CompAD-II & CompAD-III
The AD-Enabled NAGWare Fortran 95 compiler

Uwe Naumann, Michael Maier
RWTH Aachen, Germany

Bruce Christiansson, Dimitij Gendler, Jan Riehme
Department of Computer Science, University of Hertfordshire, UK
www.stce.rwth-aachen.de, riehme@stce.rwth-aachen.de

AD Workshop, Oxford, Nov. 24-25, 2008
Longterm project to put current (or new) AD-technology into an industrial strength compiler

Funded by EPSRC

Prototype implementation, full language support beyond scope
- Target application: Multi grid flow solver, F77, QinetiQ

Early application to other codes, exploiting the hybrid approach:
- Source transformation and overloading techniques in one AD-compiler
CompAD - Overview

CompAD - I — Jul 2001 - Dec 2002
- Tangent Linear mode source trafo (TLM)

- Adjoint source trafo
- Hybrid: Applications
- AD of assembler code (ADAC)
- F95 - Inliner

CompAD - III — Jan 2009 - Dec 2010 (estimated)
- Efficiency
  - Automatic call graph reversal
  - Applications
  - Automatic loop-level checkpointing
  - Adol-C / ADIC on unparsed output of the compiler

Naumann, Christianson, Maier, Gendler, Rieh
CompAD-II : Concepts
CompAD -II : Concepts : Hybrid approach

Overloading part:

- Provide a COMPAD_MODULE implementing a COMPAD_TYPE with overloaded operators/intrinsics
- Parser: Automatic type changes: \( \text{REAL} \rightarrow \text{COMPAD_TYPE} \)
- Semantic/global analysis: Overloading of operators/intrinsics for COMPAD_TYPE
```plaintext
subroutine F(x, y)
  double precision :: x, y
  y = sin(x)
end subroutine
```

```
|ROOT
\-S_SUBP
  \-SUBROUTINE
    \-NAME=F
    \-LIST
      \-NAME=X
      \-NAME=Y
    \-TYPE
      \-DOUBLE_P
      \-NAME=X
      \-NAME=Y
    \ASGN
      \-NAME=Y
    \-FUNREF
      \-NAME=SIN
      \-NAME=X
\-END
```

```
 Naumann, Christianson, Maier, Gendler, Rieh
CompAD-II & CompAD-III
AD Workshop, Oxford
8 / 31
```
Overloading part:
- Provide a COMPAD_MODULE implementing a COMPAD_TYPE with overloaded operators/intrinsics
- Parser: Automatic type changes REAL into COMPAD_TYPE
- Semantic/global analysis: Overloading of operators/intrinsics for COMPAD_TYPE

Source transformation part: Generate in addition
- passive evaluation code
- passive evaluation code with result checkpoints (statement wise)
- Single assignment code (code list, SAC)
- Adjoint code build from SAC
- Code for store/restore subroutine argument checkpoints
  - Global variables: user support via pragmas
- Joint mode reversals
subroutine F(x,y)
  double precision :: x, y
  y=sin(x)
end subroutine

|IF |
\-OPERATOR .EQ.
\_-COMPAD_CALL_MODE
\_-COMPAD_EVAL ONLY
\-ASGN
|NAME=Y
\-COMPONENT
|NAME=X
\-NAME=VAL
\-FUNREF
|NAME=SIN
\-COMPONENT
|NAME=X
\-NAME=VAL
\-ELSEIF
\-OPERATOR .EQ.
\_-COMPAD_CALL_MODE
\_-COMPAD_STORE_ARGS
\-END
AD - mode selection:

- **ad**
  Generate Adjoint code (source transformation)

- **ad_scalar**
  Generate Adjoint code, special scalar mode (source transformation)

- **ad_secorder**
  Generate Adjoint code (source transformation)
  Compute second order Adjoints via Overloading (TLM)

- **ad_adj_tape**
  Compute Adjoints via tape generation by overloaded operators

- **ad_adj_tape_secorder**
  Compute Adjoints via tape generation by overloaded operators
  Second order Adjoints via overloaded tapeinterpreter

- **ad_tlm**
  Compute Sensitivities via tangent-linear mode by ovl. operators

- **ad_tlm_scalar**
  Compute Sensitivities via scalar tangent-linear mode by ovl. ops

Other options:

- **ad_driver**
  Compile driver programm (not differentiated)

AD mode selection required too!!

- **help | --help**
  Show usage information
CompAD - Limitations

Adjoint code generation:
- F77 only (target application)
- Covering more language features is a question of manpower

Tape based Adjoints:
- Writing into a global tape is a side effect not allowed in pure routines
- Thus overloaded operators are not elemental
- Therefore FORALL and WHERE statements must be rewritten

General problem: I/O
- Automatic type changes activates all I/O -statements
- Preprocessor macros to access the value component of COMPAD_TYPE
NAGWare Fortran 95 based inliner

- Main developer: Micheal Maier
- Based on the NAGWare Fortran 95 compiler
- Overloaded arithmetic in F95: significant overhead by calling subroutines instead of processor intrinsic operators

Intermediate Data Generation

- Compilation of the module to be inlined in collector mode
- Generates an InterStorage Module file (ISM)
  - XML-based code prepresentation

Propagation

- Compile the code that uses the module
- Replace calls of overloaded operators with parse tree segments derived from the ISM representation of the operator:
  - Generate a temporary variable
  - Replace arguments of ISM code with actual arguments, ...

Inliner recursion level

- Reapply inlining for recursive operators / functions
Inliner : Applications

Inlined TLM for Sisyphe v5,4
- Compared to overloading: 5-6 times faster

Inlined TLM for multi-grid flow solver
- Compared to overloading: 3-4 times faster

Inlined McCormick Relaxation
- libMC (www.yoric.mit.edu/libMC/), P.I.Barton, B.Chachuat, A.Mitsos
  Computes convex/concave relaxations of factorable functions as well as subgradients of these relaxations, recursively
- F95 re-implementation of libMCF95: mccormick_module

<table>
<thead>
<tr>
<th>inliner, rec. level</th>
<th>none</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>compilation time</td>
<td>0.64s</td>
<td>5.93s</td>
<td>43.60s</td>
<td>26.74s</td>
<td>1652.94s</td>
</tr>
<tr>
<td>segments inlined</td>
<td>n/a</td>
<td>61</td>
<td>270</td>
<td>1453</td>
<td>8527</td>
</tr>
<tr>
<td>binary size KiB</td>
<td>453</td>
<td>450</td>
<td>714</td>
<td>2158</td>
<td>10126</td>
</tr>
<tr>
<td>runtime 1k loops</td>
<td>1.14s</td>
<td>0.63s</td>
<td>0.44s</td>
<td>0.35s</td>
<td>0.35s</td>
</tr>
</tbody>
</table>
ADAC - Automatic Differentiation of Assembler Code

- Main developer: Dimitrij Gendler
- Standalone tool, based on a free available x86 assembler grammar
- TLM only
- Research prototype, limit set of instructions
  - but enough for TLM of the QinetiQ multi-grid flow solver
- Besides the asm-source code a detailed configuration file is required
  - no variables names $\Rightarrow$ description of subroutine arguments needed
- Successful applied to
  - asm-code of the flow solver generated by the NAGWare F95 compiler
  - asm code of adjoint code of some MINPACK-II problems generated by the AD-enabled NAGWare F95 compiler

Naumann, Christianson, Maier, Gendler, Rieh
CompAD-II & CompAD-III
AD Workshop, Oxford
Adjoints for a multi-grid Euler solver, QinetiQ, UK

- Fortran 77, 2500 lines of code
- Subroutines have no arguments, all data transfer via COMMON blocks
- COMMON block was transformed into a F95 module
- Joint reversals, all (global) variables in argument checkpoints
  - room for optimisation by using pragmas
- arithmetic if-constructs had to be replaced
- Final test phase

Also done:
- TLM by Overloading / Inlining / ADAC
- Tape based adjoints
Adjoints for Time-Dependent Optimal Control

- CNS - Optimal temperature control in plasma spraying
- Cooperation with Jörg Stiller, Andrea Walther, Dresden University of Technology, Germany
- Heavy F95 (userdefined types with overloaded operators, modules, ..)
- TVD - Runge - Kutta method with time and space discretization
- Tape based Adjoints for one time step
- Adjoint of the complete evolution via optimal checkpointing
  - coupling with REVOLVE, Andrea Walther
- Already a simple steepest descent to solve optimal control problem gave promising results
Cooperation with Rebekka Kopmann, Federal Waterways Engineering and Research Institute, Karlsruhe, Germany

Heavy F95 (modules, pointer, userdefined datatypes)

100K lines of code

Serial version only

Dune of some material moves on ground due to flowing water

Sensitivities of the surface of the dune with respect to some parameters of the material TLM by overloading / inlining (factor 5-6)

Currently:

Transition to new version

TLM of parallel version
ICON, 2d shallow water model

- Cooperation with Peter Korn, Florian Rauser, MPI-M, Hamburg, Germany
- Heavy F95 code
- 30K lines of code
- Tape based adjoints
- Used to compute error estimations
- Checkpointing of time loop in preparation [REVOLVE, Andrea Walther]
Sensitivities for a Truncated Newton Solver

- `optnhp.for`, OPTIMA package, Hatfield Polytechnic, Numerical optimisation centre
- Compiler generates code to compute gradient and Hessain vector products of the target function
- Overloaded TLM of compiler generated Adjoint code
- Tape based Adjoint, overloaded tape interpreter
Sensitivities for the NAG Library

Prototype implementation:

- New implementation wrapper routines for E04KYFN to compute Adjoints and tangents for user provided target functions
- Complete AD business behind the scenes
- Algorithm EO4KYFN can compute exact derivatives instead of approximating with FD
- Allows new types of algorithms and solvers in the library (matrix free Newton-Krylov methods, ...)

Naumann, Christianson, Maier, Gendler, Rieh
CompAD - II : Results

CompAD - II : Results
CompAD - II : Results

AD-enabled NAGWare Fortran 95 compiler:

- Adjoint code, joint reversals
- Adjoints via tape, second order via overloading
- TLM via overloading
- Compiler coupled NAG Library
- Interface to OpenAnalysis (Michelle Strout) for compile time code analysis

NAGWare Fortran 95 compiler based inliner:

- Inlined TLM
- Inlined McCormick relaxation

ADAC – Automatic Differentiation of Assembler Code:

- TLM
- Second order by applying ADAC on adjoint code generated by the AD-enabled Nagware Fortran 95 compiler
**Theory:** 12 reviewed papers, 4 technical reports

- Optimal Jacobian accumulation is NP-complete
- Optimal data-flow reversal is NP-complete
- An L-attributed grammar for the syntax-directed compilation of tangent-linear code
- An L-attributed grammar for the syntax-directed compilation of adjoint code
- An algorithm for optimal vertex elimination for single-expression-use graphs

**Applications**

- ... 60K Euro in 2008 for STCE


Naumann: Call Graph Reversal is NP-Complete. To appear in proceedings of 5th International Conference on Automatic Differentiation, LNCSE, Springer.


Christianson, Naumann, Riehme: Unconstrained Nonlinear Optimization with Matrix-Free Truncated Newton. To be submitted to Optimization Methods and Software.


Optimization of Data-Flow Reversal

- Automatic call graph reversal
- Heuristics for near optimal call graph reversal
  - combinations of split and joint with argument and with result checkpointing reversal modes) based on static and/or dynamic (profiling) complexity estimates
- Automatic loop-level checkpointing
Support for Second Derivatives

- Reapplication of compiler to unparsed tangent-linear and adjoint Fortran codes
- Increased efficiency of tangent-linear code generated by ADAC
- Coupling ADAC with AD compiler to generate second-order tangent-linear and adjoint codes
- Application of ADOL-C and/or ADIC to unparsed tangent-linear and adjoint C-codes
Applications from Science and Engineering

- Shape optimization in collaboration with QinetiQ
- Sensitivity analysis with the Met Office’s Ported Unified Model
- Adjoint of a structural dynamics solver provided by Prof. Keane from the University of Southampton
- Coupling the compiler with the NAG Fortran 90 Library
- Further commercialisation of the project’s results through NAG
Thank you!

www.stce.rwth-aachen.de
naumann,riehme@stce.rwth-aachen.de