On Automatic Differentiation of Computer Codes

Presented to *The Institute of Cybernetics, Tallinn, 3 September 2007*

Emmanuel M. Tadjouddine

Computing Sciences Department

Aberdeen University

Aberdeen, AB24 3UE

e.tadjouddine@abdn.ac.uk

www.csd.abdn.ac.uk/~etadjoud

Outline

- *A*utomatic Differentiation
  - *F*ormalisation
  - *T*he Forward Mode
  - *T*he Reverse Mode
- *A*pplication: Optimisation of a Satellite Boom Structure
- *C*omputational Issues
  - *P*rogram Analyses
  - *G*raph Elimination
  - *N*umerical Results
- *S*ome Open Problems

Evaluating Derivatives [2]

- Problem Statement

  Given a program P computing a numerical value function $\mathbf{F}$

  $$\begin{aligned} \mathbf{F} \quad : \quad & \mathbb{R}^n \to \mathbb{R}^m \\ & \mathbf{x} \mapsto \mathbf{y} \end{aligned}$$

  build up a program that computes $\mathbf{F}$ and its *derivatives*.

- What derivatives?

  Precisely, we want derivatives of the *dependents*, e.g., some variables in the outputs $\mathbf{y}$ w.r.t. the *independents*, e.g., some variables in the inputs $\mathbf{x}$. This may be

  - The Jacobian matrix $J = \nabla \mathbf{F} = \frac{\partial \mathbf{y_i}}{\partial \mathbf{x_j}}$.
  - A directional derivative $\dot{\mathbf{y}} = J * \dot{\mathbf{x}}$.
  - Gradients when $m = 1$ as well as higher order derivatives

Finite Differencing (FD)

Given a directional derivative $\dot{\mathbf{x}}$, run P twice to compute

$$\dot{\mathbf{y}} = \frac{P(\mathbf{x} + h\dot{\mathbf{x}}) - P(\mathbf{x})}{h}$$

where h is a small non negative number.

- (+) Easy to implement
- (-) Aproximation: What step size h?
- (-) May be expensive but not always!

Accurate derivatives are needed in Optimisation for example.

Automatic Differentiation (AD)

- A semantics augmentation framework

$$P(\mathbf{x} \mapsto \mathbf{y}) \Longrightarrow \dot{\mathbf{P}}(\mathbf{x}, \dot{\mathbf{x}} \mapsto \mathbf{y}, \dot{\mathbf{y}})$$

using the chain rules of calculus to elementary operations in an automated fashion.

- A simple example

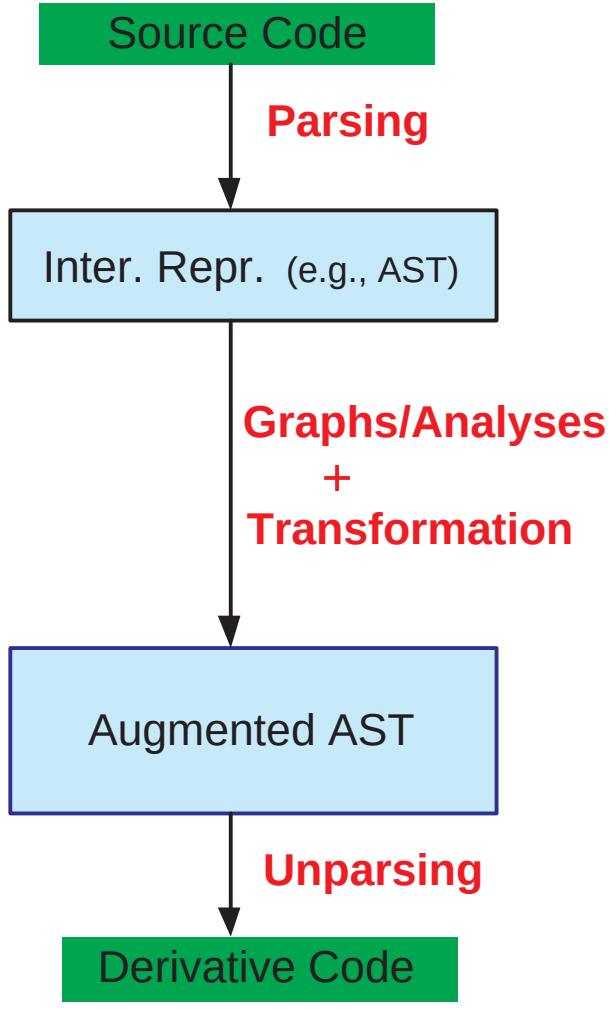
```
proc foo(a,b,s)
  REAL a,b,s

  s = a*b
end proc foo
```

$\Longrightarrow$

```
proc food(a,da,b,db,s,ds)
  REAL a,b,s
  REAL da,db,ds
  ds = a*db+da*b
  s = a*b
end proc food
```

The AD Framework



Some AD Tools: ADIFOR, Tapenade, TAF, etc.

Differentiating Programs? (1)

AD relies on the assumption that

the input program is piecewise differentiable.

To implement AD,

- Freeze the control of the input program
- View the program as a sequence of simple instructions
- Differentiate the sequence of instructions
- Caution: Some programs may not be piecewise differentiable while representing a differentiable function!

Differentiating Programs? (2)

A program P is viewed as a sequence of instructions

$$P : I_1, I_2, \ldots, I_{p-1}, I_p$$

where each I_i represents a function ϕ_i

$$I_i : v_i = \phi_i\big(\{v_j\}_{j \prec i}\big), \quad i = 1, \ldots, p.$$

computes the value of v_i in terms of previously defined v_j.

P is a composition of functions $\phi = \phi_p \circ \phi_{p-1} \circ \ldots \circ \phi_2 \circ \phi_1$

Differentiating ϕ yields

$$\phi'(\mathbf{x}) = \phi'_{\mathbf{p}}(\mathbf{v_{p-1}}) \times \phi'_{\mathbf{p-1}}(\mathbf{v_{p-2}}) \times \ldots \times \phi'_{\mathbf{1}}\big(\mathbf{x}\big)$$

$\Longrightarrow$ a chain of matrix multiplications

Forward and Reverse Modes

- Forward mode

$$\dot{\mathbf{y}} = \phi'(\mathbf{x}) \times \dot{\mathbf{x}} = \phi'_{\mathbf{p}}(\mathbf{v_{p-1}}) \times \phi'_{\mathbf{p-1}}(\mathbf{v_{p-2}}) \times \ldots \times \phi'_{\mathbf{1}}(\mathbf{x}) \times \dot{\mathbf{x}}$$

The cost of computing $\nabla \mathbf{F}$ is about $3n$ times the cost of computing $\mathbf{F}$

- Reverse mode

$$\bar{\mathbf{x}} = \phi'(\mathbf{x})^{\mathbf{T}} \times \bar{\mathbf{y}} = \phi'_{\mathbf{1}}(\mathbf{x})^{\mathbf{T}} \times \phi'_{\mathbf{2}}(\mathbf{v_1})^{\mathbf{T}} \times \ldots \times \phi'_{\mathbf{p}}(\mathbf{v_{p-1}})^{\mathbf{T}} \times \bar{\mathbf{y}}$$

The cost of computing $\nabla \mathbf{F}$ is about $3m$ times the cost of computing $\mathbf{F}$ but the memory requirement may explode.

- Gradients are cheaper by reverse mode AD.

The Reverse Mode AD

$$\bar{\mathbf{x}} = \phi_1'(\mathbf{x})^{\mathbf{T}} \times \phi_{\mathbf{2}}'(\mathbf{v_1})^{\mathbf{T}} \times \ldots \times \phi_{\mathbf{p}}'(\mathbf{v_{p-1}})^{\mathbf{T}} \times \bar{\mathbf{y}}$$

$\downarrow$

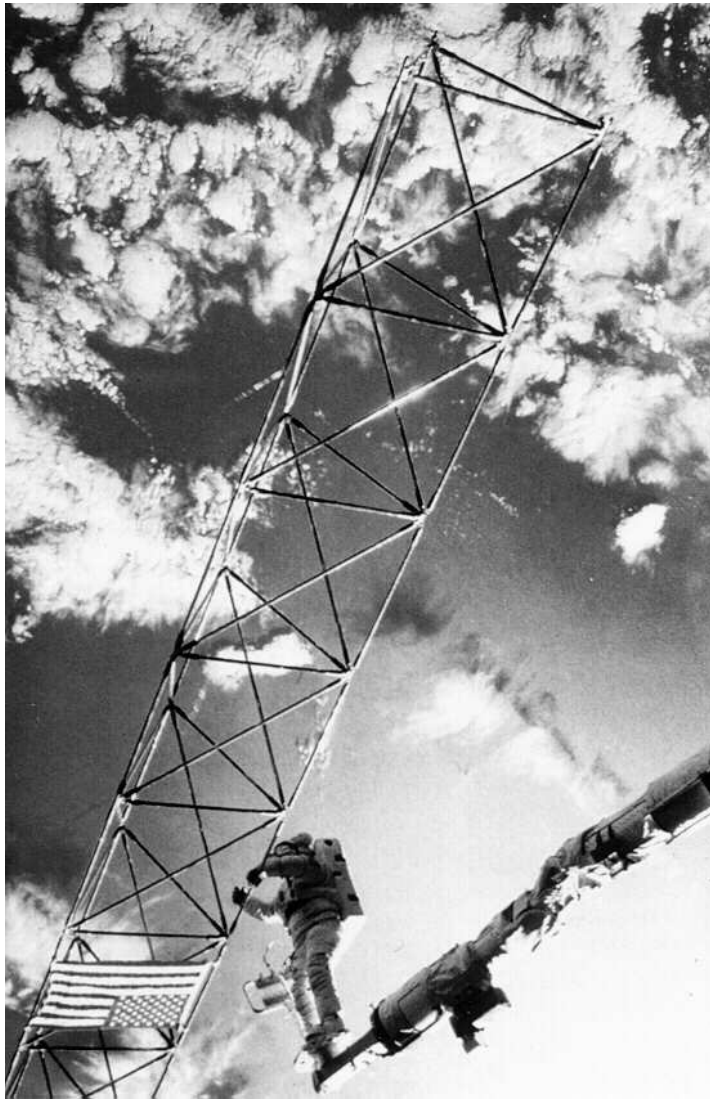
$$\begin{array}{ll} I_1: & v_1 = \phi_1(\mathbf{x}) \\ I_2: & v_2 = \phi_2(v_1, \mathbf{x}) \\ & \ldots \\ I_p & v_p = \phi_p(v_1, .., v_{p-1}, \mathbf{x}) \end{array}$$

$\uparrow$

$$\begin{array}{ll} & \bar{\mathbf{x}} = \bar{\mathbf{y}} \\ & \text{Restore } v_{p-1} \text{ before } I_{p-1} \\ \bar{I}_p & \bar{\mathbf{x}} = \phi_p'(v_{p-1})^T * \bar{\mathbf{x}} \\ & \ldots \\ & \text{Restore } \mathbf{x} \text{ before } \mathbf{I_1} \\ \bar{I}_1 & \bar{\mathbf{x}} = \phi_1'(\mathbf{x})^{\mathbf{T}} * \bar{\mathbf{x}} \end{array}$$

- Instructions are differentiated in reverse order
- Either Recompute or Store the required values. The memory/execution time usage is a bottleneck!

Structural Design [3]

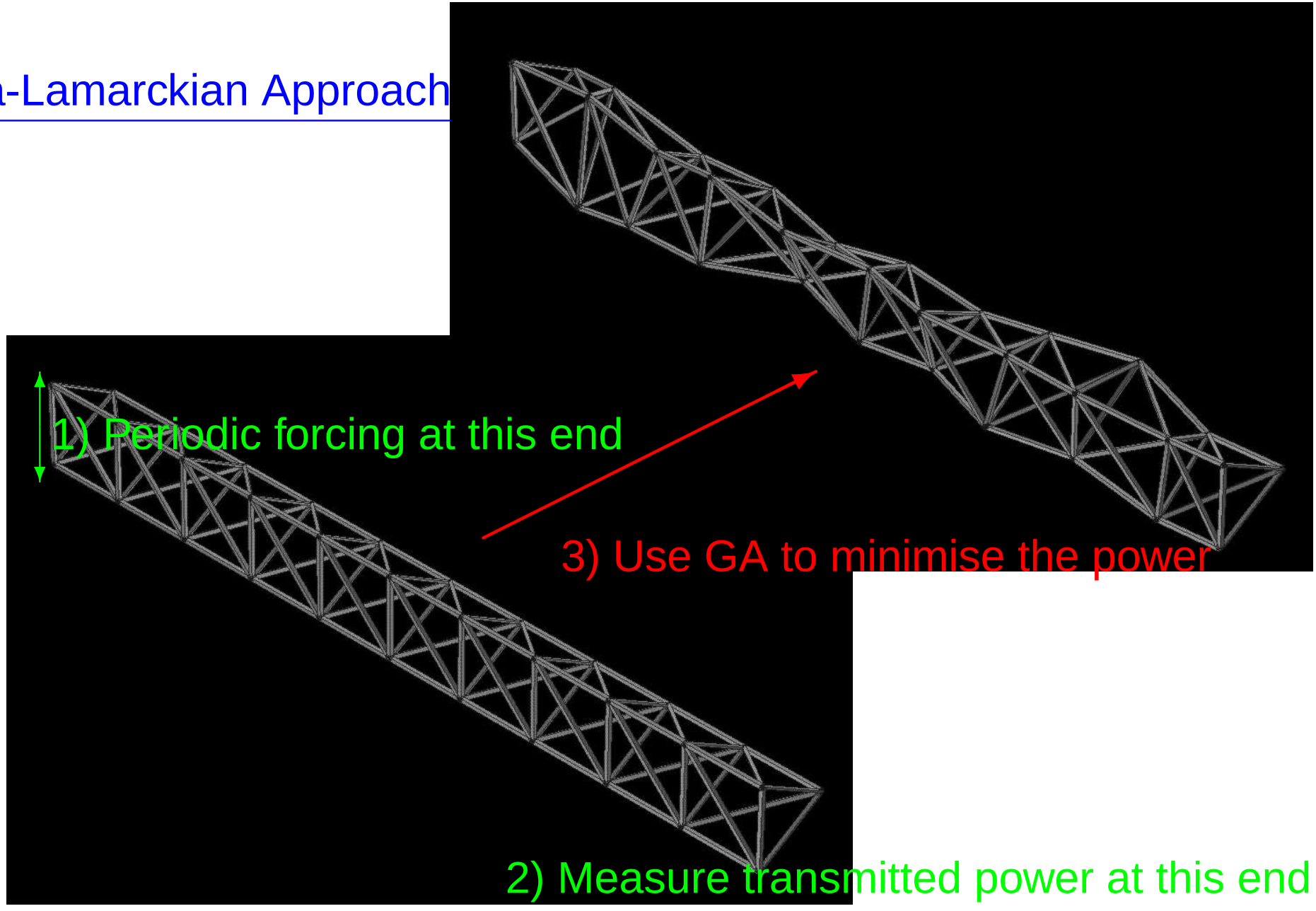


NASA Photo ID: STS61B-120-052

- Lightweight cantilever structure for suspending scientific instruments away from satellite.
- Wish to minimise transmission of vibration through structure from satellite to instrument

Optimisation of the Structure

Meta-Lamarckian Approach



Memetic Algorithm [4]

- Not a huge problem, the number of independents $n{=}453$
- Use of GA with population of 100 gives slow convergence
- Run times of 83 CPU days for 10 generations [3].
- Wish to speed convergence using Meta-Lamarckian approach [4]
  - Lamarckian evolution - *Inheritance of acquired characteristics*
  - Couple gradient descent with the GA
  - But gradient of transmitted power expensive to approximate with FD
- Need the reverse mode AD

Improved Performance

AD coupled with hand-coded optimisations gave the following results:

Method	CPU($\nabla \mathbf{F}$)(s)	CPU($\nabla \mathbf{F}$)/CPU($\mathbf{F}$)
ADIFOR(reverse)	192.0	8.2
FD (1-sided)	10912.7	464.4

- Gradient obtained now for cost equivalent to 8.2 function evaluations
- 56 times faster than FD and without truncation error
- Memory requirement of just 0.3 GB

Improving the AD process

- Avoiding useless memory storage and computations
  - Data flow analyses (e.g., dependencies between program variables)
  - Undecidability $\Longrightarrow$ conservative decisions
  - Abstract Interpretation using conservative approximations of semantics of computer programs over lattices.
- Exploiting the processor architecture
  - Code reordering techniques
  - Heuristics on code tuning à la ATLAS project (J. Dongara et al).

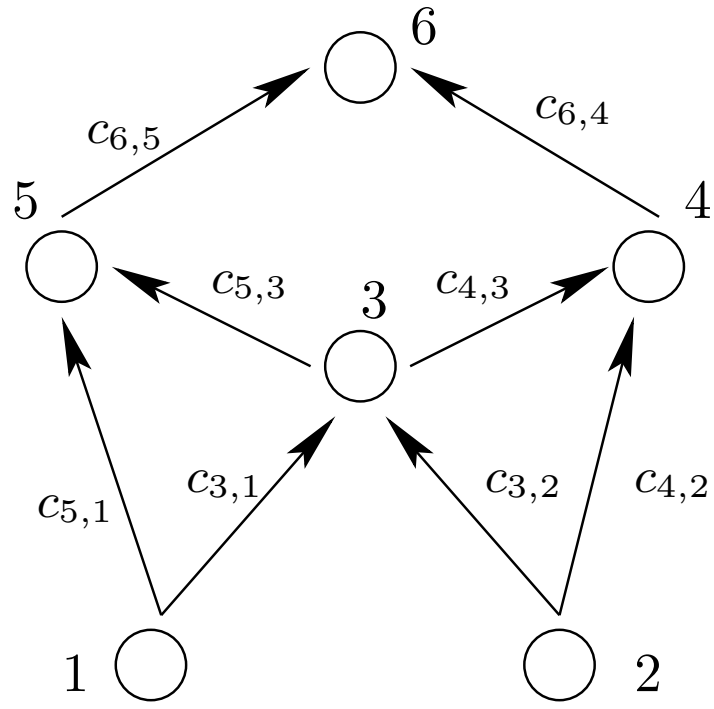
Program analyses

- Activity analysis: determine the set of active variables, e.g., those that depend on an independent and that impact a dependent.
- Common subexpression elimination: reduce the number of floating-point operations (FLOPs).
- Tape size: Minimise the set of variables to be stored or recomputed for the reverse mode.
- Sparse computations:
  - Dynamic exploitation via a sparse matrix library (as in the ADIFOR tool).
  - Static exploitation via array region analysis to detect sparse derivative objects, select an appropriate data structure and generate codes accordingly [5].
  - Graph elimination techniques to pre-accumulate local derivatives at basic block level [1, 6].

AD by Vertex Elimination (1)

Consider the code (left) and its computational graph (right):

$$
\begin{aligned}
v_3 &= \phi_3(v_1, v_2) \\
v_4 &= \phi_4(v_2, v_3) \\
v_5 &= \phi_5(v_1, v_3) \\
v_6 &= \phi_6(v_4, v_5)
\end{aligned}
$$



wherein $c_{i,j} = \partial\phi_i/\partial x_j$.

The derivative $\frac{\partial v_6}{\partial(v_1,v_2)}$ is obtained by eliminating vertices $3, 4, 5$.

AD by Vertex Elimination (2)

The Vertex Elimination (VE) approach consists in

- Building up explicitly the computational graph from the input code
- Finding a vertex elimination sequence using heuristics (forward, reverse, or cross-country orderings)
- Then, generating the derivative code

This approach enables us to

- Re-use expertise from sparse linear algebra
- Exploit the sparsity of the computation at compilation time
- Pre-accumulate local Jacobians at basic block level in a hierarchical manner.

Hierarchical AD

To pre-accumulate a basic block, perform an in-out analysis

- The inputs and outputs of basic blocks are determined using a read and write analysis.
- The inputs are those active variables that are written before, and read within, the basic block.
- The outputs are those active variables that are written in the basic block and read thereafter.

The overall differentiation is carried out by hierarchically pre-accumulating each basic block.

Numerical Results (1)

The Osher scheme is an approximate Riemann solver. It is used to evaluate the inviscid flux normal to a surface interface. The Jacobian is a 5×10 matrix.

technique	cpu(∇F)/cpu(F)	error(∇F)
Hand-Coded	2.646	0.0E+00
FD	10.824	1.6E-03
ADIFOR(fwd)	6.373	5.7E-14
ADIFOR(rev)	23.235	7.4E-14
Tapenade(fwd)	4.784	6.8E-14
VE method (best)	3.788	5.7E-14

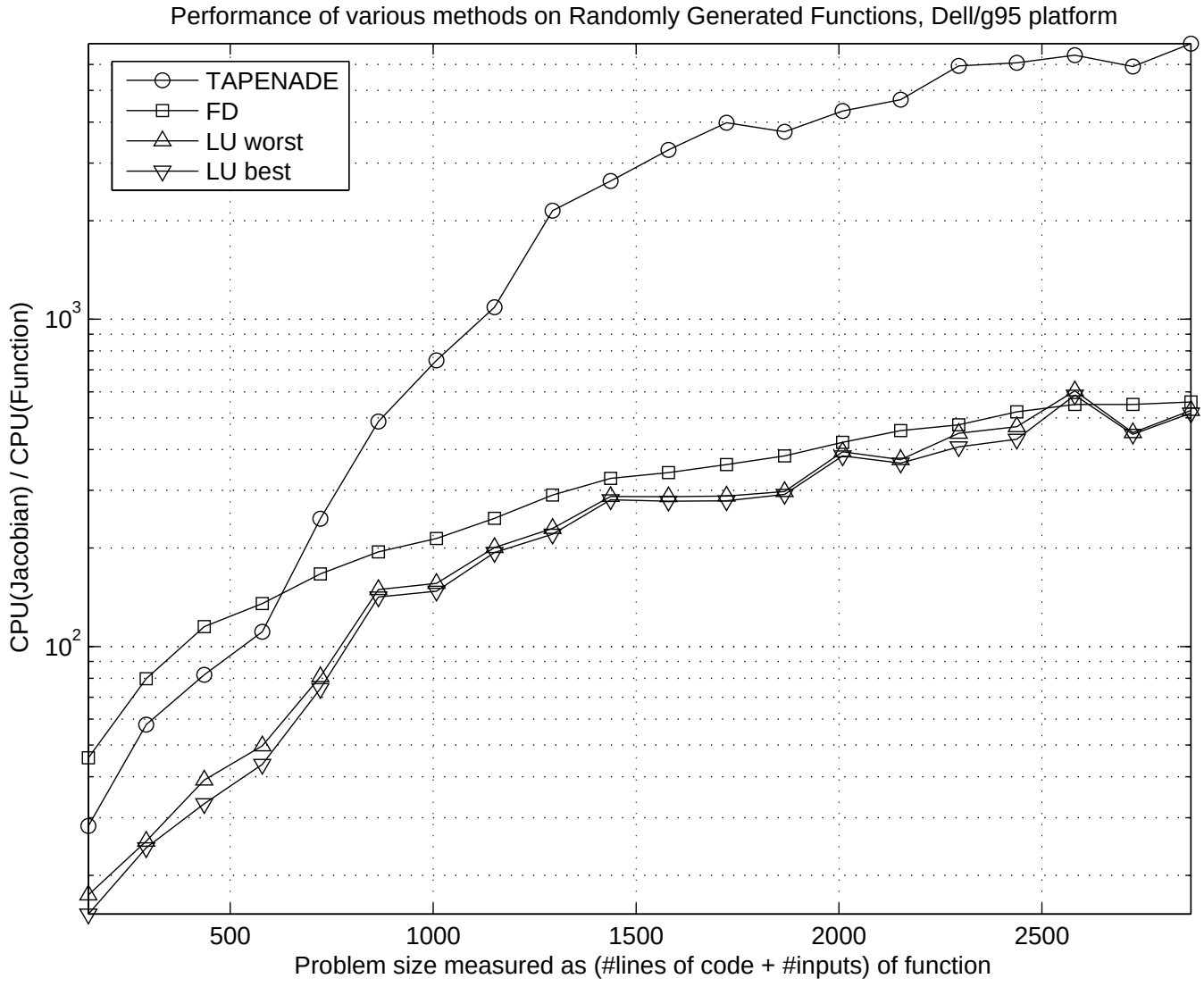
Numerical Results (2)

The Roe scheme is an approximate Riemann solver. It computes numerical fluxes of mass, energy and momentum across a cell face in a finite-volume compressible flow calculation. The Jacobian is a 5×20 matrix.

technique	cpu(∇F)/cpu(F)	error(∇F)
ADIFOR(fwd)	18.5	0.0
ADIFOR(rev)	9.1	$\mathcal{O}(10^{-16})$
FD	24.1	$\mathcal{O}(10^{-6})$
VE method (best)	4.7	$\mathcal{O}(10^{-15})$

Numerical Results (3)

LU methods are versions of *v*ertex elimination methods



Floating-point performance (1)

- Extensive tests showed performance of AD codes dependent on platforms (processors + Compilers) [1]
- Hard to attain the so-called peak performance for a given processor (the best result was 1.6 FLOPs per clock cycle for the Compaq EV6 for which the theoretical maximum is 2).
- Assembler Inspection to work out the FLOPs count as well as the memory accesses
- Hardware performance monitors, e.g., the SGI SpeedShop, SUN Workshop
- Observation: Cache utilisation is a key issue when the memory traffic dominates the computation.

Floating-point performance (2)

- CPU-Memory performance gap (annual growth of about 55% for CPU versus 7% for memory)
- Importance of developping new algorithms adapted to current cache-based machines

I would rather have today's algorithms on yersterday's computers than vice versa. [P. Toint]

Some Open Problems

AD being a multidisciplinary young research topic, here are some open questions with a computer science flavour.

- Technical issues
  - Pointers and memory allocation
  - Communications and random control (undeterministic choices by processes)

- Fundamental issues
  - Reverse mode AD for large-scale codes
  - Nondifferentiability of functions such as $\max$, abs, ...
  - The Piecewise Differentiability (PD) hypothesis

Piecewise Differentiability

- Consider the function $y = x$ coded as

```
if (x == 0.0) then
   y=0.0
else
   y=x
endif
```

  for which AD yields $dy/dx = 0$ at $x = 0$!

- In the code of the satellite boom example, such branches were constraints on geometry.

```
IF (XDIFF == 0.0 & YDIFF == 0.0) THEN
   YOR(1,I) = ZDIFF/BEAM_LENG(I)
   YOR(2,I) = 0.0
   YOR(3,I) = 0.0
ENDIF
```

Trust is an Issue

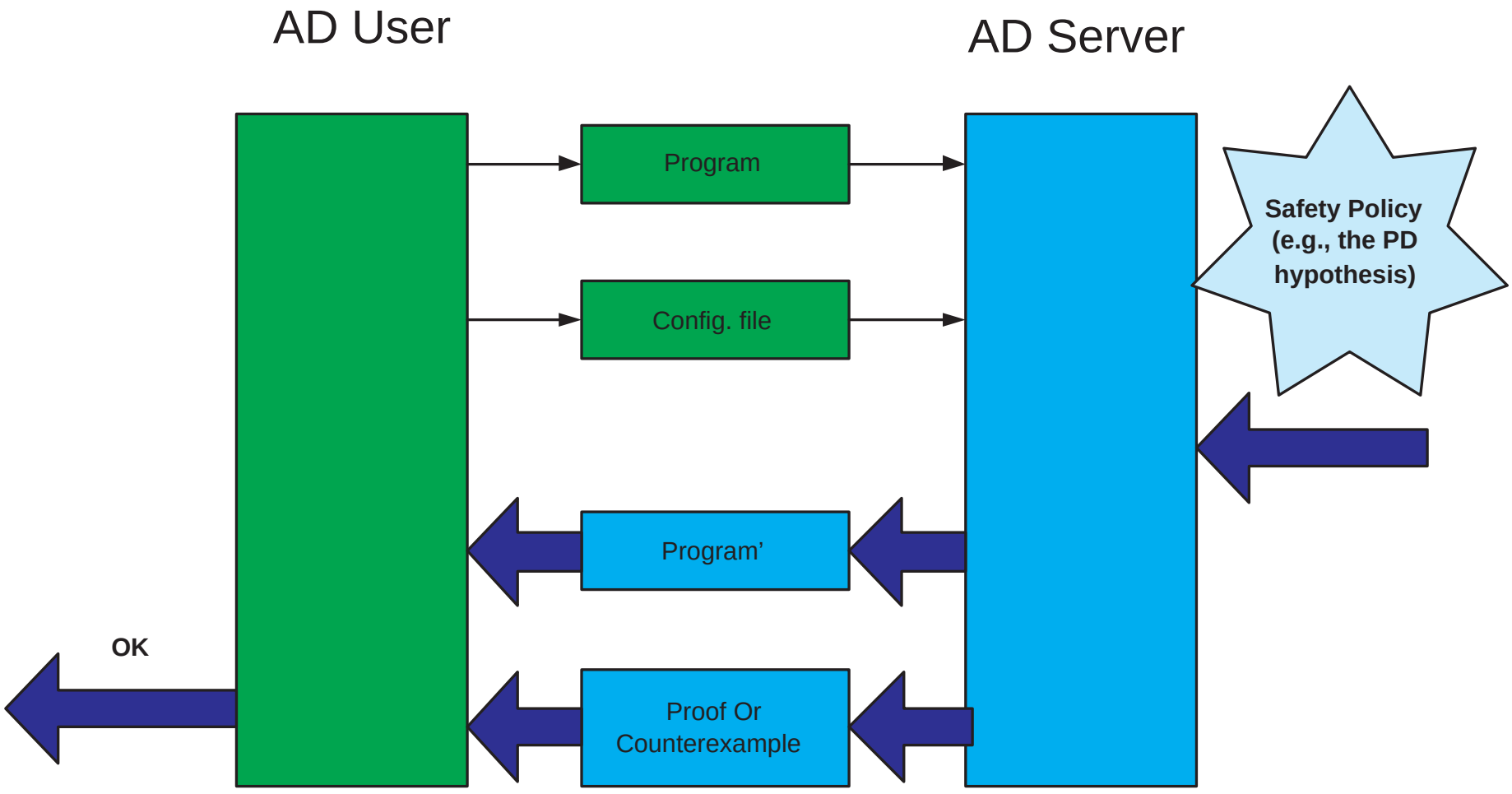
- The PD hypothesis may not be true for some codes although it is satisfied for most cases.
- Side-effect instructions need be rewritten into a canonical form suitable for AD, e.g.,

```
a[i++] = b   ⟶   a[i] = b
                 i = i+1
```

Does the canonicalized code before AD semantically equivalent to the input one?

- Users may be reluctant to changes of their input codes and may need guaranties about those changes (legacy codes).
- In this scenario, the proof-carrying code paradigm is relevant.

Towards Certified AD



Thank You!

Questions?

References

[1] FORTH, S. A., TADJOUDDINE, M., PRYCE, J. D., AND REID, J. K. Jacobian code generated by source transformation and vertex elimination can be as efficient as hand-coding. *ACM Transactions on Mathematical Software 30*, 3 (Sept. 2004), 266–299.

[2] GRIEWANK, A. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. No. 19 in Frontiers in Appl. Math. SIAM, Philadelphia, Penn., 2000.

[3] KEANE, A., AND BROWN, S. The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques. In *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 96* (Plymouth, 1996), I. C. Parmee, Ed., pp. 107–113.

[4] ONG, Y., AND KEANE, A. Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computing 8*, 2 (2004).

[5] TADJOUDDINE, M., EYSSETTE, F., AND FAURE, C. Sparse Jacobian computation in automatic differentiation by static program analysis. In *Proceedings of the Fifth International Static Analysis Symposium, Pisa, Italy* (1998), no. 1503 in LNCS, Springer, pp. 311–326.

[6] TADJOUDDINE, M., FORTH, S. A., AND PRYCE, J. D. Hierarchical automatic differentiation by vertex elimination and source transformation. In *ICCSA (2)* (2003), V. K. et al, Ed., vol. 2668 of *LNCS*, Springer, pp. 115–124.