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

Alan Carle
Rice University
Department of Computational & Applied Mathematics

Mike Fagan
Rice University
Department of Computer Science

Lawrence L. Green
NASA Langley Research Center
Multidisciplinary Optimization Branch

Outline

The Goal - Derivatives for Shape Optimization

Automatic Differentiation

Shape Optimization and the Iterated Reverse Mode

MYGRID and CFL3D

Results

Conclusions

Future Work

Thanks to NASA, NSF/CRPC and Boeing
for supporting this effort.

The Goal

Compute derivatives for aerodynamic shape optimization fast, accurately, and with little human-intervention.

Develop the ADJIFOR automatic adjoint code generation tool for Fortran 77 and apply to a production CFD code.

Demonstrate on model problems (Rice) and simple aerodynamic shape optimization problems (Langley).

Transition to industry design for use on design problems.

Automatic Differentiation

AD is a chain-rule based technique for computing derivatives of functions described by computer programs.

Given F: $R^n \rightarrow R^m$ which takes T(F) time and M(F) memory to evaluate, and matrices R: n x p and L: q x m ...

AD forward mode computes J_F * R using time O(p * T(F)) and memory O(p * M(F)) by computing the <u>gradient</u> of each intermediate value during a forward execution of the function code. Good for small p and large m.

AD reverse mode computes L * J_F using time O(q * T(F)) and memory O(T(F) + q * M(F)) by computing <u>adjoints</u> of each intermediate value during a reverse execution of the function code. Very good for q = 1, and large n.

The Forward Mode

Associate a “gradient” **g_** with every program variable and apply the rules of differential calculus**.**

s = f(v,w) ➔ g_s += ds/dv * g_v + ds/dw * g_w

```
t := 1.0
do i = 1, n
    if (x(i) > 0) then
        t := t*x(i)
    endif
endif
```

➔

```
g_t:=0.0
t:=1.0
do i = 1, n
    if (x(i) > 0) then
        g_t:= x(i) * g_t + t *g_x(i)
        t := t*x(i)
    endif
endif
```

Initializing **g**_x(i) = e_i ➔ **g**_t = dt/dx(1:n).

The Reverse Mode

Associate an adjoint α with every program variable and apply the following rule to the inverted program:

s = f(v,w) ➔

α_**v = α_v + ds/dv * α_s**
α_**w = α_w + ds/dw * α_s;**
α_**s = 0**

Step 1: Forward Computation

```
LOG t
t = 1.0
do i = 1, n
      jump(i) = x(i) > 0
      if (jump(i)) then
            LOG t
            tmp = t * x(i)
            t = tmp
      endif
enddo
```

Step 2: Adjoint Computation

α_**t = 1.0,all other α_*=0**

```
do i = n, 1, -1
      if (jump(i)) then
            α_tmp = a_t; α_t = 0.0
            UNLOG  t
            α_x(i) = α_x(i)  + t * α_tmp
            α_t     = α_t + x(i) * α_tmp
            α_tmp = 0.0
      endif
enddo
UNLOG t
```

The Canonical Shape Optimization Problem

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

G - grid generator

S - flow field stepping function

F - objective function

B - shape parameters

X - grid

Q - flow field

V - objective function value

We assume that the only derivatives we need are dV/dB.

How to compute dV/dB

Let n be the number of times the stepping function S is executed, and apply the chain rule to get:

$$dV/dB = L\ J_F\ J_{Sn} \ldots J_{S1}\ J_G\ R$$

...

objective function

iterating stepping function

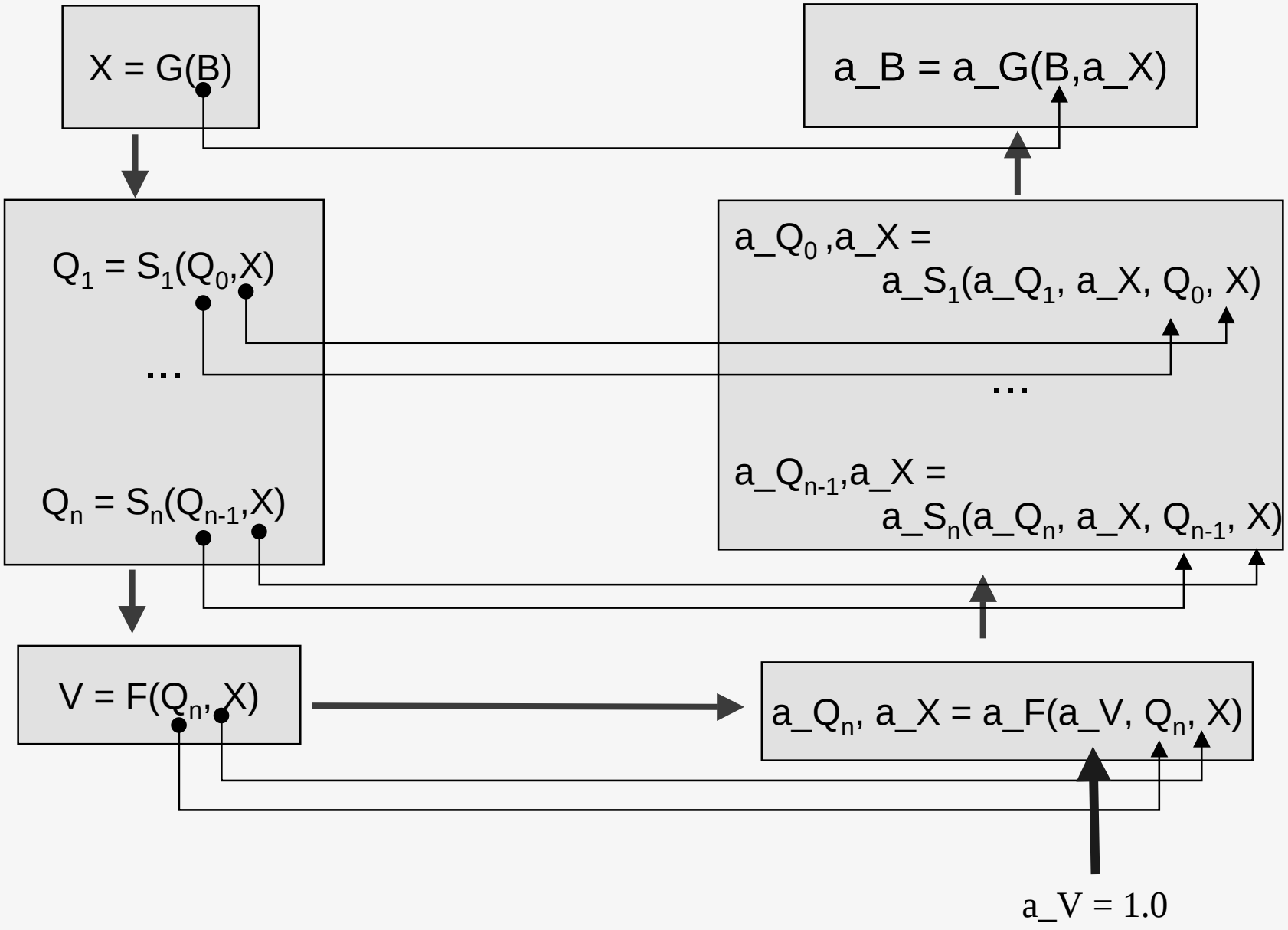
grid generation

L and R are block matrices that identify the dependent and independent variables.

All derivatives are wrt to all program variables in the canonical order: shape parameters B, grid X, flow field Q, and objective value V.

This is correct, but infeasible due to logging requirements of the reverse mode.

Reverse Mode Code Coupling



The Iterated Reverse Mode (IRM) - A better way to compute dV/dB

If $Q^* = Sn(Q^*, X)$ then

$$dV/dB = L\ J_F\ J_{Sn} \ldots J_{S1}\ J_G\ R$$

Adjoint code must log G, Sn, …, S1, and F.

$$= L\ J_F\ J_{Sn} \ldots J_{Sn}\ J_{Sn-1} \ldots J_{S1}\ J_G\ R$$

Implicit function theorem ensures that $J_{Sn-1} \ldots J_{S1}$ is a fixed point of J_{Sn}

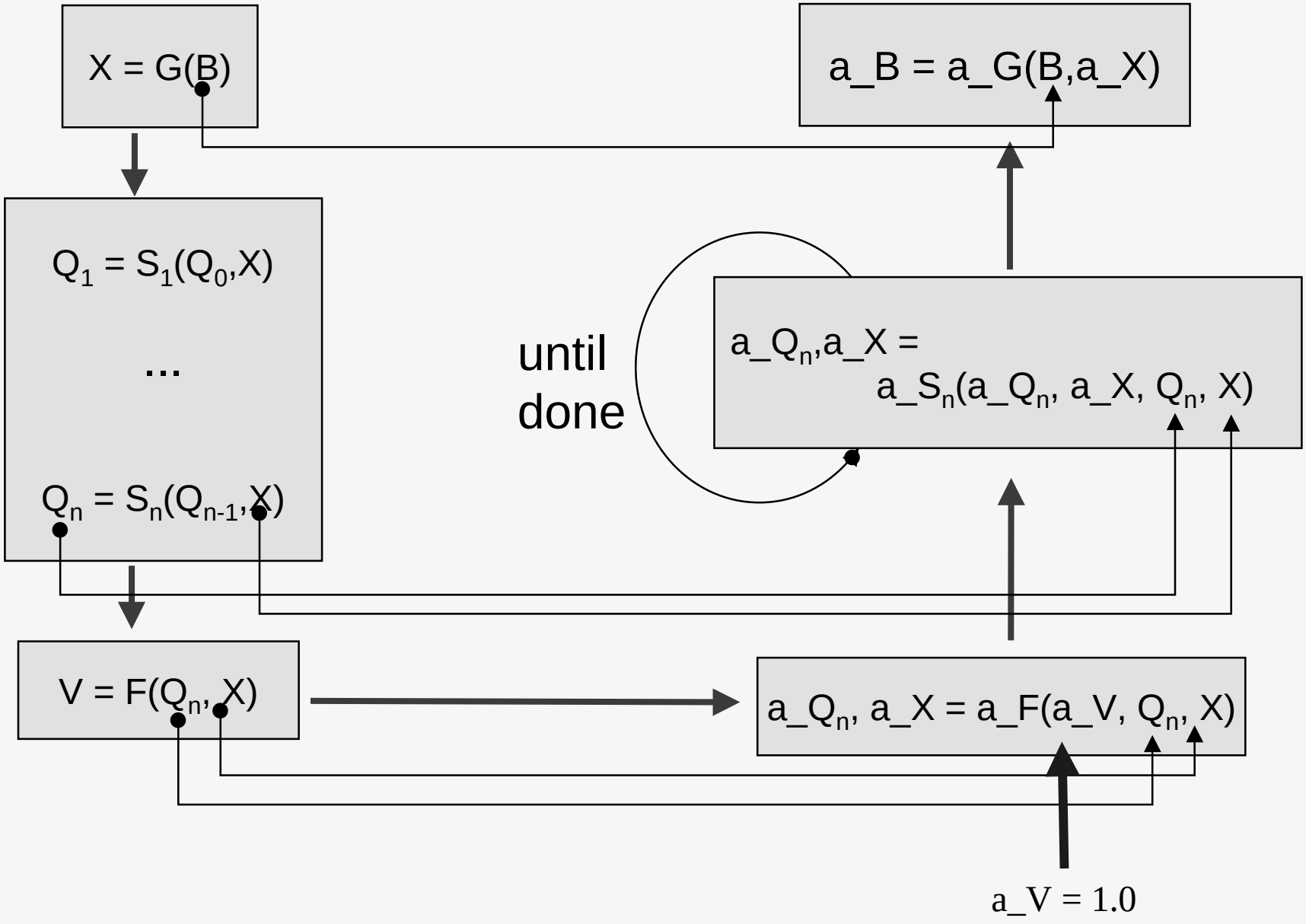
$$= L\ J_F\ J_{Sn} \ldots J_{Sn}\ J_G\ R$$

Adjoint code must log G, Sn, and F, only.

Contractive mapping theorem ensures that a fixed point of J_{Sn} can be computed by raising J_{Sn} to successive powers.

Caveats…

IRM Code Coupling



Geometry and Grid Generation

MYGRID wing grid generator

- Simple, algebraic, used for ADIFOR and ADJIFOR studies
- Defines 3-D wings by set of wing sections, 8 DV per section (xle, yle, zle, crd, cmx, xcm, thk, tws)
- Generates single-block grids

SPLITTER

- Single-block grid split into multiple-block grid by code written by Biedron (NASA LaRC) Splits CFL3D input file as well!

Test cases

- 11 wing sections for total of 88 DV
- volume grid sizes: 17x5x5, 33x9x9, 65x17x17, 129 x 65 x 33,
  - split into 1, 2, 4 and 8 zones

Computational Fluid Dynamics

CFL3D

- Code by Thomas, Rumsey and Biedron of NASA LaRC
- Solves Euler/Navier-Stokes equations in convervation form
- Numerous grid, solver, and convergence acceleration options
- Sequential (CFL3D 5.0) and MPI parallel (CFL3D 4.1) code versions

Objective Function - ratio of lift-to-drag

Test case

- steady, inviscid, transonic flow around 3-D wing using point-mached grids without multigrid (M = .84, alpha = 3.06°)

Generating derivative code with ADJIFOR

1. Prepare source code.

   CFL3D had been processed with ADIFOR before, so no preparation was required.

2. Process with ADJIFOR.

   108 minutes to generate code on SPARC Ultra 1 Wkstn
   54k lines of orig. code $\rightarrow$ 212k lines of fwd and rev. code

3. Modify the CFL3D stepping loop to form the IRM.

   We log 4 steps of S to disk and iterate the 3rd step. We reduce I/O costs by reading the log for each of the adjoint steps of S into memory before executing the step.

4. Compile and link with ADJIFOR-MPI.lib and Tape.lib.

5. Connect MYGRID, SPLITTER, CFL3D, a_CFL3D, a_SPLITTER and a_MYGRID.

6. Run and verify answers against finite differences and ADIFOR.

Derivatives of V wrt tws.
CFL3D 5.0.

Wing Section	ADIFOR Forward Mode	ADJIFOR Iterated Reverse Mode
1	-9.3596682272D-02	-9.3596521007D-02
2	-0.14369275653618	-0.14369243579322
3	-0.14565042805518	-0.14565008124860
4	-0.14780920646490	-0.14780883618091
5	-0.15016909176536	-0.15016870059014
6	-0.20168090120663	-0.20168033692085
7	-0.15104420505533	-0.15104381205916
8	-0.13700349641320	-0.13700314053909
9	0.14102086061985	-0.14102054023426
10	-0.16309629767528	-0.16309601114465
11	-8.2278444748D-02	-8.2278045874D-02

Derivatives of V wrt tws.
CFL3D 4.1.

Wing Section	ADIFOR Forward Mode	ADJIFOR Iterated Reverse Mode
1	-8.9783175395E-02	-8.9783304818E-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	-2.2707098520E-02	-2.2707403366E-02

Timing comparisons for CFL3D 4.1 on the IBM SP.

	33 x 9 x 9				65 x 17 x 17			
# zones	1	2	4	8	1	2	4	8
Function Time	141	107	88	71	1385	1070	682	336
Iterated Reverse Mode Time	1447	870	625	426	11697	7606	4634	2568
Ratio of Iterated Reverse Mode to Function Time	10.2	8.13	7.10	6.00	8.45	7.10	6.79	7.64

Tests used either 1 or 2 “4-way SMP nodes.”

Timings comparisons for CFL3D 5.0 on the IBM SP.

	Timings (seconds)	Ratio to Function
Function	132	1
One-sided FD	117E+2	89
Forward Mode	530E+2	402
Iterated Reverse Mode	146E+1	11.1

Storage comparisons for CFL3D 4.1 on the IBM SMP (per compute node).

Test Case	33 x 9 x 9				65 x 17 x 17			
# zones	1	2	4	8	1	2	4	8
# pts	2673	1377	765	425	18785	9537	5049	2673
IRM Dyn Mem	21M	11M	6M	3.4M	150M	75M	40M	21M
IRM Disk	84M	44M	25M	14M	601M	304M	165M	90M
IRM Dyn Mem (bytes/pt)	7776	7745	7875	7965	7885	7824	7923	7950
IRM Disk (bytes/pt)	31598	31683	32518	33918	31969	31907	32610	33659

Does it scale?

	Required Dynamic Memory (bytes/pt)	Required Disk (bytes/pt)
No multigrid, real*8	8000	32000
No multigrid, real*4	4000	16000
Multigrid, real *8	16000 + …	64000 + …
Multigrid, real*4	8000 + …	32000 + …

These estimates suggest that we could run a 400,000 grid point problem without multigrid, real*8, on 32 procs, each having 128 Mbytes memory and 400 Mbytes disk.

L. Green has run a 129x65x33 case, with multigrid, real*8, on 32 procs of the NAS Origin 2K.

dyn. mem. $\rightarrow$ 167M per proc
disk $\rightarrow$ 646M per proc
T(Adj)/T(F) $\rightarrow$ 15

Conclusions

We have demonstrated that ADJIFOR-generated code for MYGRID and CFL3D produces

- correct derivatives,
- at a cost ranging from 7 to 21 function evaluations independent of the number of shape parameters, and,
- with minimal human effort.

Although ADJIFOR-generated code uses extensive amounts of memory and disk, the requirements are consistent with the resources available on today's parallel hardware.

Future Work

Full Navier-Stokes including all turbulence models , and other advanced CFL3D features

Reduce memory requirements of generated derivative code through improved program analysis - linearity analysis, recomputation of function values

Performance tuning of generated derivative code - goal is O(Adj)/O(F) = 5

Release of ADJIFOR automatic adjoint code generator to public - if interested, contact **carle@rice.edu.**