Preliminary Results from the Application of Automatic Adjoint Code Generation to CFL3D

Alan Carle, Mike Fagan, and Lawrence L. Green

CRPC-TR98741
February 1998

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

Submitted February 1998; Submitted to the 7th AIAA/USAF/NASA/ISSMO Multidisciplinary Analysis and Optimization Symposium

Preliminary Results from the Application of Automatic Adjoint Code Generation to CFL3D[1]

Alan Carle and **Mike Fagan**[2]

Lawrence L. Green[3]

Submitted to the
7th AIAA/USAF/NASA/ISSMO
Multidisciplinary Analysis
and Optimization (MA&O) Symposium

CRPC-TR98741

February, 1998

Abstract

We describe ADJIFOR, an automated adjoint code generator for Fortran, and present preliminary results from the application of ADJIFOR to augment the CFL3D flow solver. Based on these initial results with CFL3D, we argue that, even in its infancy, ADJIFOR can deliver derivatives accurately and efficiently for use in aerodynamic shape optimization problems with 200 to 500 independent design variables.

Keywords: ADIFOR, ADJIFOR, Adjoint code generation, Automatic differentiation, CFL3D

1 Introduction

Automatic differentiation (AD) is a set of techniques for automatically augmenting computer codes to compute derivatives of their outputs with respect to their inputs. Numerous papers presented at recent MDO conferences have claimed that AD can provide the derivatives required for use in simulation-based design [17, 2, 9, 10, 15, 19, 11, 8]. These papers describe the use of ADIFOR, an AD tool for Fortran [3, 4]. ADIFOR implements the "forward mode" of AD. For a code with n independent variables (or design variables) and m dependent variables, the forward mode computes the derivatives using time and space proportional to n. Obviously, for problems with a large number of independent variables, this cost is prohibitive.

ADJIFOR, a substantially extended version of ADIFOR, implements the "reverse (or adjoint) mode" of AD. For a code with n independent variables and m dependent variables, the reverse mode computes the derivatives using time proportional to m, not n, albeit by using space proportional to the *time* required to execute the original code. Fortunately, as we will show, for CFL3D, we can exploit the special mathematical properties of the steady-state solution to dramatically reduce the space requirements of the reverse mode.

2 The Shape Optimization Problem

Aerodynamic shape *analysis* couples a grid generator to a flow solver. Given shape *shape parameters*, the grid generator creates a grid. The newly created grid then becomes an input of the flow solver, and the output of the flow solver are the aerodynamic quantities of interest. Thus, for a given set of shape parameters, the grid generator/flow solver combination determines the aerodynamic quantities of interest. Often, the point of shape analysis is to determine the values of shape parameters that give rise to "favorable" aerodynamic

[1]This work was supported by the National Aeronautics and Space Administration under Cooperative Agreement Number NCC 1 234, and by the National Science Foundation, through the Center for Research on Parallel Computation, under Cooperative Agreement No. CCR-9120008. Carle and Fagan were responsible for adjoint code generation and initial code execution demonstrations. Green was responsible for the problem definition, grid generation, and CFL3D code support.

[2]Rice University MS 41, Center for Research on Parallel Computation, 6100 Main Street, Houston, TX 77005

[3]Mail Stop 159, 18-C West Taylor Street, NASA Langley Research Center, Hampton, VA 23681-2199

outputs. The process of seeking shape inputs that lead to the favorable outputs is called shape *optimization*. To convert a shape analysis problem into a shape optimization problem requires defining exactly what criteria constitute "favorable." The definition of favorable includes an *objective function* to be minimized or maximized. Geometric and flow constraints might also be included.

Gradient-based optimization requires the derivatives of the objective function and the constraints with respect to the parameters that control the shape of the aerodynamic body. In the following we use Q for flow fields, X for the grid, and B for the shape parameters. A grid generator G generates the grid X, given shape parameters B, that is, $X = G(B)$. The flow solver computes the final "converged" flow field by iterating a stepping function S. The stepping function computes the next iterate, using the current iterate and grid. We emphasize this dependence by writing $S(Q, X)$ for the step. We will sometimes abuse the notation a little bit and write $Q = S(Q, X)$ to abbreviate $Q_{\text{next}} = S(Q_{\text{current}}, X)$. We indicate the initial iterate in the procedure as Q_0, which is independent of the shape parameters B. The final flow field is indicated as Q_*, where $Q_* = S(Q_*, X)$. We refer to the objective function as F, a function of both the flow field and the grid, and use V to refer to the value of F for given Q and X, that is, we write $V = F(Q, X)$.

With this notation in place, the following pseudocode defines a canonical procedure:

```
$X = G(B)$
$Q = Q_0$
Do until $Q$ "converged"
    $Q = S(Q, X)$
end
$V = F(Q, X)$
```

As indicated previously, to solve shape optimization problems we need to compute the derivatives of the objective function and the constraints with respect to the shape parameters. To simplify our initial applications of ADJIFOR, we have not attempted to compute derivatives for problems with constraints[4]. Hence, for our simplified problems, we require the derivative of the dependent variable V with respect to each of the components of the independent variables B.

In this paper, we use a uniform notation for derivatives. The matrix representation for the derivative linear operator for any function Z will be written J_Z. In addition, the derivative of any variable Z with respect to B will be written as Z', and the derivative of V with respect to a variable Z will be written as $\overline{Z}$. Consequently, using this notation, we wish to compute V', or equivalently, $\overline{B}$.

As a final notational convenience, we write I for any identity matrix, and $\mathbf{0}$ for the zero matrix. The dimensions of these matrices will either be obvious from context or explicitly indicated.

3 AD for Shape Optimization

The classic *forward mode* of automatic differentiation accumulates derivatives as a computation proceeds from the inputs to outputs. It follows the control flow of the original program and, for a matrix R with p columns, computes the p directional derivatives $J * R$ with a time and memory complexity that is roughly p times that of the original program, given the Jacobian J. If $R = I$, then the forward mode computes J. An alternative approach, the *reverse mode*, accumulates the derivatives in the opposite direction—from the outputs to inputs. To propagate adjoints, one must be able to reverse the flow of the program, and record or recompute any intermediate value that nonlinearly affects the final result. Once these technical difficulties are overcome, then, for a matrix L with q rows, the row linear combination $L * J$ can be computed with a time complexity that is roughly q times that of the original program. If $L = I$, then the reverse mode computes J.

Unless checkpointing [7] or additional mathematical knowledge is employed, the need to record intermediate program state makes the memory requirements of adjoint codes very high. Minimizing the storage requirements represents the most significant challenge to automatic adjoint tools.

[4]The techniques described below apply directly to constraints that are functions of Q_* and X by simply expanding F to a vector-valued function that computes the objective function and the constraints. The dimension of the derivatives expands accordingly.

Assuming that sufficient memory for the reverse mode is available, the choice of forward mode or reverse mode for computing the Jacobian depends on p, q, O_f which is the ratio of the cost of computing a column of the Jacobian to the cost of the function using the forward mode, and, O_r which is the ratio of the cost of computing a row of the Jacobian to the cost of the function using the reverse mode. If $q * O_r < p * O_f$ then reverse mode is indicated. Since the operations performed by forward and reverse mode are vector operations, O_r and O_f actually depend on q and p, respectively. O_r and O_f also depend on platform, compiler and application, as well. Typically, O_f for ADIFOR ranges from .5 to 5.0. In limited tests so far, O_r for ADJIFOR ranges from 16 to 20. Hence, the reverse mode is particularly attractive for computing the sensitivity of shape optimization problems with a large number of shape parameters and either one objective function or one objective function plus a "few" constraints. For example, if $O_r = 20$ and $O_f = .5$, reverse mode outperforms forward mode whenever $p/q > 40$, where $q = 1 + |\text{constraints}|$, and p is the number of independent design variables.

To mathematically justify our approach to derivative computation for shape optimization problems, we need to elaborate our framework. Our framework is conceptually simple, relying only on the fact that derivatives are linear functions, representable by matrices. In this view, composition of operations becomes simple matrix multiplication. To avoid index "mismatch," we view the matrices we employ as linear functions of *all* variables involved. We emphasize that this convention is solely for mathematical convenience. The implemented derivative computation does *not* form huge matrices and then multiply them together.

Using our framework, the derivative of the shape optimization problem can be written as

$$V' = L\ J_F\ J_{S_n}\ \ldots\ J_{S_1}\ J_G\ R$$

where, using the notation for linear operators introduced in Section 2, J_Z represents the derivative of a function Z evaluated at its inputs. In particular, J_{S_k} is evaluated at the input to the kth step of the flow solver. L and R are projection matrices that select the desired outputs and inputs. Similarly, L and R are block row and block column vectors, with blocks corresponding to shape parameters B, grid X, flow field Q and objective value V. We use B, X, Q, V as the canonical index ordering.

For our problem, variable V is the desired output, and B is the vector of desired inputs. The two projections are

$$L = \begin{pmatrix} \mathbf{0}_{1\times|B|} & \mathbf{0}_{1\times|X|} & \mathbf{0}_{1\times|Q|} & 1 \end{pmatrix}$$

and

$$R = \begin{pmatrix} I_{|B|\times|B|} \\ \mathbf{0}_{|X|\times|B|} \\ \mathbf{0}_{|Q|\times|B|} \\ \mathbf{0}_{1\times|B|} \end{pmatrix}.$$

As described previously, assuming sufficient storage for recording intermediate program state for each of the functions $G, S_1, \ldots, S_n, F$, reverse mode computes V'. As shown in Section 6, storing the intermediate values for all of the S_i requires a tremendous amount of storage. Fortunately, we can take advantage of mathematical properties of the flow solver to substantially reduce the storage requirements. We call this reverse mode variant the *iterated reverse mode*.

Ideally, the flow solver computes a *fixed point*, that is, it determines a flow field Q_* such that $Q_* = S(Q_*, X)$. Using the implicit function theorem, we see that

$$\begin{pmatrix} B' \\ X' \\ Q'_* \\ V' \end{pmatrix} = J_S \begin{pmatrix} B' \\ X' \\ Q'_* \\ V' \end{pmatrix}.$$

This means that $(B', X', Q'_*, V')^T$ is a fixed point of J_S. Recall that our framework expands derivative linear operators to cover *all* variables. Since S has no effect on B, X, or V, the appropriate entries in J_S will be 1.

When $\left\| J_{S_{Q_*}} \right\| < 1$, then $J_{S_{Q_*}}$ is contractive, and we can use the contraction mapping theorem to compute a fixed point of $J_{S_{Q_*}}$ through simple iteration. Flow solvers and test cases we have encountered appear to have the necessary contractive properties. Consequently, we compute V' using

$$V' = L\ J_F\ J_{S_{Q_*}}\ \ldots\ J_{S_{Q_*}}\ J_G\ R.$$

For this computation, note that the *same* operator $J_{S_{Q_*}}$ is used for each iteration. Note also that the number of iterations required for convergence of the derivatives is not necessarily n. Consequently, instead of storing n intermediate states, we only need to store one such state. Implementing the iterated reverse mode requires only two small changes (less than 10 lines of code) to the ADJIFOR-generated reverse mode code: (1) we turn off intermediate state recording for the first $n-1$ steps of S, saving only step n, and, (2) we modify the flow-solver control loop for S to repeatedly execute the reverse-mode code represented by $L * J_{S_{Q_*}}$ until the derivatives converge.

4 Description of the Codes

For the current study, G is a "home grown" wing grid generator named MYGRID, S is the stepping function in the CFL3D flow solver code, and the objective function F is taken to be the ratio of the coefficient of lift cl to the coefficient of drag cd, cl/cd. The objective function F is coded as part of CFL3D. We now describe MYGRID and CFL3D.

4.1 MYGRID Grid Generation Code

MYGRID implements a fast and simple algebraic method for generating wing grids. The grid generation code was developed (in Fortran) for use with ADIFOR and ADJIFOR studies. It is very robust in grid generation, but the code does not include many of the techniques commonly used in commercial grid generation packages to insure high quality grids.

MYGRID defines 3-D wings by a set of wing sections, using an expanded definition of the NACA four digit airfoil section family, based upon real numbers for the maximum thickness, maximum camber, and location of maximum camber, rather than the usual integer designation. This allows for easily perturbing the airfoils shapes by small amounts to facilitate the construction of finite-difference approximations to verify the results of ADIFOR and ADJIFOR. Each wing section is described by eight design parameters: xle, yle, and zle (the x, y and z leading edge coordinates), crd (the wing section chord length to trailing edge), cmx (the maximum camber line height in y-direction), xcm (the streamwise location of maximum camber height), thk (the streamwise maximum airfoil thickness), and tws (the section twist angle).

The number of design variables can be increased by simply increasing the number of wing sections that are specified. The code also allows the user to specify the number of grid points in each of the coordinate directions and a few choices that affect the grid stretching and distribution.

4.2 The CFL3D Flow Solver Code

The CFL3D (Computational Fluids Laboratory 3-Dimensional) code is a general purpose computational fluid dynamics (CFD) solver developed by Rumsey, Biedron, and Thomas of the NASA Langley Research Center, with contributions from numerous other researchers. From its inception in the early 1980's, the CFL3D code has been continuously improved, applied to a wide variety of problems, verified extensively by experiment and other CFD results, and widely distributed for use in industry[1, 5, 6, 12, 14, 13, 16, 18]. The most recent CFL3D code version 5.0, used in the current studies, includes significant new capability (sliding patched-zones for use in rotor-stator computations) and improved computational efficiency and memory usage compared to previous code versions.

The CFL3D version 5.0 code solves the time-dependent Reynolds-averaged Navier-Stokes equations in conservation form using upwind-biasing for the convective and pressure terms, and central differencing for the shear stress and heat transfer terms. The code includes the ability to solve inviscid, laminar, or turbulent flows around complex 2-D or 3-D geometries, using one of four possible grid schemes (point-matched, patched, overlapped, or embedded) provided. CFL3D includes the ability to compute steady or unsteady flows with implicit time advancement. Both multigrid and mesh sequencing techniques can be used for convergence acceleration. The code also provides numerous turbulence models, including Baldin-Lomax, Baldwin-Barth, Spalart-Allmaras, as well as several other popular models.

For simplicity, we have used a limited subset of the CFL3D code options. The techniques used to generate CFL3D adjoint code, however, are not in any way limited to these simple code options. We expect that the primary impact of additional options will be to increase the memory requirements of the adjoint code, in proportion to the added complexity of the added computation.

The CFL3D code was used to solve steady, inviscid, transonic flow around a simple 3-D transport wing. The algorithmic choices included the use of local time stepping, Roe's flux-difference splitting scheme, and scalar tridiagonal matrix inversion with smooth flux limiters. No multigrid or grid sequencing was used in these demonstrations. The grids used were all single-block, or manually decomposed into several smaller point-matched PLOT3D style grids.

5 Description of the Test Problem

The choice of a test problem for this ADJIFOR demonstration was influenced by several considerations: (1) the ADJIFOR-generated adjoint formulation is appropriate for problems with many design variables, (2) the ADJIFOR-generated adjoint code requires memory to store the intermediate computation state which depends on the size of the input grid, (3) the prospect of ADJIFOR validation by ADIFOR and finite differences, which may require time and/or storage proportional to the number of design variables, requires a small, quickly executing problem size, and, (4) the ability to easily and quickly generate a baseline and perturbed geometries within a simple Fortran code. Thus, the ideal test case is one that provides many potential design variables, within a relative simple computing context, and has moderate time and space requirements.

The current test problem is derived from a standard benchmark case for the ONERA M-6 swept and tapered wing, similar to those used on numerous commercial transport aircraft today. A 33x9x9 grid size was used for this test case. The grid size was chosen for this adjoint demonstration as a compromise between the flow modeling resolution and the in-core storage requirement for the adjoint flow solver. Initial demonstrations of the adjoint capability used the flow solver but not the grid generation package. Each of the grid x-y-z coordinates input to CFL3D was considered to be an independent variable. This yielded a potentially large number of design variables, up to a total of 33x9x9x3 = 8019, with minimal time and memory requirements for the CFD solver.

Currently, the grid and flow solver have been coupled to produce flow sensitivities with respect to the shape specification parameters (xle, yle, zle, crd, cmx, xcm, thk, and tws) at each of the input sections. The number of independent design variables can be increased by simply increasing the number of input wing sections, so long as the spanwise grid resolution is approximately the same (or more) than the number of input sections. For this test case, there are 11 input wing sections and 8 design variables per wing section for a total of 88 shape specification design variables. If many more wing sections are input than spanwise grid lines computed, the process will still work, but some of the design variables may become ineffective, due to the linear interpolation between the input wing sections; several interpolations to obtain grid lines may be possible, and the effect of a design variable at any one input section becomes less clear.

For this test case, the wing span is taken to be 1.0, the root chord is 0.6737, the tip chord is 0.3789. For simplicity of grid generation in this case, however, the usual ONERA M-6 wing section has been replaced with that of an NACA 0010 wing section. Also, the trailing edge has been fixed to close at a normalized chordwise location of 1.0, not 1.00893, as is usually the case in the NACA four digit airfoil family. The grid generation for this case was done with the MYGRID program, described previously. Since multigrid was not used in this study, converging the flow solution for this problem took about 1000 cycles.

Since the grid is so coarse, particular attention was given to the grid stretching and distribution, in order to obtain the most reasonable inviscid flow solution grid possible, within the limitations of the MYGRID grid generation package. In the streamwise coordinate direction with 33 grid points, the grid direction index starts, as is commonly done, at the lower downstream wake, wraps around the airfoil from lower trailing edge, to the leading edge, to the upper trailing edge, and continues to upper downstream wake. Grid clustering has been provided near the wing leading edge and trailing edge out the span. An unusually large (50%) number of grid cells were placed in the wake, distributed equally between the upper and lower wake regions, to improve the grid distribution for this coarse grid case. Both the wing normal and spanwise directions have 9 points. In the normal direction, grid points are somewhat clustered toward the airfoil surface; grid lines are normal to the airfoil surface near the leading edge, but vertical beyond the airfoil maximum thickness point and in the wake. In the spanwise direction, the grid points are equally spaced out the wing span. The wing tip has zero thickness. The outer boundary is placed at a distance somewhat larger than 5 root chords away from airfoil surface.

It is worth noting that, from our experience, the initial ADIFOR/ADJIFOR validation for accuracy, relative to carefully constructed finite-difference (FD) approximations can usually be done on such coarse grids. In the current studies, grids as coarse as 17x5x5 points were used in the initial verification of ADIFOR and ADJIFOR results. The accuracy of the ADIFOR/ADJIFOR derivatives for reasonably short runs, relative to such FD approximations, appears to be independent of the resolution of the flow field features. This is true as long the coarseness of the grid does not lead to inconsistencies in the basic flow solver algorithm. In fact, although the 17x5x5 grid is already extremely coarse, it may be possible to due these validation tests on a 9x3x3 grid, but we were unsure if that level of coarseness would violate some unknown 5-point operator assumption that may be buried deep within the CFL3D computational algorithm. There is probably no way to prove that validation can always be done on these coarse grids; it may even be possible to prove that such tests can produce misleading, or incorrect, results for some cases. However, the beauty of such coarse grid validation, if it works, is that the results can be converged quickly to machine zero, on almost any workstation, with limited time invested in producing and using “high-quality” grids.

6 Results

We now demonstrate that ADJIFOR-generated adjoint code, in conjunction with the iterated reverse mode, efficiently provides accurate derivatives with memory requirements that are compatible with the amount of memory that is typically available on modern processors – the current 33x9x9 test problems have been run on a Sun UltraSparc workstation using 128 Mbytes of RAM, and no additional disk storage.

Appendix A presents four tables that show the derivative values computed using the forward mode by ADIFOR and the iterated reverse mode by ADJIFOR for the test case described in the previous section. For forward mode, the ADIFOR-enhanced MYGRID and 1000 steps of the ADIFOR-enhanced CFL3D were executed. For iterated reverse mode AD, the ADJIFOR-enhanced MYGRID and 1000 steps of CFL3D, followed by 1000 “adjoint iterations” of CFL3D were executed. Forward mode and iterated reverse mode derivative values show excellent agreement. Finite-difference approximations (not shown in the tables) have been computed and found to agree with the ADIFOR-generated derivatives. As an additional validation of the computed derivatives, observe that computed derivative value of 0.0 for $\overline{xcm}$ is known to be correct for this case. Note that we compared the derivatives computed by the forward mode to the derivatives computed using the iterated reverse mode. In fact, the storage requirements for the standard reverse mode precluded its use.

In Tables 1 and 2, we have summarized the storage required by the standard reverse mode using ADJIFOR for the 33x9x9 test problem. In both tables, the row labeled “Control Flow” indicates the number of (4 byte) data elements that ADJIFOR saves for use in reversing the execution of the shape analysis procedure, and the row labeled ”Floating Point Values” indicates the number of (8 byte) double precision data elements that ADJIFOR saves for use in the computation of adjoint values. Obviously, the memory requirements for MYGRID are insignificant in comparison to those of CFL3D. In any case, a technique that requires 30 Gbytes of storage to compute derivatives for a 33x9x9 test problem is probably not competitive.

	number of items	storage
Control Flow	1,554,923,845	6,219 Mbytes
Floating Point Values	2,986,617,965	23,892 Mbytes
Total		30,111 Mbytes

Table 1: Memory Requirements for the Reverse Mode CFL3D

Table 3 summarizes the storage used by the iterated reverse mode with ADJIFOR for the 33x9x9 test problem. The requirements for MYGRID remain the same as shown in Table 2. Clearly, the iterated reverse mode uses *much* less space than the standard reverse mode.

Now that we have shown that iterated reverse mode provides accurate derivatives using a reasonable amount of space, we turn our attention to the issue of efficiency. Table 4 presents the timings for the

	number of items	storage
Control Flow	10,586	.04 Mbytes
Floating Point Values	24,398	.20 Mbytes
Total		.24 Mbytes

Table 2: Memory Requirements for the Reverse Mode MYGRID

	number of items	storage
Control Flow	2,562,486	10 Mbytes
Floating Point Values	3,830,415	31 Mbytes
Total		41 Mbytes

Table 3: Memory Requirements for the Iterated Reverse Mode CFL3D

original shape analysis procedure, finite differences, ADIFOR-generated forward mode derivative code, and the ADJIFOR-generated adjoint code using the iterated reverse mode for our 88 design variable test problem.

	Timings (minutes)
MYGRID+CFL3D	2.305
One-Sided Finite Differences of MYGRID+CFL3D	205.1 = 89 * 2.305
ADJIFOR of MYGRID+CFL3D	41.70

Table 4: Timing comparisons for baseline function evaluation, finite differences and the iterated reverse mode

Notice that the cost of computing all 88 derivatives using ADJIFOR is about 20% of the cost of finite differences.[5]

Finally, in Figure 1, we show the convergence behaviour for $\overline{xle}$ for 6 of the 11 wing sections. Convergence for all other design parameters and wing sections is similar. Derivatives do indeed converge, and appear to do so in fewer steps then the original flow solver.

7 Significant Results and Conclusions

The first significant result of this paper is the demonstration of the use of the ADJIFOR adjoint code generation tool for the computation of shape sensitivities for a small problem. We have verified that the derivatives for the design variables in this simple problem have been accurately computed. This gives us confidence that the ADJIFOR tool is working correctly. This is significant since the validation of the adjoint results by finite-differences and/or forward-mode differentiation will become more cumbersome as the number of design variables increases, and as the adjoint code becomes more efficient relative to the comparison methods.

The second significant result of this paper is the development of the "iterated reverse mode." The iterated reverse mode uses a well-converged solution to save the intermediate program state and then repeatedly executes the final iteration until the derivatives converge. This technique provides accurate derivatives with substantially smaller storage requirements than the standard reverse mode, and, in addition, appears to converge the derivatives in fewer iterations than were required for the solution convergence. The reduced storage requirements should make it possible to tackle much larger sensitivity and optimization problems.

The third significant result of this paper is that the ADJIFOR prototype is capable of providing shape sensitivities at a cost of about 18 function evaluations for the 88 design variable test case. Compare this

[5]We have not presented the timings for the ADIFOR-generated forward mode code here, because no effort has been made to obtain reasonable performance on that code, and would bias the comparison of reverse mode to forward mode in favor of the reverse mode. In the final paper, we will include fair timings for ADIFOR forward mode, an "Incremental Iterative" implementation of CFL3D, and finite differences using a restart file.

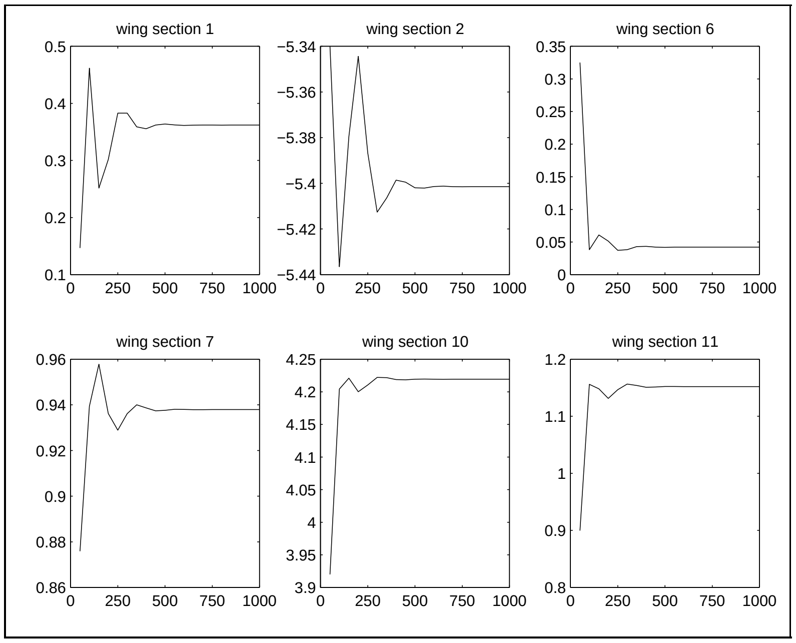

Figure 1: Iterated Reverse Mode AD Convergence for the Derivative of V with respect to xle $(\overline{xle})$

to the 89 function evaluations that would have been required for one-sided finite differences. These timing results agree with our expectations based on our experience with several other codes.

We believe that the current demonstration represents a significant advance in the ability to automatically generate sensitivity code for shape optimization. The demonstration uses a widely distributed industrial-grade aerodynamic solver. Furthermore, we have made a special effort to accommodate the aerospace industry's need for rigorous validation. We are quick to recognize, however, that more work remains to be done on several fronts to make this demonstration more realistic.

8 Remaining Work

Not surprisingly, the current small, inviscid, single-block, single-grid test problem falls far short of what is ultimately required of this adjoint demonstration, from both the points of view of industry acceptance, and the completion of NASA program milestones. In order to be complete, the adjoint generation capability and validation for accuracy must be demonstrated for laminar and turbulent flows, with other boundary condition options, and on larger, multi-block grids with grid sequencing, multigrid, and other convergence acceleration techniques. Previous experience with the forward-mode ADIFOR tool, suggests that none of these issues should pose a difficult problem for ADJIFOR. Each issue, however, must be addressed with a view toward rigorous validation for accuracy, and best-attainable efficiency. Also, the adjoint code must be demonstrated to be efficient within a realistic optimization problem, rather than just a sensitivity analysis.

From discussions with engineers doing shape optimization for a major aerospace firm in the United States, we believe that the minimum realistic target test problem for this adjoint demonstration is a wing-body configuration in inviscid flow with at least 400,000 grid points, and hundreds, to perhaps thousands, of design variables. This grid size, configuration and flow complexity, and lower bound on the number of design variables are compatible with the current state-of-the-art routinely used for shape optimization of new aircraft.

For the final version of this paper, increasing the number of design variables should pose no problem whatsoever for the adjoint method; in fact, it should only make the comparison with forward-mode differentiation and finite-differences more favorable toward the adjoint mode. The demonstration of adjoint capability for grid sequencing and multigrid solutions is also expected be included in the final version of this paper. Rather than investing immediate effort to produce laminar or turbulent solutions, we intend to focus on the solution of larger grid size problems, since problem size poses a greater challenge. We will also look at problems that have geometric and flow constraints to ascertain how this impacts the time required to compute derivatives.

It is believed that grid size can be increased with little or no performance penalty by applying ADJIFOR to an existing parallel version of CFL3D, which uses MPI/PVM message passing to distribute large problems across multiple processors. Thus, each processor will be required to solve only a small piece of a larger problem. Of course, we are assuming that a distributed problem can be converged adequately to use the iterated reverse mode technique, which also remains to be seen. We will probably include results within the final paper for a series of increasing larger grids, to see how well this method scales.

A remaining question for this work is to examine how the convergence of the shape analysis procedure affects the derivatives computed by the iterated reverse mode. For instance, convergence of the shape analysis procedure is sometimes based on convergence of the force and moment coefficients, rather than the flow field itself. This will impact the accuracy of the derivatives. We will attempt to quantify this impact.

9 Acknowledgements

The authors wish to express their sincere thanks to Dr. Tom Zang, Head of the MultiDisciplinary Optimization Branch (MDOB) at NASA Langley Research Center (LaRC), without whose support through both funding and advocacy, this project would not have been possible. The authors wish to thank Dr. Chris Rumsey of the Aerodynamic and Acoustic Methods Branch (AAMB) at NASA LaRC for the guidance he provided to the authors in the use of the CFL3D code. The authors also wish to express their thanks to Dr. Bob Biedron of AAMB at NASA LaRC for his validation efforts with the parallel CFL3D code version, which is expected to be used as the starting point in the final paper for adjoint solution of large grid problems. The authors express their thanks to Drs. Arthur Taylor and Amidu Oloso formerly of Eagle Engineering, for the development and validation of the CFL3D.ADII code for MDOB that will be used for timing and memory comparisons with the CFL3D.ADJ in the final paper. The authors express their thanks to Dr. Perry Newman of MDOB for his direction and support of MDOB research in automatic differentiation tools. The authors express their thanks to Drs. Shreekant Agrawal, Geojoe Kuruvila, Peter Hartwich, Pichuraman Sundaram, James Hager, Bob Narducci, and Eric Unger of Boeing Long Beach for their informative discussions with the authors about the status of shape optimization within the NASA High Speed Research Program. Finally, the authors express their thanks to Dr. Chris Bischof of Argonne National Laboratories for mathematical developments contributing to the ADJIFOR tool.

References

[1] R. Biedron and J. Thomas. A generalized patched-grid algorithm with application to the F-18 forebody with actuated control strake. *Computing Systems in Engineering*, 1(2–4):563–576, 1990.

[2] C. Bischof, T. Knauff, L. Green, and K. Haigler. Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation. In *5th AIAA/NASA/USAF/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 73–86, Panama City, Florida, September 7–9, 1994. AIAA-94-4261-CP.

[3] Christian Bischof, Alan Carle, George Corliss, and Andreas Griewank. ADIFOR—generating derivative codes from FORTRAN programs. *Scientific Programming*, 1:11–29, 1992.

[4] Christian Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. Adifor 2.0: Automatic differentiation of fortran 77 programs. *IEEE Computational Science and Engineering*, 3(3):18–32, Fall 1996.

[5] W. Compton, J. Thomas, W. Abeyounis, and M. Mason. Transonic Navier-Stokes solutions of three-dimensional afterbody flows. TM 4111, NASA, July 1989.

[6] F. Ghaffari, J. Luckring, J. Thomas, B. Bates, and R. Biedron. Multiblock Navier-Stokes solutions about the F/A-18 wing-lex-fuselage configuration. *Journal of Aircraft*, 30(3):293–303, 1993.

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

[8] J. Issac and R. Kapania. Aeroelastic sensitivity analysis of wings using automatic differentiation. In *6th AIAA/NASA/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 1176–1186, Bellevue, Washington, September 4–6, 1996. AIAA-96-4119-CP.

[9] V. Korivi, L. Sherman, A. Taylor, G. Hou, L. Green, and P. Newman. First- and second-order aerodynamic sensitivity derivatves via automatic differentiation with incremental iterative methods. In *5th AIAA/NASA/USAF/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 87–120, Panama City, Florida, September 7–9, 1994. AIAA-94-4262-CP.

[10] V. Korivi, A. Taylor, and P. Newman. Aerodynamic optimization studies using a 3-d supersonic Euler code with efficient calculation of sensitivity derivatives. In *5th AIAA/NASA/USAF/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 170–194, Panama City, Florida, September 7–9, 1994. AIAA-94-4270-CP.

[11] C. Moen, P. Spence, J. Meza, and T. Plantenga. Automatic differentiation for gradient-based optimization of radiatively heated microelectronics manufacturing equipment. In *6th AIAA/NASA/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 1167–1175, Bellevue, Washington, September 4–6, 1996. AIAA-96-4118-CP.

[12] C. Rumsey, R. Biedron, and J. Thomas. CFL3D: Its history and some recent applications. TM 112861, NASA, May 1997. presented at the "Godunov's Method for Gas Dynamics" Symposium, Ann Arbor, MI, May 1-2, 1997.

[13] C. Rumsey, M. Sanetrik, R. Biedron, N. Melson, and E. Parlette. Efficiency and accuracy of time-accurate turbulent Navier-Stokes computations. *Computers & Fluids*, 25(2):217–236, 1996.

[14] C. Rumsey and V. Vatsa. Comparison of the predictive capabilities of several turbulence models. *Journal of Aircraft*, 32(3):510–514, 1995.

[15] J. Su and J. Renaud. Automatic differentiation in robust optimization. In *6th AIAA/NASA/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 201–215, Bellevue, Washington, September 4–6, 1996. AIAA-96-4005-CP.

[16] J. Thomas, S. Krist, and W. Anderson. Navier-Stokes computations of vortical flows over low-aspect-ratio wings. *AIAA Journal*, 28(2):205–212, 1990.

[17] E. Unger and L. Hall. The use of automatic differentiation in an aircraft design problem. In *5th AIAA/NASA/USAF/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 64–72, Panama City, Florida, September 7–9, 1994. AIAA-94-4260-CP.

[18] V. Vatsa, J. Thomas, and B. Wedan. Navier-Stokes computations of a prolate spheroid at angle of attack. *Journal of Aircraft*, 26(11):986–993, 1989.

[19] B. Wujek and J. Renaud. Automatic differentiation for more efficient multidisciplinary design analysis and optimization. In *6th AIAA/NASA/ISSMO Symposium on multidisciplinary analysis and optimization*, pages 1151–1166, Bellevue, Washington, September 4–6, 1996. AIAA-96-4117-CP.

A Comparison of Derivative Values Computed by Forward Mode AD and Iterated Reverse Mode AD

The following tables show the derivative values computed for the test problem described in the paper. For forward mode AD, 1000 steps of the ADIFOR-enhanced CFL3D flow solver are executed. For iterated reverse mode AD, 1000 steps of CFL3D, followed by 1000 "adjoint iterations" are executed. Derivatives for cmx, xcm, thk and tws, are shown.

Wing Section	Forward Mode AD	Iterated Reverse Mode AD
1	-2.8074623368076	-2.8075015838680
2	-2.1557384226978	-2.1558060888804
3	-0.74251675378404	-0.74257164665815
4	0.49640646872707	0.49636233576429
5	1.5610312448355	1.5609958583870
6	3.8498180547396	3.8497831660147
7	4.2767317665527	4.2767127609971
8	4.8447400636135	4.8447266302691
9	5.3210006854931	5.3209919040510
10	5.7055136321912	5.7055085823427
11	1.3168001900473	1.3168022914470

Table 5: Derivative of V with respect to cmx ($\overline{cmx}$.)

Wing Section	Forward Mode AD	Iterated Reverse Mode AD
1	0.0	0.0
$\vdots$	$\vdots$	$\vdots$
11	0.0	0.0

Table 6: Derivative of V with respect to xcm ($\overline{xcm}$.)

Wing Section	Forward Mode AD	Iterated Reverse Mode AD
1	-9.5695837013795	-9.5695986673453
2	-14.251867997215	-14.251891341742
3	-14.141176854928	-14.141198769129
4	-13.723488012199	-13.723508628008
5	-12.998801469027	-12.998820918380
6	-15.980831897981	-15.980854638019
7	-10.774052769313	-10.774065856847
8	-10.637164833371	-10.637174862534
9	-9.6854245878471	-9.6854312280733
10	-7.9188320327403	-7.9188349534652
11	-0.35577690671199	-0.35577409729939

Table 7: Derivative of V with respect to thk ($\overline{thk}$.)

Wing Section	Forward Mode AD	Iterated Reverse Mode AD
1	-8.9783175395046E-02	-8.9783304818605E-02
2	-0.13777493172478	-0.13777513263480
3	-0.14033316309455	-0.14033340058874
4	-0.14268876834394	-0.14268902700298
5	-0.14484174747294	-0.14484201187753
6	-0.19469374919372	-0.19469415020925
7	-0.14544334149336	-0.14544361023223
8	-0.13203643206894	-0.13203667493141
9	-0.13562460240379	-0.13562481525497
10	-0.15620785249790	-0.15620803120294
11	-2.2707098520427E-02	-2.2707403366190E-02

Table 8: Derivative of V with respect to tws ($\overline{tws}$.)