An Overview of Automatic Differentiation Tools and Techniques for Nuclear Reactor Applications

Mihai Anitescu, Paul Hovland, Giuseppe Palmiotti, and Won Sik Yang

Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439; {anitescu,Hovland}@mcs.anl.gov, {GPalmiotti,wyang}@anl.gov

INTRODUCTION

Automatic differentiation is an established technique for computing local sensitivities of complex simulations [1]. Modern automatic differentiation tools typically produce code that meets or exceeds the performance of hand-coded derivative computations. Automatic differentiation has been used extensively in the geosciences and aeronautical engineering, but it also has the potential to be a useful tool for nuclear reactor design and evaluation.

AUTOMATIC DIFFERENTIATION

Automatic differentiation (AD) is a family of techniques for computing the derivatives of a function defined by a computer program. The basis for AD is the assumption that the computation of a vector function $y = f: R^n \rightarrow R^m$ is accomplished by a sequence of p elemental operations $v_i = F_i(\ldots, v_j, \ldots)$, $i = 1,\ldots,p$ as found in a computer program implementing an evaluation procedure for f. Each of the F_i is differentiable at least in open subdomains. The derivatives of these elemental operations are combined according to the chain rule of differential calculus.

The associativity of the chain rule leads to the two major modes of computing derivatives with AD. The *forward mode* multiplies derivatives starting with the independent variables and proceeding toward the dependent variables. The *reverse mode* multiplies derivatives starting with the dependent variables and proceeding toward the independent variables.

In abstract terms, the number of operations (temporal complexity) required for computing the Jacobian of $f: R^n \rightarrow R^m$ in the forward mode is O(n) while the computation with reverse mode is O(m). Thus, the forward mode is appropriate for functions with small numbers of independent variables (or, via a simple transformation, Jacobian-vector products), while the reverse mode is appropriate for functions with small numbers of dependent variables (or, via a simple transformation, transposed-Jacobian-vector products). In practice, the adjoint mode must recompute intermediate quantities to avoid excessive memory requirements. Maintaining a proper balance between storage and recomputation requires sophisticated tools and users.

TOOLS

Many automatic differentiation tools are available today [2]. Established tools such as ADIFOR (for Fortran 77) and ADOL-C (for C/C++) are robust and reliable. Newer tools such as OpenAD/F, Tapenade, and TAF (for Fortran 90) and ADIC and TAC++ (for C/C++) offer additional functionality and support for parallelism [3,4] and often incorporate sophisticated compiler analysis [5,6] to reduce the cost of sensitivity and adjoint computation. Tools such as ADiMat and MAD support differentiation of simulation prototypes implemented in Matlab. Our group is actively developing ADIC and OpenAD/F and is investigating ways to make these tools more suitable for use in reactor design simulation.

TECHNIQUES

The forward mode is well suited for computing the derivative of a large number of observations with respect to a small number of design or physical parameters, with time and memory costs proportional to the number of parameters. It can also be adapted to the computation of sparse Jacobians. The forward mode is ill suited for computing sensitivities with respect to a large number of parameters. In contrast, the reverse mode is well suited to computing the derivative of a single or small number of cost or quality metrics with respect to a large number of design parameters or other unknowns, with time requirements proportional to the number of metrics. The memory requirements for the reverse mode may be a logarithmic factor larger than those for the original simulation. A combination of forward and reverse modes can be used to compute low rank Jacobians.

Unfortunately, neither mode is well suited for computing the Jacobians of large numbers of observations with respect to a large numbers of parameters. This is precisely the situation that may arise in nuclear reactor simulations, where one frequently wishes to compute the derivatives of thousands or millions of state variables with respect to millions of cross sections and other parameters. This situation mandates the use of something other than pure forward or reverse mode computations. We are therefore developing techniques to break a simulation into components whose Jacobians are effectively sparse or low rank, then assemble the full Jacobian from these component Jacobians. We conjecture that such problem decomposition can best be accomplished through a

combination of compiler analysis and user insight. Effectively sparse or low-rank Jacobians can then be approximated through the use of carefully chosen Jacobian-vector products.

EXAMPLE

For example, consider the case of a function $f = F(G(H_1(x),H_2(x)))$,

$F: R^m \rightarrow R^n$
$G: R^{m+1} \rightarrow R^m$
$H_1: R^n \rightarrow R^m$
$H_2: R^m \rightarrow R$

implemented as

```
call sub_H1(x,y1)
call sub_H2(x,y2)
call sub_G(y1,y2,z)
call sub_F(z,f)
```

where the Jacobian of H_1 is sparse, the Jacobian of G is low rank, and the Jacobian of F is effectively sparse, with most elements below some threshold ε. The sparsity pattern of dy_1/dx can be determined by compiler analysis or inexpensive runtime analyses. We believe that these techniques can be adapted to the effectively sparse case, but otherwise expert knowledge can be employed. The reverse mode can efficiently compute the gradient of H_2, dy_2/dx. The Jacobians of H_1 and F can be computed by using coloring or partial coloring, respectively [7]. The Jacobian of G can be approximated by using low-rank updates (Broyden, TR1 [8]) or by adapting the efficient subspace method [9] to forward or reverse mode sensitivity computations. Once the individual Jacobians have been computed, the chain rule can be used to compute the total Jacobian, df/fx.

ACKNOWLEDGMENT

Work supported by DOE under Contract DE-AC02-06CH11357.

REFERENCES

1. A. GRIEWANK. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, 2000.
2. http://www.autodiff.org/?module=Tools
3. P. D. HOVLAND. "Automatic Differentiation of Parallel Programs." Ph.D. thesis, University of Illinois at Urbana-Champaign, 1997.
4. M. M. STROUT, B. KREASECK, and P. D. HOVLAND. "Data-Flow Analysis for MPI Programs," in Proceedings of the International Conference on Parallel Processing, August 2006.
5. L. HASCOET, U. NAUMANN, and V. PASCUAL. *Future Generation Computer Systems*, 21:8, 1401–1417 (2005).
6. M. M. STROUT, J. MELLOR-CRUMMEY, and P. D. HOVLAND. "Representation-Independent Program Analysis," in Proceedings of the Sixth ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, September 5–6, 2005.
7. A. H. GEBREMEDHIN, F. MANNE, and A. POTHEN. *SIAM Review,* 48:4, 629– 705 (2005).
8. A. GRIEWANK and A. WALTHER. *Optimization Methods and Software*, 17:5, 869– 889 (2002).
9. H. S. ABDEL-KHALIK, P. J. TURINSKY. *Nuclear Technology*, 151:1, 9– 21 (2005).