Using TAMC to generate efficient adjoint code: Comparison of automatically generated code for evaluation of first and second order derivatives to hand written code from the Minpack-2 collection

Ralf Giering
Department of Earth, Atmospheric and Planetary Sciences
Massachusetts Institute of Technology
e-mail: ralf@sea.mit.edu

Thomas Kaminski
Max-Planck-Institut für Meteorologie
e-mail: kaminski@dkrz.de

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. Thereby the degree of consistency is quantified by a scalar valued misfit or cost function, which is defined trough the (usually large and complex) numerical model of the system under consideration. The cost function can be minimized most efficiently by use of powerful iterative gradient algorithms [9], if first order derivatives can be provided. Applying the reverse mode of automatic differentiation (AD) adjoint code evaluates this first order derivative or gradient (see introductory section). To analyze the uncertainties in the inferred optimal values of the control variables, second order derivates 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 vector products are relatively cheap and provide a module to evaluate 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.

In practise, these 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.

During 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 [19, 16]. The adjoint of the atmospheric model applied for (4d-var) data assimilation at the ECMWF constitutes an exception: it has been constructed and is maintained by hand. However construction of the adjoint seems to have taken almost a decade and has started before AD tools were well enough developed to tackle this challenge. Code for evaluation of second order derivatives, as a consequence of its even larger degree of complexity, has not been hand written for large scale applications [4].

Recently a number of AD tools are being developed that are capable of generating adjoint code (Odyssee [15], GRESS [12], TAMC [7], see also other contributions to this document). Other tools operating in reverse mode are employing operator overloading capabilities of C^{++} or Fortran-90 (ADOL-C, AD01, ADOL-F, IMAS, OPTIMA90) [3].

TAMC (Tangent linear and Adjoint Model Compiler, [7]) 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 [18]. These principles can be derived from the chain rule of differentiation [8]. 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 statements and an increasing number of Fortran 90 extensions, check the latest manual version on the TAMC home page [6] for the current state of development.

Recently, TAMC has been successfully applied to generate the adjoint codes of an increasing number of large and complex CFD codes [13, 17, 14, 5]. A mayor challenge of adjoint code is providing intermediate results 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. For generation of recalculations a reverse data flow analysis is applied, and, as far as possible, only statements being absolutely necessary are inserted into the adjoint code. Concerning this key issue for generation of efficient derivative code, TAMC is unique among the AD tools. For the abovementioned applications checkpointing schemes have been implemented semi automatically by TAMC. The checkpointing technique allows to use the available resources for storing intermediate results more efficiently at the cost of an additional model run and is indispensable for these large applications [10]. For some applications even a multi level checkpointing is necessary. Depending on the level of checkpointing, the run time of the adjoint code is in between a factor of 3-6 of that of the model. Thereby the pure derivative code (without the additional model evaluations) is in between a factor of 1-3 of that of the model. See the TAMC home page [6] for more details on the adjoints of these models.

TAMC generates code to compute second order derivates operating in the so-

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

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 computes Hessian times vector products or the full Hessian. Alternative approaches use the forward over forward mode (FOF) or Taylor series expansion (TSE) [1]. For scalar valued functions FOR is much faster, and the relative run time is independent of the number of control variables, while the cost of FOR and TSE increases with this number. In theory, a relative run time below 10 should be attainable [11].

Although for the abovementioned applications, in theory, adjoint models also could have been hand coded, probably, in practise, without AD none of these applications would have been possible. This means in particular that there exist no hand coded counterparts to compare the automatically generated adjoint code to in terms of efficiency. Hence, for this purpose we employed the Minpack-2 test problem collection [2]. For each problem the collection contains hand written code to compute a scalar valued function, its gradient, and the product of its Hessian times a vector. The number of independent variables can be chosen arbitrarily.

We selected six problems that 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.

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.

The results for evaluation of the gradient codes are depicted in Fig. 1 for Sun 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 calculated for different numbers of independent variables. On Sun

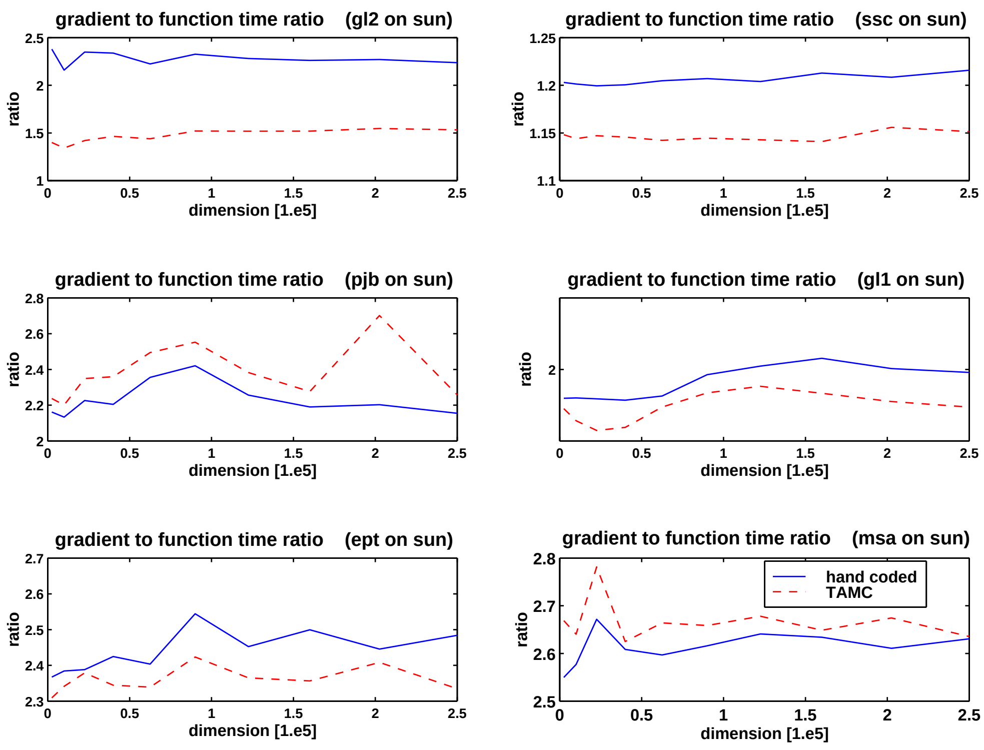

Figure 1: Relative run time of gradient code on Sun Ultra-1 (x-axis is the number of control variables).

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, in all other cases differences are small. A nested loop in the function computing code of GL2 is split into three loops in the hand written gradient code: one for interior points of the domain and two for boundary points. This has been 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 compromised 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 the problem size.

Some recalculations in the adjoint code are independent of the problem size. If they, for small sizes, constitute a mayor part of the whole calculations 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

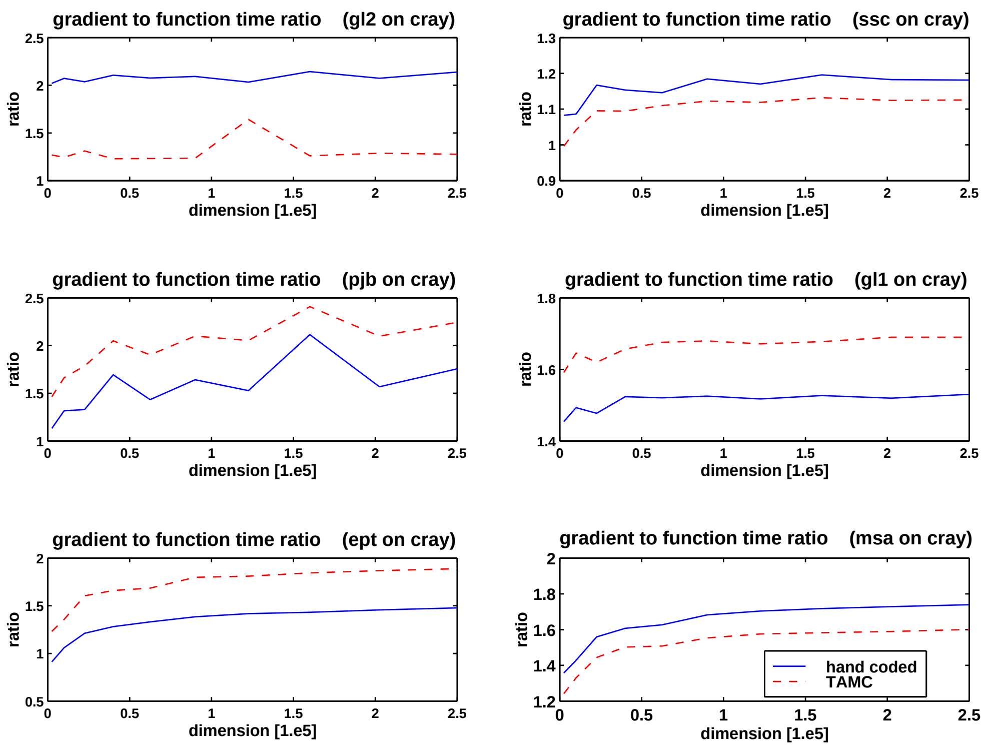

Figure 2: Relative run time of gradient code on Cray C90 (x-axis is the number of control variables).

test problems the effective vector length increases with the number of independent variables. Thus, on a Cray C90, in contrary to the Sun Ultra-1, the abovementioned transition to dominance of updating adjoint variables is 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 much more pronounced: the TAMC generated code is about a factor 2 faster! For the other 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.

In summary, the efficiency of TAMC generated adjoint code and Hessian times vector code is comparable to that of their hand written counterparts. In detail, the results depend on particular features of the computer and on the compiler that are used and also on details of the implementation of both the particular function to be differentiated and the hand written derivative code. TAMC is available through its home page [6] or by electronic mail to its designer (ralf@sea.mit.edu).

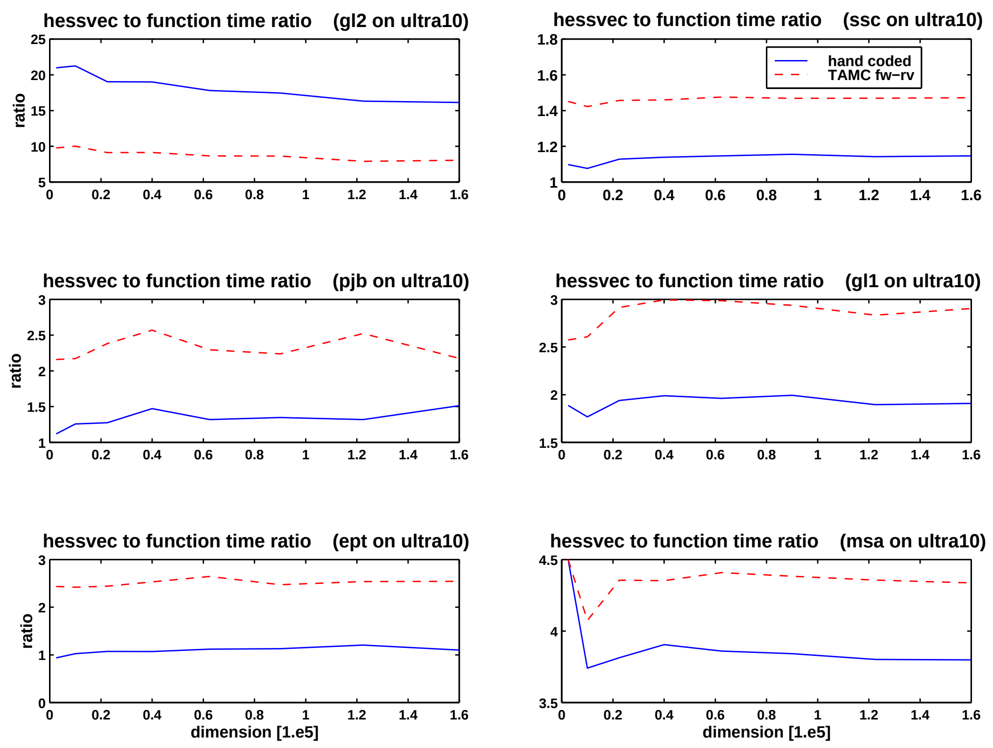

Figure 3: Relative run time of Hessian times vector code on Sun Ultra-1.

References

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

[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*, 3:625–638, 1992.

[5] Christian Eckert. *On Predictability Limits of ENSO - A Study Performed with a Simplified Model of the Tropical Pacific Ocean-Atmosphere System.* PhD thesis, Max-Planck-Institut für Meteorologie, Hamburg, Germany, 1998.

[6] Ralf Giering. Tangent linear and Adjoint Model Compiler home page. URL=`http://puddle.mit.edu/~ralf/tamc`.

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

[8] Ralf Giering and Thomas Kaminski. Recipes for Adjoint Code Construction, 1998. in press ACM Trans. On Math. Software.

[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*, 1:35–54, 1992.

[11] Andreas Griewank. Some bounds on the complexity of gradients, Jacobians, and Hessians. In Panos M. Pardalos, editor, *Complexity in Nonlinear Optimization*, pages 128–161. World Scientific Publishers, 1993.

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

[13] 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(5–6):457–462, 1996.

[14] Geert Jan van Oldenborgh, Gerrit Burgers, Stephan Venzke, Christian Eckert, and Ralf Giering. Tracking down the delayed ENSO oscillator with an adjoint OGCM. Technical Report 97-23, Royal Netherlands Meteorological Institute, P.O. Box 201, 3730 AE De Bilt, The Netherlands, 1997. Monthly Weather Review, in press.

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

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

[17] 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. Technical Report 49, Center for Global Change Science, Massachusetts Institute of Technology, 1997.

[18] 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.

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