

# Status Report: Adjoint Code with the NAGWare Fortran Compiler v5.1

Michael Maier

Department of Computer Science  
RWTH Aachen University  
Germany

`maier@stce.rwth-aachen.de`

5<sup>th</sup> European AD Workshop 2007-05

# Contents

- ① History & Overview
- ② Source Transformation Updates
- ③ Merge
- ④ Outlook

# History & Overview

- **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.

# History & Overview

- **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.

## History & Overview

### Prototype 4: Reverse mode through direct PT transformations:

- based on NAGWare F95 compiler sources v4.3
- own C++ object-oriented interface to the compiler API
- association by *name*
- *intraprocedural* transformation of subroutines
- does control-flow reversal (loops, branches)
- does data-flow reversal
- supports basic subset of Fortran operators and functions:
  - +, -, \*, /, \*\*,
  - sin, cos, exp, sqrt,
  - pass-thru for data conversion methods, e.g. dble
- supports vectors

## History &amp; Overview

## Prototype 4: Reverse mode example:

	<b>Input Code</b> →	<b>Transformed AD</b> (unparsed PT)
1	<code>c = SIN(a)</code>	<code>CALL DATAPUSH(dummy)</code>
2		<code>dummy = SIN(a)</code>
3		<code>CALL DATAPUSH(c)</code>
4		<code>c = dummy</code>
5		<code>CALL DATAPOP(c)</code>
6		<code>dummy_adj = dummy_adj + c_adj</code>
7		<code>c_adj = 0</code>
8		<code>CALL DATAPOP(dummy)</code>
9		<code>a_adj = a_adj &amp;</code>
.		<code>&amp; + dummy_adj * COS(a)</code>
10		<code>dummy_adj = 0</code>

## History & Overview

- Published in *Proceedings of ECT 2006*:
  - 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*. In G. M. B.H.V. Topping and R. Montenegro, editors, *Proceedings of Fifth International Conference on Engineering Computational Technology*, paper 112. Civil-Comp Press, Kippen, Stirlingshire, UK, 2006. Also available under AIB-2006-03, visit: <http://aib.informatik.rwth-aachen.de/>

# Migration to Compiler Sources

## v5.1

- resolved several problems due to migration, including:
  - adjusting insertion of new symbols
  - adjusting resolution of generic calls
  - adjusting insertion of intrinsic function
  - setting internal flags
- re-engineered & optimised C++ interface  
→ optimised transformation algorithms
- restored functionality as described in ECT2006 paper

## Association by Address

- replaced *association by name* by *association by address*:
  - introduced user type `COMPAD_TYPE` (via module), having (at least) one VAL and DRV component
    - variable activation
  - operations on active variables are replaced by operations on their VAL component
  - adjoint code assigns sensitivities to DRV components
- advantages:
  - users can write own `COMPAD_TYPE` implementation with special semantics
  - supplied *pure* `COMPAD_TYPE` implements VAL and DRV components of type `DOUBLE PRECISION`
    - reverse AD without any overhead
  - supplied forward mode `COMPAD_TYPE` using operator overloading enables direct computation of second-order derivatives

## Association by Address

## Association by Name (Review):

	<b>Input Code</b> →	<b>Transformed AD</b> (unparsed PT)
1	<b>c = SIN(a)</b>	<b>CALL</b> DATAPUSH(dummy)
2		dummy = <b>SIN(a)</b>
3		<b>CALL</b> DATAPUSH(c)
4		c = dummy
5		<b>CALL</b> DATAPOP(c)
6		dummy_adj = dummy_adj + c_adj
7		c_adj = 0
8		<b>CALL</b> DATAPOP(dummy)
9		a_adj = a_adj &
.		& + dummy_adj * <b>COS(a)</b>
10		dummy_adj = 0

## Association by Address

## Example:

	<b>Input Code</b> →	<b>Transformed AD</b> (unparsed PT)
1	<b>c = SIN(a)</b>	<b>CALL</b> DATAPUSH(dummy%val)
2		dummy%val = <b>SIN</b> (a%val)
3		<b>CALL</b> DATAPUSH(c%val)
4		c%val = dummy%val
5		<b>CALL</b> DATAPOP(c%val)
6		dummy%drv = dummy%drv + c%drv
7		c%drv = 0
8		<b>CALL</b> DATAPOP(dummy%val)
9		a%drv = a%drv &
.		& + dummy%drv * <b>COS</b> (a%val)
10		dummy%drv = 0

# Merge

- **branch 1:**

AD transformation of parse tree (as described)

- **branch 2:**

overloading module & automatic variable activation



→ cf. talk: J. RIEHME: *Second-Order Adjoints with the NAGWare Fortran 95 Compiler.*

*5<sup>th</sup> European Workshop on Automatic Differentiation,*  
University of Hertfordshire, UK, May 22nd, 2005.

- **merged branches**

- automatic activation of variables
- automatic use of modules
- robust overloading
- extensive test suite from 2nd branch
- second-order adjoints

# Outlook

- focus on **checkpointing**
- *interprocedural* source transformations
- currently: copy on definition (overwrite)  
→ in some cases copy on use?
- generic COMPAD\_TYPE layout  
→ no restrictions on names of component etc.
- extensive tests
- covering the Fortran Standard in source transformation