Preliminary Study of Analytic Sensitivities for CFD Codes Through Automatic Differentation

Alan Carle
Mike Fagan

CRPC-TR96662
October 1996

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Preliminary Study of Analytic Sensitivities for CFD Codes Through Automatic Differentiation [1]

CRPC-TR96662.

Alan Carle (carle@cs.rice.edu)
Mike Fagan (mfagan@cs.rice.edu)

Rice University
Center for Research on Parallel Computation
6100 Main St., MS 41
Houston, Texas 77005

1 Introduction

Modern engineering design depends on sensitivity calculations. To obtain these sensitivities, engineers have two choices: use finite differences, or develop a program to compute the analytic derivatives. To avoid programming, engineers have typically chosen to use finite differences. The accuracy of finite difference calculations, however, depends on getting a "good" step size. A "good" step size, however, depends on two competing factors. To approximate the derivative well, the step size should be "small." On the other hand, to avoid critical cancellation, the step size should be "moderate." Finding an appropriate step size to satisfy both criteria can be difficult, especially for complex processes like CFD calculations. Furthermore, the step size exercise may need to be repeated for each new point in the design space.

Analytic derivatives, by definition, avoid the step size problem. Generating an analytic derivative program by hand, however, is tedious, time consuming, and prone to error. Fortunately, engineers can avoid the "by hand" part by using an automatic differentiation (AD) tool. An AD tool automatically applies the derivative chain rule to all of the expressions in a program, thereby generating a program that computes both the desired analytic derivatives, in addition to the function value. The specific AD tool we use for our research is ADIFOR 2.0, developed jointly by Argonne National Laboratory and Rice University. ADIFOR stands for Automatic Differentiation of Fortran. Our hardware testbeds consisted of Rice's 8 node SP2 (RS6000 nodes), and Langley's Cray YMP. To find out more about ADIFOR, see `http://www.mcs.anl.gov/adifor` or `http://www.cs.rice.edu/~adifor`.

In this work, we describe results obtained by application of automatic differentiation to OVERFLOW, an extensively used CFD code from NASA Ames. OVERFLOW is currently a focal point of a NASA project that uses derivatives for shape optimization and inverse problems. The projected problems are expected to be large, and, more significantly, require a large number of derivatives. For these large problems, a naive ("black box") use of AD will probably be ineffective. Consequently, we have been exploring more sophisticated techniques for computing CFD derivatives. In particular, we have been exploring the use of stripmining to exploit available parallel resources. In addition, we have also explored the incremental iterative method (IIM) for computing derivatives. The IIM method reduces the amount of redundant computation that occurs with naive application of AD.

2 Black Box Use of AD

The ADIFOR 2.0 tool can augment any pure Fortran 77 program to produce analytic derivatives. OVERFLOW, however, uses some non-Fortran subroutines to do dynamic memory allocation. Fortunately, the memory management is sufficiently disciplined that straightforward post-processing of the ADIFOR generated source resulted in correct derivative code. We refer to this use of ADIFOR as "black box" (BB).

As seen from Figure 1, the BB calculation for the RS6000 is about 5 times the cost of a function evaluation, which places its cost between one-sided and two-sided difference methods. As shown in Figure 1, the BB

[1]Research funding provided by NASA Ames High Performance Computing and Communication Program Testbed 1 Cooperative Research Agreement (through IBM P.O. # 835600Z-E1). This material was presented at the NASA Ames Computational Aerosciences Workshhop '96.

computation cost for the Cray shows a penalty of 10 times the cost of the function evaluation — much more expensive than even two-sided differences.

Furthermore, the memory requirements for a derivative code can be given as $O(I*M)$, where I is the number of independent variables in the differentiation, and M is the memory required for the function evaluation. Hence, for large problems, the memory requirements are likely to be the limiting resource with regard to derivative computation.

The relatively steep cost of computing derivatives can be alleviated somewhat by use of extra processors using OVERFLOW's own PVM-based parallel capability. OVERFLOW is an overlapping multizone flow solver. Current parallel versions use one process per zone with one additional master process coordinating the zone interactions. To differentiate parallel OVERFLOW, we used ADIFOR-MP, a new "message passing aware" version of ADIFOR. ADIFOR-MP processed parallel OVERFLOW, yielding a parallel multizone derivative code. The performance of the largest zone determines the overall performance of the parallel derivative code. For parallel OVERFLOW on the SP2, the ratio of derivative cost to function cost was still about 5.0, the same as for sequential OVERFLOW. The memory requirements, however, are reduced in the sense that each processor needs only enough memory for one zone, not the entire grid. In this way, function code parallelism enables parallel derivative calculation. As an additional benefit, for problems with large memory requirements, function grids could be divided into smaller zones, leaving more memory available for derivative computations.

3 Parallel Derivative Computation

Independent of the parallelism of the CFD code, the derivative computation can be parallelized by the simple expedient of computing derivatives with respect to different sets of independent variables on different processors. This method, called stripmining, has already been described in [1]. As shown in [1], ADIFOR-augmented code supports this method of parallel derivative computation with minimal programmer effort. We note, however, that this method redundantly recomputes the function value on each processor doing derivative work. We implemented a simple experiment on our SP2, comparing 1 processor computing 6 derivatives (13.27 seconds/step), 2 processors each computing 3 derivatives (8.21 seconds/step), and 6 processors each computing 1 derivative (4.02 seconds/step). Furthermore, since memory usage is approximately linear in the number of derivatives, using 6 processors instead of 1, reduces the memory required by about a factor of 6.

4 Intelligent Use of Domain Knowledge

Instead of (or perhaps in addition to) brute processor power, we can gain leverage on the high cost of derivatives by exploiting specific properties of CFD to formulate a more efficient derivative computation strategy. One such promising formulation is the incremental iterative method (IIM) [3,4]. The basic idea behind the IIM derives from the properties of many CFD solvers. Most implicit methods repeatedly solve

$$As = r$$

where A is some linearization of the problem, r is the residual, and s is the flow field update. Each time step computes a new update s to the flow field. The derivative of the process yields

$$A's + As' = r'.$$

Straightforward BB calculation implements this iterative process. Now suppose, however, that the flow field is converged. In this case s should be "small," and the BB iteration simplifies to

$$As' = r'.$$

Note that this process differs from the original process only in the use of r'. The IIM method for calculating derivatives, then, consists of two separate processes. First, converge the flow field. Second, compute r', and using the original solver, compute the update for the flow field derivatives. Repeat the second step until the derivatives converge.

The IIM formulation has a number of advantages:

- The flow field is converged using the original "tuned" solver, free of interference from derivative augmentation.
- The derivatives of the flow field are converged using the same "tuned" solver.
- Derivative calculation for some variables can sometimes be factored out of the main time-stepping loop.

At present, construction of an IIM derivative code cannot be totally automated. However, for well-modularized programs, like OVERFLOW, an IIM code can be fairly easily constructed with the aid of an AD tool. Such a construction proceeds by identifying the parts of the CFD program that compute r, differentiating those parts, and then reusing the original solver routine. We constructed an IIM code for sequential OVERFLOW1.6ap, using ADIFOR. The details of the construction appear in [2].

From Figure 1, the IIM improvement is modest. Further improvements were made by converting OVERFLOW's solver to use multiple right hand sides, thereby eliminating the cost of repeated factoring for multiple derivatives. We constructed a special tool based on ADIFOR's program analysis to do this conversion. The timings in Figure 1 show significant improvement over BB computation and are much closer to the one-sided difference scheme than a two-sided difference scheme.

One other possibility for potential improvements has been suggested by Art Taylor. Taylor suggested that the use of the already converged flow field in IIM should have increased stability properties, and thus the size of the time step at each iteration could be increased. Increasing the time step would have the effect of lowering the number of iterations necessary to get a final answer. We increased the time steps by factors of 2,4, and 8. In all cases, the answer converged to the correct AD value. The number of iterations was reduced by factors of 2, 4, and 4. Apparently, increasing the time step by a factor of 8 does not improve the convergence any more than a does a factor of 4. See Figure 2 for the convergence plots. We do not know if the CFL number can always be increased, but the technique does bear further consideration.

5 Conclusions and Open Questions

The relevant timings for the various different kinds of derivative evaluations for the RS6000 (SP2) and Cray appear in Figure 1. From this table, we argue that IIM with multiple right hand sides is our most promising technique on a single processor. The IIM technique will be spectacular if the time step speedup is sound.

Using parallelism is also a fine way to generate improvement. The multizone parallelism currently built into OVERFLOW carries its advantage to derivative computation as well. In addition, the derivative computation may be advantageously split among several processing elements, via the stripmining idea.

The main import of this work is to show that automatic differentiation combined with other sophisticated techniques provides a powerful toolbox for computing analytic derivatives. As always, however, the more problem-specific knowledge one can bring to bear, the more effective the tools will be. Looking ahead, we expect to use all three techniques on current design problems. Our first step towards this goal is to combine IIM with multizone parallelism and to apply this to a realistic shape optimization problem.

6 Afterword

While attempting to verify derivatives for one of our test cases, we encountered the following problem (phenomenon?). We were attempting to use finite differences to verify a derivative computed by code generated by ADIFOR. The derivative in question was coefficient of lift with respect to angle of attack. Using an ostensibly reasonable step size of 1.0e-1, we generated the disturbing finite-difference plot shown at the far left of Figure 3. The large scale of the derivative values relative to the finite difference values worried us. After several rounds of tracing and debugging indicated nothing, we attempted other step sizes. As can be seen from Figure 3, finite difference at smaller step sizes does indeed increase the scale. Since the analytic derivative is a limiting process, the evidence indicates that the computed analytic derivatives are the correct derivatives of the CFD process. We are not sure, why the coefficient of lift appears to be so sensitive to the angle of attack.

7 References

1. C. H. Bischof, L. L. Green, K. J. Haigler, and Jr. T. L. Knauff. Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation. In AIAA/NASA/USAF/ISSMO 5th Symposium on Multidisciplinary Analysis and Optimization, pages 73-86, 1994.

2. Alan Carle and Mike Fagan. Improving derivative performance for CFD using simplified recurrences. Technical Report CRPC-TR96643, Rice University/CRPC, March 1996. Also to appear in Proceedings of the Second International Workshop on Computational Differentiation.

3. Andreas Griewank, Christian Bischof, George Corliss, Alan Carle, and Karen Williamson. Derivative convergence for iterative equation solvers. Optimization Methods and Software, 2:321-355, 1993.

4. L.L. Sherman, A.C. Taylor, L.L. Green, P.A. Newman, G. J.-W. Hour, and V.M. Korivi. First- and second-order aerodynamic sensitivity derivatives via automatic differentiation with incremental iterative methods. In AIAA/NASA/USAF/ISSMO 5th Symposium on Multidisciplinary Analysis and Optimization, pages 87-120, 1994.

Evaluation	SP2 (RS6000) Time per Step (sec)	Cray Time per Step (sec)
The original CFD function evaluation	1.62	0.26
"Black box" derivatives (includes 1 function evaluation)	8.21	2.60
Incremental iterative method	8.03	2.16
Incremental iterative method, with multiple right hand sides	7.00	1.56
Incremental iterative method, multiple right hand sides, time scale multiplied by 4	1.75	0.39
One-sided differences	4.86	0.78
Two-sided differences	9.72	1.56

Figure 1 Timings for single processor derivatives. (All derivative computation done with 3 independent variables. The derivative times shown are the necessary times to compute the derivative only. Note that the BB evaluation must do a function evaluation to obtain its derivative value.)

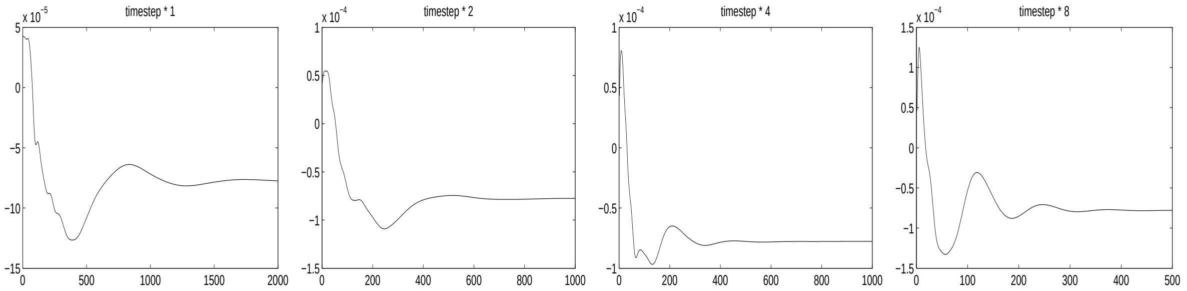

Figure 2 Convergence behavior for IIM with increased time steps

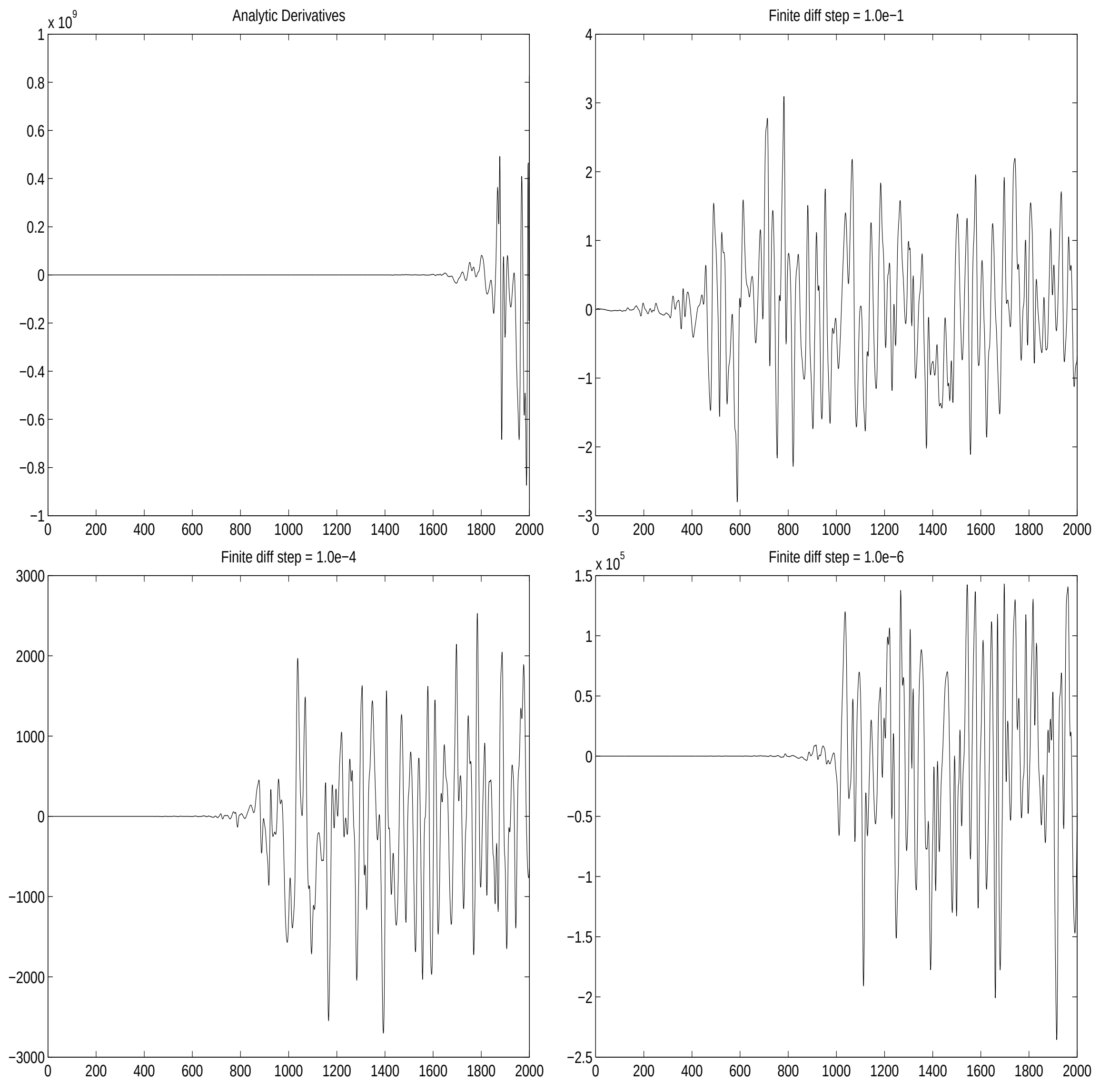

Figure 3 Afterword: Unusual sensitivity for wing component. Derivative of lift coefficient with respect to angle of attack, shown from 1 to 2000 time steps.