Contents

20

Integrating AD with Object-Oriented Toolkits for High-Performance Scientific Computing

Jason Abate, Steve Benson, Lisa Grignon, Paul Hovland, Lois McInnes and Boyana Norris

ABSTRACT Often the most robust and efficient algorithms for the solution of large-scale problems involving nonlinear PDEs and optimization require the computation of derivatives. We examine the use of automatic differentiation (AD) for computing first and second derivatives in conjunction with two parallel toolkits, the Portable, Extensible Toolkit for Scientific Computing (PETSc) and the Toolkit for Advanced Optimization (TAO). We discuss how the use of mathematical abstractions in PETSc and TAO facilitates the use of AD to automatically generate derivative codes and present performance data demonstrating the suitability of this approach.

20.1 Introduction

As the complexity of advanced computational science applications has increased, the use of object-oriented software methods for the development of both applications and numerical toolkits has also increased. The migration toward this approach can be attributed in part to code reusability and encapsulation of data and algorithms. Code reuse justifies expending significant effort in the development of highly optimized object toolkits encapsulating expert knowledge. Furthermore, well-designed toolkits enable developers to focus on a small component of a complex system, rather than attempting to develop and maintain a monolithic application.

Many high-performance numerical toolkits include components designed to be combined with an application-specific nonlinear function. Examples include optimization components, nonlinear equation solvers, and differential algebraic equation solvers. Often the numerical methods implemented by these components also require first and possibly second derivatives of the function. Frequently, the toolkit is able to approximate these derivatives by using finite differences (FD); however, the convergence rate and robustness are often improved if the derivatives are computed analytically.

This represents an ideal situation for using automatic differentiation (AD) [14, 15]. Developing correct parallel code for computing the derivatives of a complicated nonlinear function can be an onerous task, making an automated alternative very attractive. In addition, the well-defined interfaces used by object toolkits simplify AD by removing the need for the programmer to identify the independent and dependent variables and/or write code for the initialization of seed matrices.

We examine the use of AD to provide code for computing first and second derivatives in conjunction with two numerical toolkits, the Portable, Extensible Toolkit for Scientific Computing (PETSc) and the Toolkit for Advanced Optimization (TAO). We describe how AD can be used with these toolkits to generate the code for computing the derivatives automatically. We present results demonstrating the suitability of AD and PETSc for the parallel solution of nonlinear PDEs and mention preliminary results from the use of AD with TAO.

20.2 Portable, Extensible Toolkit for Scientific Computing

PETSc [2, 3] is a suite of data structures and routines for the scalable solution of scientific applications modeled by partial differential equations. The software integrates a hierarchy of components that range from low-level distributed data structures for vectors and matrices through high-level linear, nonlinear, and timestepping solvers. The algorithmic source code is written in high-level abstractions so that it can be easily understood and modified. This approach promotes code reuse and flexibility and, in many cases, helps to decouple issues of parallelism from algorithm choices.

Newton-based methods (see, e.g., [23]), which offer the advantage of rapid convergence when an iterate is near to a problem's solution, form the algorithmic core of the nonlinear solvers within PETSc. The methods employ line search, trust region, and pseudo-transient continuation strategies to extend the radius of convergence of the Newton techniques, and often solve the linearized systems inexactly with preconditioned Krylov methods. The basic Newton method requires the Jacobian matrix, $J = F'(u)$, of a nonlinear function $F(u)$. Matrix-free Newton-Krylov methods require Jacobian-vector products, $F'(u)v$, and may require an approximate Jacobian for preconditioning.

20.3 Toolkit for Advanced Optimization

TAO [4, 5] focuses on scalable optimization software, including nonlinear least squares, unconstrained minimization, bound constrained optimization, and general nonlinear optimization. The TAO optimization algorithms use high-level abstractions for matrices and vectors and emphasize the reuse of external tools where appropriate, including support for using the linear algebra components provided by PETSc and related tools.

Many of the algorithms employed by TAO require first and sometimes second derivatives. For example, unconstrained minimization solvers that require the gradient, $f'(u)$, of an objective function, $f(u)$, include a limited-memory variable metric method and a conjugate gradient method, while solvers that require both the gradient, $f'(u)$, and Hessian, $f''(u)$, (or Hessian-vector products) include line search and trust region variants of Newton methods. In addition, algorithms for nonlinear least squares and constrained optimization often require the Jacobian of the constraint functions.

20.4 Experimental Results

The AD tools used in this research are the ADIFOR [6] and ADIC [7] systems. Given code for a function $f(u)$ in Fortran 77 or ANSI C, respectively, these tools generate code for $f'(u)$ and, if desired, $f''(u)$.

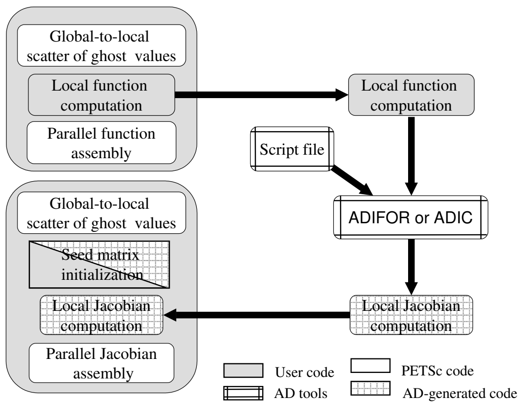

FIGURE 20.1. Schematic diagram of the use of AD tools to generate the Jacobian routine for a nonlinear PDE computation.

Other work [11, 12, 21, 22] has demonstrated the benefits of well-defined interfaces for automating the AD process. The object-oriented designs of PETSc and TAO lead to such well-specified interfaces. However, rather than exploit this feature directly, we have chosen to take advantage of the structure of a typical PETSc/TAO nonlinear function evaluation to simplify the AD process. The parallel nonlinear function code usually includes several calls to PETSc/TAO routines for generalized vector scatters before and after the actual local function computation. These calls take care of data structure setup and communication, enabling a completely local function computation. In the current semi-automatic approach, illustrated in Figure 20.1, we differentiate this local function using AD. This produces code for local derivative computation, which is coupled with code for assembling the gradient, Jacobian, or Hessian. While in principle it is possible to differentiate through the parallel scatter and assembly routines [8, 10, 17, 18], currently the corresponding seed-matrix initialization and assembly code are generated manually. Future development will automate this process.

We present experimental results for nonlinear PDEs and unconstrained minimization problems that demonstrate the utility of AD in conjunction with parallel numerical libraries. These computations were run on an IBM SP with 120 MHz P2SC nodes with two 128 MB memory cards each and a TB3 switch. We have observed analogous qualitative behavior on a range of other current parallel architectures.

20.4.1 Using AD and PETSc for nonlinear PDEs

We used AD and PETSc to solve the steady-state, three-dimensional compressible Euler equations on mapped, structured meshes using a second-order, Roe-type, finite-volume discretization. In particular, we solved in parallel a nonlinear system, using matrix-free Newton-Krylov-Schwarz algorithms with pseudo-transient continuation to model transonic flow over an ONERA M6 airplane wing. See [16] for details about the problem formulation and algorithmic approach. The linearized Newton correction equations were solved by using restarted GMRES preconditioned with the restricted additive Schwarz method with one degree of overlap.

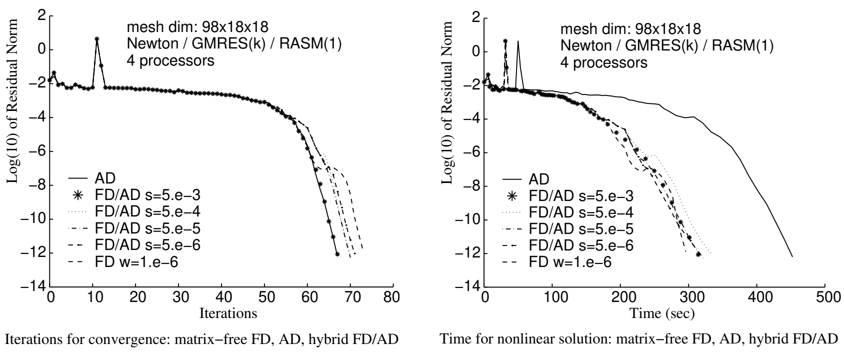

FIGURE 20.2. Comparison of convergence using matrix-free Newton-Krylov-Schwarz methods with finite differencing (FD), AD, and hybrid variants (FD/AD) with various switching parameters, s. Iterations (left) and time (right).

As discussed in depth in [19] and summarized in Figure 20.2, our results indicate that, for matrix-free Newton-Krylov methods, AD offers sig-

TABLE 20.1. Execution times (sec) for various stages of the elastic-plastic torsion minimization problem.

Number of Processors	1		16		32	
Number of Vertices	10,000		160,000		320,000	
Hessian Computation Method	FD	AD	FD	AD	FD	AD
Linear System Solution	1.19	1.20	7.10	7.04	9.16	9.04
Compute Function	0.01	0.01	0.04	0.02	0.04	0.02
Compute Gradient	0.21	0.20	0.22	0.24	0.23	0.24
Compute Hessian	1.89	1.48	3.86	1.58	6.11	1.65
Total Time	5.20	4.86	14.13	11.87	18.73	14.32

nificantly greater robustness and converges in fewer iterations than FD. This figure plots the residual norm $\|F(u)\|_2$ versus both nonlinear iteration number and computation time on four processors for a model problem of dimension 158,760. The runtime for AD is slightly higher than for FD (when an appropriate step size is used), due to the higher cost of computing Jacobian-vector products using AD. However, coupling AD with FD in a hybrid scheme provides the robustness of AD with the lower computation time of FD, without needing to identify the proper step size for FD. Additional experiments show that this hybrid technique scales well [19].

20.4.2 Using AD and TAO for unconstrained minimization

We evaluated the preliminary performance of AD in conjunction with TAO using a two-dimensional elastic-plastic torsion model from the MINPACK-2 test problem collection [1]. This model uses a finite element discretization to compute the stress field on an infinitely long cylindrical bar to which a fixed angle of twist per unit length has been applied. The resulting unconstrained minimization problem can be expressed as $min\, f(u)$, where $f : \Re^n \to \Re$. All of the following numerical experiments use a line search Newton method with a preconditioned conjugate gradient linear solver.

We compared two approaches for computing the full Hessian of $f(u)$. First, we applied AD to the hand-coded routine computing the analytic gradient of $f(u)$. Second, we used FD to approximate the Hessian, again using the analytic gradient of $f(u)$. In both cases we used graph coloring techniques to exploit the sparsity of the Hessian computation [9, 13].

The graphs in Figure 20.3 show the scaling of the complete minimization problem using either no preconditioning or a block Jacobi preconditioner. The block Jacobi preconditioner uses a subdomain solver of ILU(0) on each block, with one block per processor. For this relatively simple problem, both AD and FD exhibit rapid convergence in terms of number of iterations. However, AD outperforms FD in terms of total time to solution, mainly because of the good scalability of the AD Hessian computation. Overall, the results for a fixed local 100×100 mesh size indicate that the problem

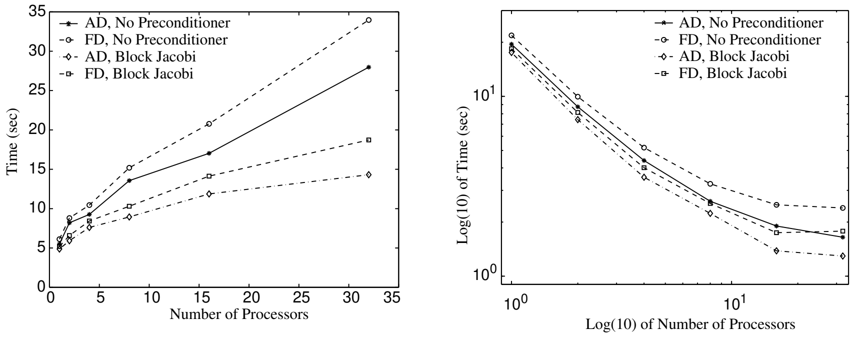

FIGURE 20.3. Comparison of total execution times for the elastic-plastic torsion minimization problem using a line search Newton method. Fixed local (left) and fixed global (right) problem sizes 10,000 and 22,500, respectively.

does not scale well for either AD or FD (although using a block Jacobi preconditioner helps somewhat). This situation is due in part to the poor performance scaling of the linear system solution (see Table 20.1). We are currently exploring the causes of this poor scalability.

20.5 Summary and Future Work

We have presented a methodology for using AD to compute first and second derivatives for use in the parallel solution of nonlinear PDEs and optimization. The robustness and, in some cases, performance of the resulting code are superior to results with finite difference approximations. Also, in contrast to hand-coding, AD can easily be re-applied if the function changes.

Future work includes moving from a semi-automatic approach to an approach, in which code for seed matrix initialization and derivative matrix assembly are completely automated. This task will be aided by the existence of well-defined interfaces for the nonlinear function component and will leverage work on developing a differentiated version of PETSc [20].

Acknowledgments: This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

We thank Jorge Moré and Barry Smith for stimulating discussions about issues in derivative computations with PDE and optimization software and Gail Pieper for proofreading an early draft of this manuscript.

[1] Brett M. Averick, Richard G. Carter, Jorge J. Moré, and Guo-Liang Xue. The MINPACK-2 test problem collection. Preprint MCS–P153–0692, ANL/MCS–TM–150, Rev. 1, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1992. See `ftp://info.mcs.anl.gov/pub/MINPACK-2/tprobs/P153.ps.Z`.

[2] Satish Balay, William D. Gropp, Lois C. McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and Hans Petter Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.

[3] Satish Balay, William D. Gropp, Lois C. McInnes, and Barry F. Smith. The portable extensible toolkit for scientific computing PETSc 2.0 users manual. Technical Report ANL-95/11 - Revision 2.0.28, Argonne National Laboratory, Mar 2000. See `www.mcs.anl.gov/petsc`.

[4] Steve Benson, Lois C. McInnes, and Jorge J. Moré. GPCG: A case study in the performance and scalability of optimization algorithms. Technical Report ANL/MCS-P768-0799, Mathematics and Computer Science Division, Argonne National Laboratory, 1999.

[5] Steve Benson, Lois C. McInnes, and Jorge J. Moré. TAO users manual. Technical Report ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2000. See `www.mcs.anl.gov/tao`.

[6] Christian H. Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. *IEEE Computational Science & Engineering*, 3(3):18–32, 1996.

[7] Christian H. Bischof, Lucas Roh, and Andrew Mauer. ADIC — An extensible automatic differentiation tool for ANSI-C. *Software–Practice and Experience*, 27(12):1427–1456, 1997. See `www-fp.mcs.anl.gov/division/software`.

[8] Alan Carle. Personal communication, 1997.

[9] Thomas F. Coleman and Jorge J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20(1):187–209, 1983.

[10] Christèle Faure and Patrick Dutto. Extension of Odyssée to the MPI library - Direct mode. Technical Report 3715, INRIA, June 1999.

[11] Michael C. Ferris, Mike P. Mesnier, and Jorge J. Moré. NEOS and Condor: Solving optimization problems over the Internet. Preprint ANL/MCS-P708-0398, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, 1998.

[12] E. Michael Gertz, Philip E. Gill, and Julia Muetherig. User's guide for SnadiOpt: A package adding automatic differentiation to Snopt. Report NA 01–1, Department of Mathematics, University of California, San Diego, 2000.

[13] Donald Goldfarb and Philippe L. Toint. Optimal estimation of Jacobian and Hessian matrices that arise in finite difference calculations. *Mathematics of Computation*, 43(167):69–88, 1984.

[14] Andreas Griewank. On automatic differentiation. In Masao Iri and Kunio Tanabe, editors, *Mathematical Programming: Recent Developments and Applications*, pages 83–108. Kluwer Academic Publishers, Dordrecht, 1989.

[15] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, Penn., 2000.

[16] William D. Gropp, David E. Keyes, Lois C. McInnes, and M. D. Tidriri. Globalized Newton-Krylov-Schwarz algorithms and software for parallel implicit CFD. *Int. J. High Performance Computing Applications*, 14:102–136, 2000.

[17] Paul D. Hovland. *Automatic Differentiation of Parallel Programs.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Ill., May 1997.

[18] Paul D. Hovland and Christian H. Bischof. Automatic differentiation of message-passing parallel programs. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 98–104, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[19] Paul D. Hovland and Lois C. McInnes. Parallel simulation of compressible flow using automatic differentiation and PETSc. Technical Report ANL/MCS-P796-0200, Mathematics and Computer Science Division, Argonne National Laboratory, 2000. To appear in a special issue of *Parallel Computing* on "Parallel Computing in Aerospace".

[20] Paul D. Hovland, Boyana Norris, Lucas Roh, and Barry F. Smith. Developing a derivative-enhanced object-oriented toolkit for scientific computations. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, pages 129–137, Philadelphia, Penn., 1998. SIAM.

[21] Steven L. Lee and Paul D. Hovland. Sensitivity analysis using parallel ODE solvers and automatic differentiation in C: SensPVODE and ADIC. In George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation: From Simulation to Optimization*, Computer and Information Science, chapter 26, pages 223–229. Springer, New York, 2001.

[22] Shengtai Li and Linda Petzold. Design of new DASPK for sensitivity analysis. Technical Report TRCS99–28, University of California at Santa Barbara, 1999.

[23] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, New York, 1999.

Index