

Intraprocedural Adjoint Code by the NAGWare Fortran Compiler and Prospects for Second Derivatives

Michael Maier

Bruce Christianson
Uwe Naumann

Dmitrij Gendler

AD-EU-2006/05

Contents

- 1 Introduction
- 2 CompAD-II History
- 3 Implementation
- 4 Technical Insight
- 5 Benchmark
- 6 Publication
- 7 Ongoing & Future Work

Introduction

- CompAD-I
 - EPSRC GR/R5525
 - 1 man year
 - Assessment:
Tending to Outstanding
- Upcoming (hopefully) CompAD-II
 - 4 man years
 - Refereeing:
 - 1× *Good*
 - 2× *Outstanding*
 - ⇒ 3× *Should Proceed*

Thank you for support!

CompAD-II History

- **Prototype 1: Forward Mode**

- using overloading mechanism



J. RIEHME, M. COHEN and U. NAUMANN: *Towards Differentiation-Enabled Fortran 95 Compiler Technology*.

Proceedings of the 2003 ACM Symposium on Applied Computing. 2003.

- **Prototype 2: Forward Mode**

- preaccumulation on statement level



U. NAUMANN and J. RIEHME: *A Differentiation-Enabled Fortran 95 Compiler*.

ACM Transactions on Mathematical Software, vol. 31, pages 1–16. 2005.

CompAD-II History

- **Prototype 3: Reverse Mode**

- using interpretation of tape



U. NAUMANN and J. RIEHME: *Computing Adjoints with the NAGWare Fortran 95 Compiler*.

H. BÜCKER, G. CORLISS, P. HOVLAND, U. NAUMANN, B. NORRIS, editors: *Automatic Differentiation: Applications, Theory, and Tools, Lecture Notes in Computational Science and Engineering*, vol. 50. Springer, 2005.

Data Reversal: Mapped vs. Real Memory

- Mapped Memory

- hash addresses of active variables
- tape partial derivatives
- interpreter

⇒ CompAD-I Prototype 3: Interpretation of Tape

- Real Memory

- association of variables by address
- augmentation of memory with adjoint memory
- no interpreter needed



M. MAIER and U. NAUMANN: *Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler.*
To appear in *Proceedings of ECT 2006.*

Implementation Status

- currently restricted to *intraprocedural* level
- does control-flow reversal (loops, branches)
- does data-flow reversal
- supports arrays
- supports a subset of Fortran operators and functions:
+, -, *, /, **, sin, cos, exp, sqrt

Data Flow & Control Flow Reversal

- 3 types of stacks:
 - ① control flow stack
 - loop iteration counters
 - conditional branch decision flags
 - ② data flow stack:
 - overwritten values are stored during augmented forward sweep
 - stack per data type (reals, integers, ...)
 - ③ vector index stack
 - stores non-constant vector indices used in augmented forward sweep

Example Fortran and C Output

Function:

$$\sigma : \mathbb{R}^s \rightarrow \mathbb{R} : \mathbf{a} \mapsto a_1 \cdot a_2 \cdots a_s = c$$

\mathbf{a} : Array of DOUBLE PRECISION values.

s : Size of array \mathbf{a} .

c : Result value of calculation.

Fortran Example:

```

1 DO i = 1, s
2
3   c = c * a(i)
4 ENDDO
```

C Output:

```

for (i=1; i<=*s; i++)
{
  *c = *c * *a[φ(i)];
}
```

$*a[\varphi(i)] \equiv *((\text{Double } *) (\text{a.addr}$
 $\rightsquigarrow + (\text{a.dim}[0].\text{mult}*(i) + \text{a.offset})))$

Augmented Version

Fortran Example:

C Output:

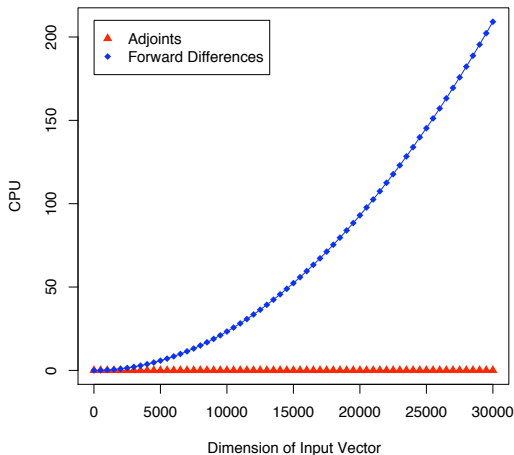
1	LOOPBD (f, 1, s, 1)	loopbd (&f, 1, s, 1);
2	DO i = 1, s	for (i=1; i<=s; i++) {
3	PUSH (b0)	push_dp (&b0);
4	PUSHINDEX (i)	push_index (&i);
5	b0 = a(i)	b0 = *a[φ(i)];
6	PUSH (b1)	push_dp (&b1);
7	b1 = c	b1 = *c;
8	PUSH (b2)	push_dp (&b2);
9	b2 = b1 * b0	b2 = b1 * b0;
10	PUSH (c)	push_dp (c);
11	c = b2	*c = b2;
12	ENDDO	}
13	PUSHCONTROL (f)	push_control (&f);

Adjoint Reverse Sweep

	Fortran Example:	C Output:
1	POPCONTROL (f)	pop_control (&f);
2	DO i = 1, f	for (i=1; i<=f; i++) {
3	POP (c)	pop_dp (c);
4	B2 = C; C=0	B2 = *C; *C=0;
5	POP (b2)	pop_dp (&b2);
6	B1 = B2 * b0	B1 = B2 * b0;
7	B0 = B2 * b1	B0 = B2 * b1;
8	POP (b1)	pop_dp (&b1);
9	C = C + B1; B1=0	*C += B1; B1=0;
10	POPINDEX (k)	pop_index (&k);
11	POP (b0)	pop_dp (&b0);
12	A(k)=A(k)+B0; B0=0	*A[$\varphi(k)$] += B0; B0=0;
13	ENDDO	}

Adjoint vs. Finite Differences

- Griewank Global Optimization Problem
 - $y = f(\mathbf{x}) = \sum_{i=1}^n \frac{x_i^2}{400} - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
 - variable input vector dimension, scalar output



Publication

- Published Technical Report AIB-2006-03
 - covers generation of intraprocedural adjoint code
 - theoretical background
 - examples of code transformation
 - details on data- and control-flow reversal
 - software development issues
 - case studies



M. MAIER and U. NAUMANN: *Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler.*

To appear in *Proceedings of ECT 2006.*

Also available under AIB-2006-03, visit:

<http://aib.informatik.rwth-aachen.de/>

Ongoing Work

- Duplication of Subroutines
 - ① subroutine 1: augmented forward code only
 - ② subroutine 2: reverse code only
⇒ *interprocedural* call graph reversal
 - ③ original code left intact for later optimization & analysis
- transformation of program code:
automatic replacement of calls to transformed subroutines with calls of their augmented and reverse parts
- dynamic stacks per subroutine:
control & data flow, vector indices

Future Work

- finish *interprocedural* code generation
- optimization of memory consumption
- support for not well-structured code
- optimization of compiler API