint code is dominated by updating adjoint variables. Thus, the ratio depends on the complexity of the non-linear operations in the corresponding function code. On vector machines like the Cray C90 run time depends mainly on the efficient use of vector pipes. For these test problems the effective vector length increases with the number of independent

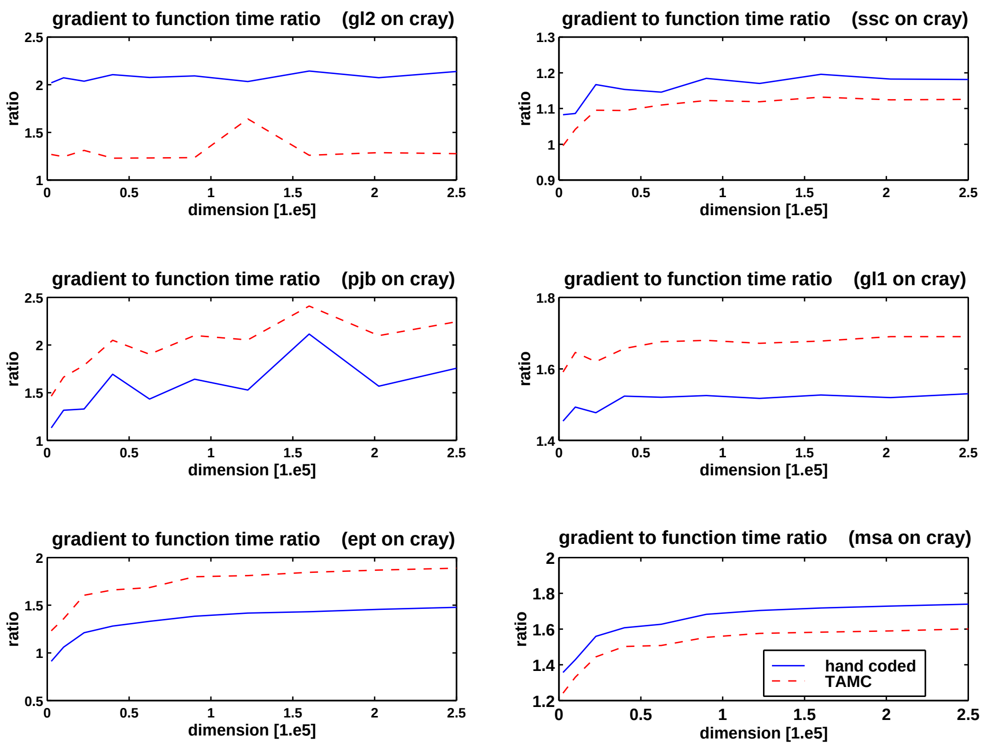

Figure 2: Relative run time of gradient code on Cray C90 (x-axis is the number of control variables).

variables. Thus, on a Cray C90, in contrary to the Sun Ultra-1, the transition can be seen at higher problem sizes.

The Hessian times vector code has only been compared on the Sun Ultra-1. The results depicted in Fig. 3 show the relative run time of the Hessian times vector code compared to the run time of the original function code. Only in one case (GL2) is the TAMC generated code faster than the hand written code. As for the gradient code, the hand written version of the Hessian times vector code for the GL2 problem splits a nested loop into three loops. But the run time penalty for this splitting is

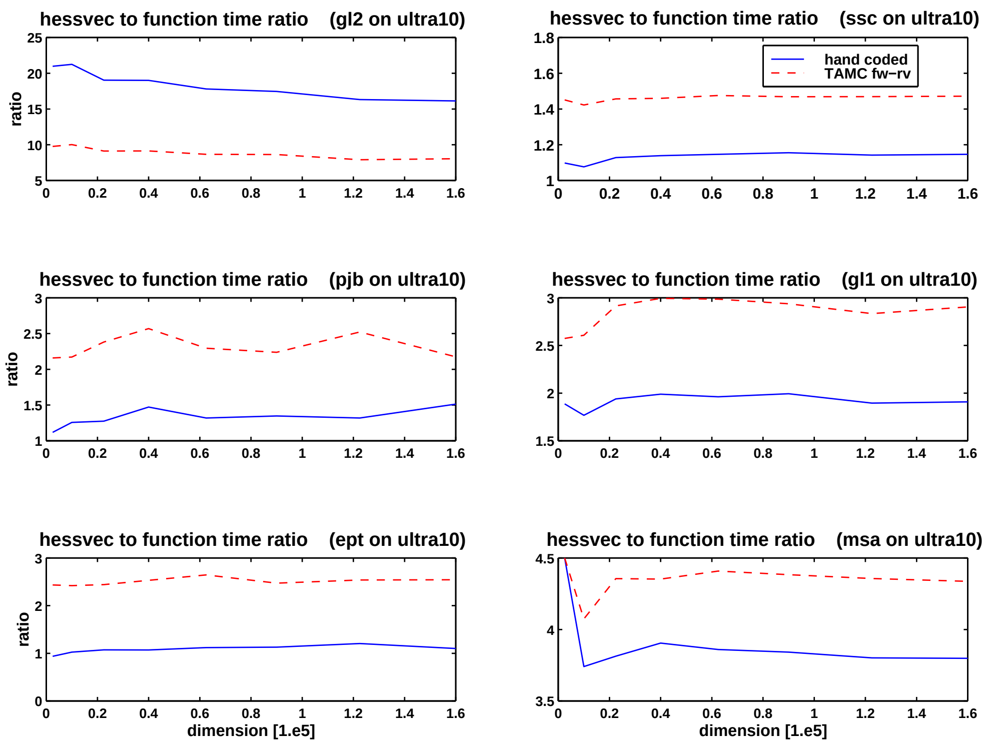

Figure 3: Relative run time of Hessian times vector code on Sun Ultra-1.

much more pronounced: the TAMC generated code is about a factor 2 faster! For the remaining problems, the TAMC generated code is slower, because TAMC generates some initializations of adjoint variables to zero that could be omitted by combining them with subsequent assignments to the same variable. Although humans can easily detect these cases, automatization can become arbitrarily complex, because it might involve comparison of array subscript expressions.

4 Performance of large CFD Codes

TAMC is being successfully applied to generate derivative computing codes for an increasing number of large and complex CFD models used for various applications. We select five of the most complex of these models, for which the derivative computing code was the essential tool for several (published) applications in their respective fields. Two of the models (MIT and HOPE) simulate the oceanic circulation, one the coupled ocean atmosphere system (HCM), one the atmospheric transport of tracers (TM2), and one the dynamics of ocean waves (WAM). For each of these models, in this section, we give a brief description (together with references for more details), summarize the respective applications, and discuss the performance of the respective derivative computing codes.

The MIT GCM solves the incompressible Navier-Stokes equation on an Arakawa C-grid, with optional hydrostatic approximation. The model has been applied to a large range of scales of ocean dynamics ranging from studies of convective chimneys to global ocean circulation estimation (25; 24) and has been developed specifically for use on modern parallel computing platforms. Adjoints of the model have been generated for data assimilation (30), and sensitivity studies (23). Using the forward over reverse mode, code for evaluation of Hessian times matrix products has been generated and is used to derive error estimates of integrated properties of the optimized ocean circulation.

The Hybrid Coupled Model (HCM) consists of an ocean model, which solves the Navier-Stokes equation for the equatorial pacific, and a simple diagnostic atmosphere model. It is applied to analyze and forecast (5) the El Niño/Southern Oscillation

(ENSO) phenomenon (27). Christian Eckert has generated its adjoint and tangent liner models to iteratively (22) determine the model's most unstable modes, i.e. the leading singular values of the product of the model's Jacobian matrix by its adjoint (5; 6).

The Hamburg Ocean Primitive Equation model (HOPE) (33) solves the Navier-Stokes equation on an Arakawa E-grid. It is run for a limited domain or globally, also coupled to atmospheric models. For a sensitivity study investigating the origin of an ENSO event, Geert Jan van Oldenborgh and Gerrit Burgers generated the adjoint (26) of HOPE on a tropical Pacific domain coupled to a simple diagnostic atmosphere model. They used a combination of TAMC and its predecessor AMC. At two points, which involved inversion of tridiagonal matrices, improved the performance by replacing the generated adjoint code by hand written adjoint code.

TM2 is a three-dimensional atmospheric transport model which solves the continuity equation for an arbitrary number of passive tracers on an Eulerian grid spanning the entire globe (13). It is driven by stored meteorological fields derived from analyses of a weather forecast model. For sensitivity studies (18; 19) and data assimilation (20; 17) Jacobian matrices have been computed by the model's adjoint.

WAM is a so called third generation model for ocean wave prediction (see, e.g., 12). It solves the two-dimensional energy balance equation for ocean wave spectra, which describes the generation, interaction, propagation, and decay of ocean waves. Using AMC the adjoint has been generated by (14; 15). Preparation of the code comprised replacing a number of constructs that AMC could not yet handle and also declaration of passive variables, i.e. those for which no derivative code is needed. Recall that the

Model	Platform	$\frac{cpu(\nabla f)}{cpu(f)}$	$\frac{cpu(f, \nabla f)}{cpu(f)}$	check-pointing
MIT CM5	CM5	2.5	4.5	2-level
MIT C90	Cray-C90	2.4	4.4	2-level
MIT T90	Cray-T90	2.1	5.1	3-level
HOPE	Cray-90	2 - 3	5 - 6	3-level
TM2	Cray-90	1.4	3.4	2-level
HCM	Cray-90	1.7	3.7	2-level
WAM	Dec-ALPHA	1.7	3.7	2-level

Table 3: CPU time used by the adjoint models.

latter is obsolete with TAMC, because of its capability to perform a flow dependence analysis, and further that TAMC can handle more constructs than AMC. The adjoint has been applied to optimize physical parameters in the wave model (15).

The numbers quantifying the performance of the adjoints of these five models are summarized in Tables 3 and 4. The first two columns of Table 3 name the model and the platform it runs on, for the MIT model numbers for three different platforms are given. The next column quantifies the ratio of the CPU times for the evaluation of the gradient to the evaluation of the function. This ratio is in between 0.7 and 3. Depending on the level of check-pointing (last column), 2-3 additional function evaluations are needed to run these large codes for typical applications. This additional burden is taken into account in the ratios in the fourth column. Even MIT+ and HOPE, the two models with the three level check-pointing still achieve a ratio below 6.

Model	T	Δt	steps	n1	tape1	n2	tape2	n3	tape3
MIT	1a	1h	8640	20×12	640 c	36	2300 c		
MIT	1a	1h	8640	4×12	128 c	180	11000 d		
MIT+	6a	1h	51840	6	192 c	120	8000 d	72	5000
HOPE	2a	2h	8640	12	250 d	30	240 d	24	192
TM2	1a	4h	2160	40	48 d	54	60 d		
HCM	0.5a	2.25h	1920	32	58.9 d	60	58.2 d		
WAM	10m	3d	432	18	270 d	24	360 d		

Table 4: Resources needed for storing intermediate results with details of check-pointing. Columns: name of model, length of integration period, time step, number of time steps, overall number of check-points, number of check-points on lowest level, size and kind of tape (MByte core memory (c) or disk (d)), same for level 2 and, where implemented, level 3.

The details of check-pointing (see section 2) and the resources needed are summarized in Table 4. Recall that all five models perform an integration in time. The length of the integration period, the size of one time step and the number of time steps are given in columns 2 to 4. In each of the models, the time varying values of a number of different variables are required at every time step. This are mostly active variables, except in the case of TM2, which spends an important fraction of its CPU time to compute passive variables. Storing these speeds up the adjoint code. The following column gives the number of time steps in the inner loop of check-pointing as well as the tape resources needed. The letters “d” and “c” indicate whether the tapes were realized on disk or in core memory. Note that some of the models (MIT and for some variables HOPE) do not provide the current values at every time step but instead use the same value for 12 time steps. This shortcoming considerably reduces tape resources but results in an inaccurate derivative. However, since the physical system simulated is changing slowly with time the departure from the exact values is small. By inserting appropriate directives, TAMC generates such inaccurate but more efficient derivative code. In the following columns, for the higher levels of check-pointing, the number of check-points is given as well as the tape resources needed. The MPI and HOPE models arrange the check-points to exploit the available core memory for the innermost loop. For WAM, HCM, and TM2 access to both tapes is about equally fast, and, thus, check-points are arranged to yield tapes of about equal size.

The Hessian times vector code of the MPI model was run on a Cray-T90 and the check-pointing strategy of the MIT_+ model was used. The ratio of this derivative code to the function code is 11. Compared to the adjoint code for the MIT_+ model, the sizes of both tapes roughly double.

5 Summary, Conclusions, and Perspectives

For a number of functions in the Minpack-2 collection, we have compared the performance of TAMC generated adjoint code and Hessian times vector code to that of their hand written counterparts. In summary, the efficiency of generated adjoint code and Hessian times vector code is comparable to that of the hand written code. In detail, the results depend on particular features of both the computer and the compiler that are used as well as on details of the implementation of both the particular function to be differentiated and the hand written code. With increasing degree of optimization by the Fortan compiler, the speedup for the generated codes is larger than that of the hand written codes. For five large and complex CFD codes, we have discussed the performance of their TAMC generated adjoint codes. Depending on the level of check-pointing, the run time of the adjoint code is in between a factor of 3-6 of that of the model, where the pure derivative code (without the additional model evaluations) is in between a factor of 1-3 of that of the model. The TAMC generated code is efficient, because it allows the user to choose an optimal compromise between recomputation and storing of required intermediate results. Also, for more efficient use of the tape resources TAMC generates check-pointing schemes, according to the users definition of the check-points. One of the challenges for future developments is to automatically choose an optimal strategy for recomputation and storing, which also includes an optimal check-pointing scheme.

Acknowledgments

The authors thank Christian Eckert, Hans Hersbach, and Geert Jan van Oldenborgh for providing the performance details of their derivative codes. Thomas Kaminski was supported by the Bundesministerium für Bildung und Forschung (BMBF) under contract number 01LA9898/9.

References

[1] Jason Abate, Christian Bischof, Alan Carle, and Lucas Roh, *Algorithms and design for a second-order automatic differentiation module*, Int. Symposium on Symbolic and Algebraic Computing (ISSAC), Association of Computing Machinery, New York, 1997, pp. 149–155.

[2] Brett M. Averick, Richard G. Carter, Jorge J. More, and Guo-Linad Xue, *The Minpack-2 Test Problem Collection*, Preprint MCS-P153-0692, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[3] Christian Bischof, *A collection of automatic differentiation tools*, URL=`http://www.mcs.anl.gov/Projects/autodiff/AD_Tools/index.html`.

[4] Christian H. Bischof, George F. Corliss, Larry Green, Andreas Griewank, Ken Haigler, and Perry Newman, *Automatic differentiation of advanced CFD codes for multidisciplinary design*, Journal on Computing Systems in Engineering **3** (1992), 625–638.

[5] Christian Eckert, *On predictability limits of enso - a study performed with a simplified model of the tropical pacific ocean-atmosphere system*, Ph.D. thesis, Max-Planck-Institut für Meteorologie, Hamburg, Germany, 1998.

[6] Christian Eckert, Ralf Giering, and Mojib Latif, *Optimal perturbations of a hybrid coupled model of el niño*, 2000, submiited to Quarterly Journal of the Royal Meteorological Society.

[7] Ralf Giering, *Tangent linear and Adjoint Model Compiler, users manual*, 1997, unpublished, available at `http://puddle.mit.edu/~ralf/tamc`.

[8] Ralf Giering and Thomas Kaminski, *Recipes for Adjoint Code Construction*, ACM Trans. On Math. Software **24** (1998), no. 4, 437–474.

[9] P. E. Gill, W. Murray, and Margret H. Wright, *Practical optimization*, Academic Press, New York, 1981.

[10] Andreas Griewank, *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, Optimization Methods and Software **1** (1992), 35–54.

[11] ______, *Some bounds on the complexity of gradients, Jacobians, and Hessians*, Complexity in Nonlinear Optimization (Panos M. Pardalos, ed.), World Scientific Publishers, 1993, pp. 128–161.

[12] The WAMDI group. S. Hasselmann, K. Hasselmann, E. Bauer, P.A.E.M. Janssen, G.J. Komen, L. Bertotti, P. Lionello, A. Guillaume, V.C. Cardone, J.A. Greenwood, M. Reistad, L. Zambresky, and J.A. Ewing, *The WAM model - a third generation ocean wave prediction model*, Journal of Physical Oceanography **18** (1988), 1775 – 1810.

[13] Martin Heimann, *The global atmospheric tracer model TM2*, Tech. Rep. 10, Max-Planck-Institut für Meteorologie, Hamburg, Germany, 1995.

[14] Hans Hersbach, *The adjoint of the WAM model*, Scientific Report WR 97-01, Royal Netherlands Meteorological Institute, 1997.

[15] ———, *Application of the adjoint of the WAM model to inverse wave modeling*, Journal of Geophysical Research **103** (1998), no. C5, 10,469–10,487.

[16] Jim E. Horwedel, *GRESS: A preprocessor for sensitivity studies on Fortran programs*, Automatic Differentiation of Algorithms: Theory, Implementation, and Application (Andreas Griewank and George F. Corliss, eds.), SIAM, Philadelphia, Penn., 1991, pp. 243–250.

[17] S. Houweling, T. Kaminski, Frank Dentener, Jos Lelieveld, and M. Heimann, *Inverse modelling of methane sources and sinks using the adjoint of a global transport model*, J. Geophys. Res. **104** (1999), no. D21, 26,137–26,160.

[18] T. Kaminski, R. Giering, and M. Heimann, *Sensitivity of the seasonal cycle of CO_2 at remote monitoring stations with respect to seasonal surface exchange fluxes determined with the adjoint of an atmospheric transport model*, Physics and Chemistry of the Earth **21** (1996), no. 5–6, 457–462.

[19] T. Kaminski, M. Heimann, and R. Giering, *A coarse grid three-dimensional global inverse model of the atmospheric transport, 1, Adjoint model and Jacobian matrix*, Journal of Geophysical Research **104** (1999), no. D15, 18,535–18,553.

[20] ———, *A coarse grid three dimensional global inverse model of the atmospheric transport, 2, inversion of the transport of CO_2 in the 1980s*, Journal of Geophysical Research **D15** (1999), no. 104, 18,555–18,581.

[21] Koichi Kubota, *PADRE2 - Fortran precompiler for automatic differentiation and estimates of rounding error*, Computational Differentiation: Techniques, Appli-

cations, and Tools (Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank, eds.), SIAM, Philadelphia, Penn., 1996, pp. 367–374.

[22] C. Lanczos, *Applied analysis*, Prentice Hall (1957).

[23] Jochem Marotzke, Ralf Giering, K. Q. Zhang, Detlef Stammer, Chris Hill, and Tong Lee, *Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity*, Journal of Geophysical Research **104** (1999), no. 29, 529–547.

[24] John Marshall, Alistair Adcroft, Chris Hill, Lev Perelman, and Curt Heisey, *A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers*, Journal of Geophysical Research **102** (1997), no. C3, 5753–5766.

[25] John Marshall, Chris Hill, Lev Perelman, and Alistair Adcroft, *Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling*, Journal of Geophysical Research **102** (1997), no. C3, 5733–5752.

[26] Geert Jan van Oldenborgh, Gerrit Burgers, Stephan Venzke, Christian Eckert, and Ralf Giering, *Tracking down the delayed ENSO oscillator with an adjoint OGCM*, Monthly Weather Review **127** (1999), 1477–1495.

[27] S. G. H. Philander, *El Nino, La Nina, and the Southern Oszillation*, Academic, San Diego, 1990.

[28] N. Rostaing, S. Dalmas, and A. Galligo, *Automatic differentiation in Odyssée*, Tellus (1993), 558–568.

[29] Jens Schröter, *Driving of non-linear time dependent ocean models by observations of transient tracer - a problem of constrained optimization.*, Ocean Circulation Models: Combining Data and Dynamics (D.L.T. Anderson and J. Willebrand, eds.), Kluwer Academic Publishers, 1989, pp. 257–285.

[30] Detlef Stammer, Carl Wunsch, Ralf Giering, Qian Zhang, Jochem Marotzke, John Marshall, and Chris Hill, *The Global Ocean Circulation estimated from TOPEX/POSEIDON Altimetry and a General Circulation Model*, Tech. Report 49, Center for Global Change Science, Massachusetts Institute of Technology, 1997.

[31] Oliver Talagrand, *The use of adjoint equations in numerical modelling of the atmospheric circulation*, Automatic Differentiation of Algorithms: Theory, Implementation, and Application (Andreas Griewank and George F. Corliss, eds.), SIAM, Philadelphia, Penn., 1991, pp. 169–180.

[32] Eli Tziperman and W. C. Thacker, *An optimal control/adjoint equation approach to studying the ocean general circulation*, Journal of Physical Oceanography **19** (1989), 1471–1485.

[33] J.-O. Wolff, Ernst Maier-Reimer, and S. Legutke, *The Hamburg Ocean Primitive Equation model HOPE*, Technical Report 13, Max-Planck-Institut für Meteorologie, 1997.