Automatic Differentiation in the "Tropics" team

Laurent Hascoët

Projet TROPICS, INRIA Sophia-Antipolis

January 28th, 2002

A.D. Research in the Tropics team

- Utilization of A.D., mostly Reverse Mode for Optimization.
  which suggests...

- Approaches and Tactics for Reverse A.D:
  - Efficient Control-Flow Reversal
  - Selective Storage of intermediate values
  - Optimized Snapshots for Checkpointing
  - Optimized Adjoints based on Data-Dependences.

  Validated on...

Automatic Differentiation Tool: "TAPENADE"

- **TAPENADE: An A.D. Tool on the Web.**
  - A Platform for Analysis and Transformation of Imperative Languages. Accessed through an Abstract Syntax Protocol. Internal Call Graphs of Flow Graphs, with nested Symbol Tables.
  - Platform open wrt application language, and open to new transformations and algorithms.
  - A.D. forward and reverse tools built upon this platform.
  - Web interface, based on a Servlet protocol.

The TROPICS Research Team

- INRIA Sophia-Antipolis, France. Work started in 1998.

- *People from numerical analysis:*
François COURTY,
Alain DERVIEUX,
Mariano VASQUEZ

- *People from computer science:*
Rose-Marie GREBORIO,
Laurent HASCOET,
Valérie PASCUAL

A.D. in brief (1/3)

Given mathematical equations and a program (discretization), **how** can we obtain derivatives ?

– Write new equations, then discretize boldly

– Start from the given program, and

  ∗ compute divided differences

  ∗ (A.D.) transform the program to compute analytical derivatives

    · either by overloading

    · or by creating new instructions

A.D. in brief (2/3)

Consider a program $P : \{I_1; I_2; \dots I_p;\}$ implementing $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$
We identify programs with composition of functions, one per instruction:
$f = f_p \circ f_{p-1} \circ \cdots \circ f_1$ (f_k implemented by I_k). The chain rule says:

$$
\begin{aligned}
f'(x) \quad = \quad & (f'_p \circ f_{p-1} \circ f_{p-2} \circ \cdots \circ f_1(x)) \\
. & (f'_{p-1} \circ f_{p-2} \circ \cdots \circ f_1(x)) \\
. & \dots \\
. & (f'_1(x)) \\
= \quad & f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\dots.f'_1(x_0)
\end{aligned}
$$

A.D. in brief (3/3)

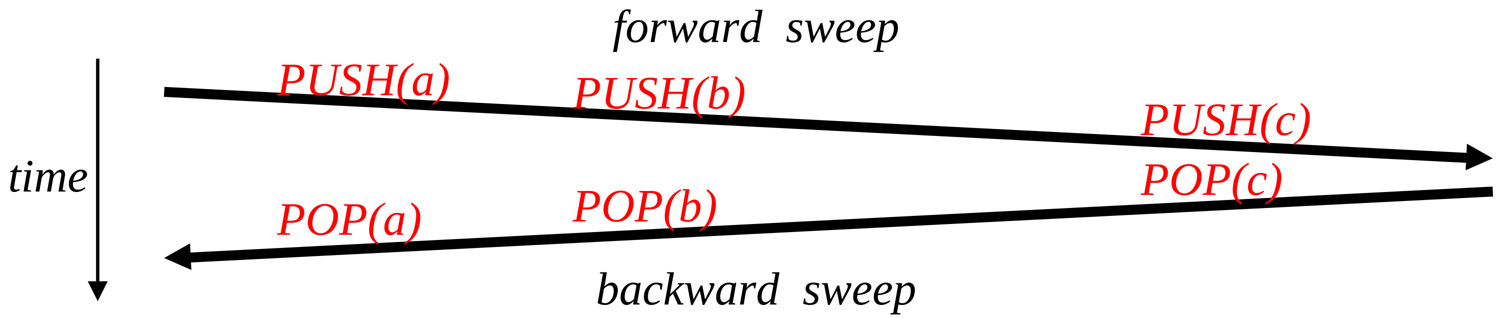
What derivatives do we need?

•**sensitivity:** $\dot{y} = f'.\dot{x}$ leads to the forward mode

•**gradient:** $\overline{x}^* = \overline{y}^*.f'$ leads to the reverse mode

$$\begin{aligned} \overline{x} &= f'^*(x).\overline{y} \\ &= f_1'^*(x_0).f_2'^*(x_1)\ldots f_{p-1}'^*(x_{p-2}).f_p'^*(x_{p-1}).\overline{y} \end{aligned}$$

$\overline{x}$ is evaluated in the reverse order of the original program's result $y = x_p$

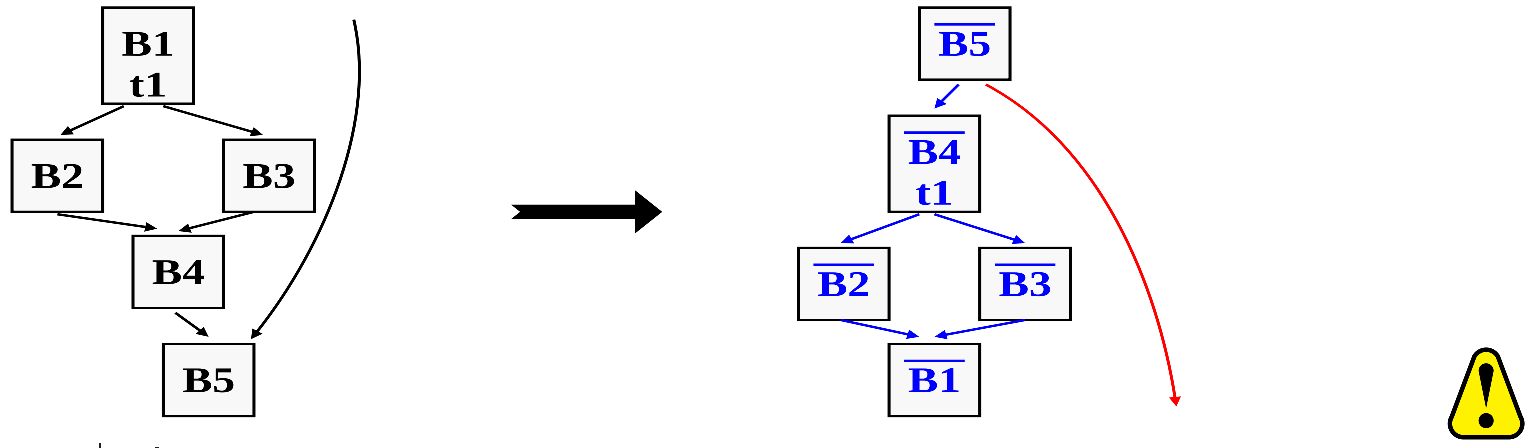


Three challenges of reverse A.D.

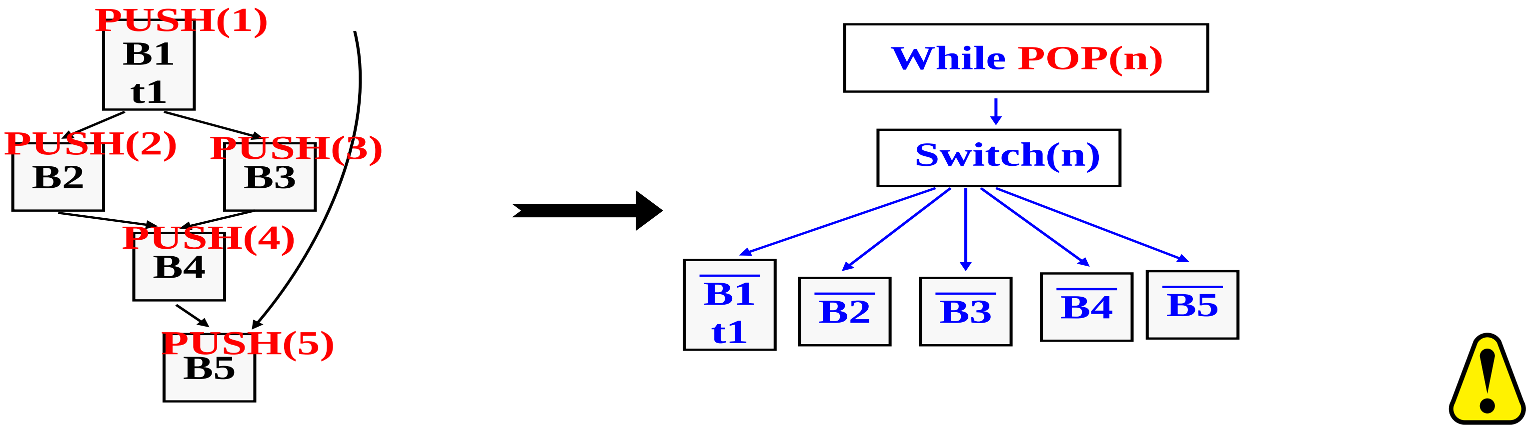
- Reverse the control flow
- Reduce storage size for intermediate values
- Exploit data-dependency information

Reverse the control flow (1/2)

A first try:

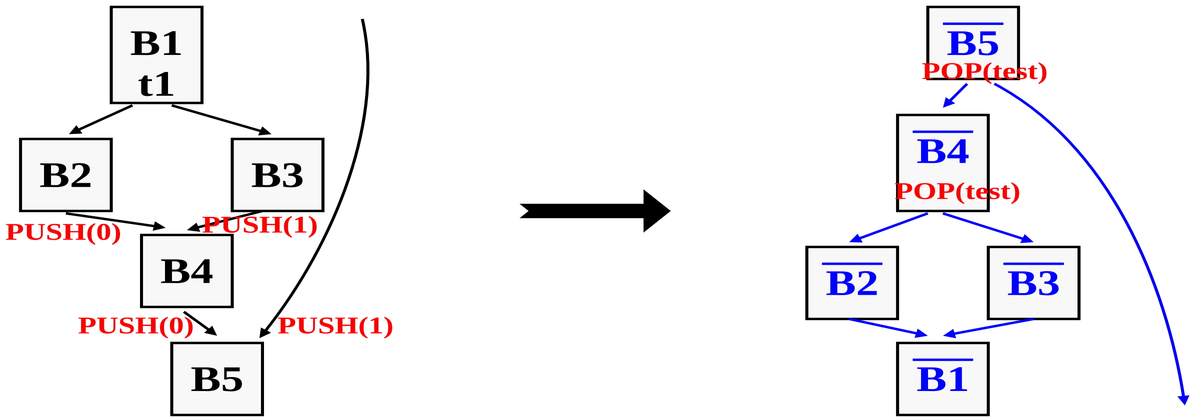


An extreme solution:

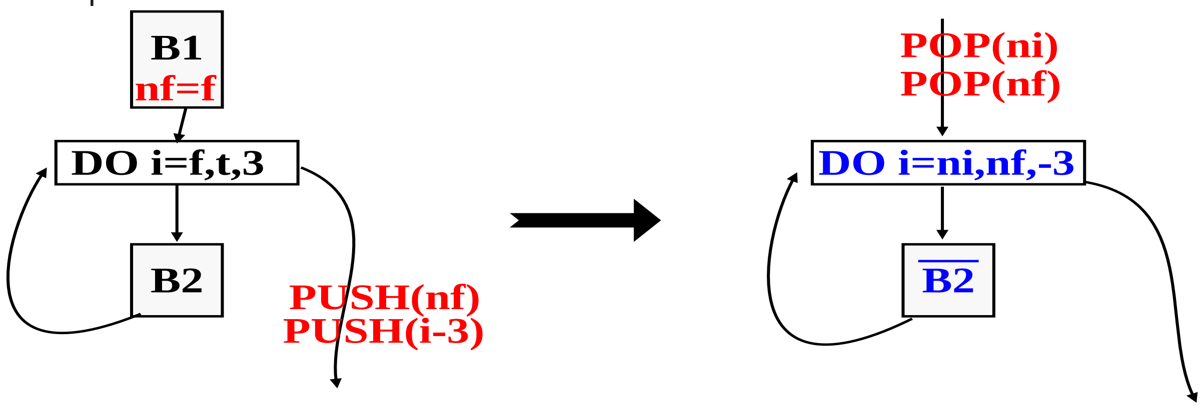


Reverse the control flow (2/2)

A better solution:



What about loops?:

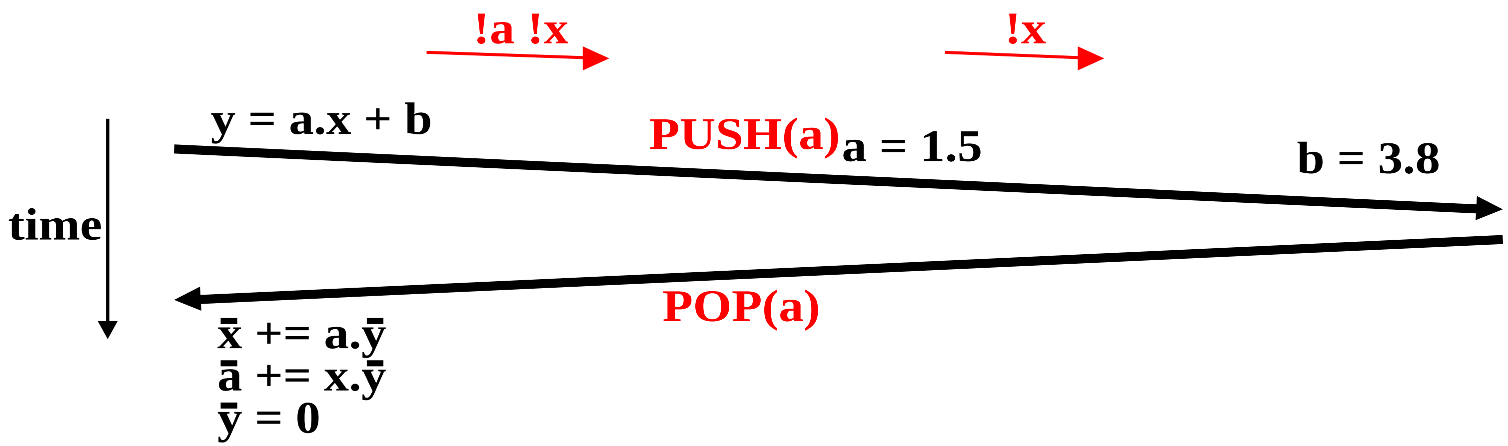


The Storage Size Problem (1/3)

Storage of intermediate values costs a lot !.

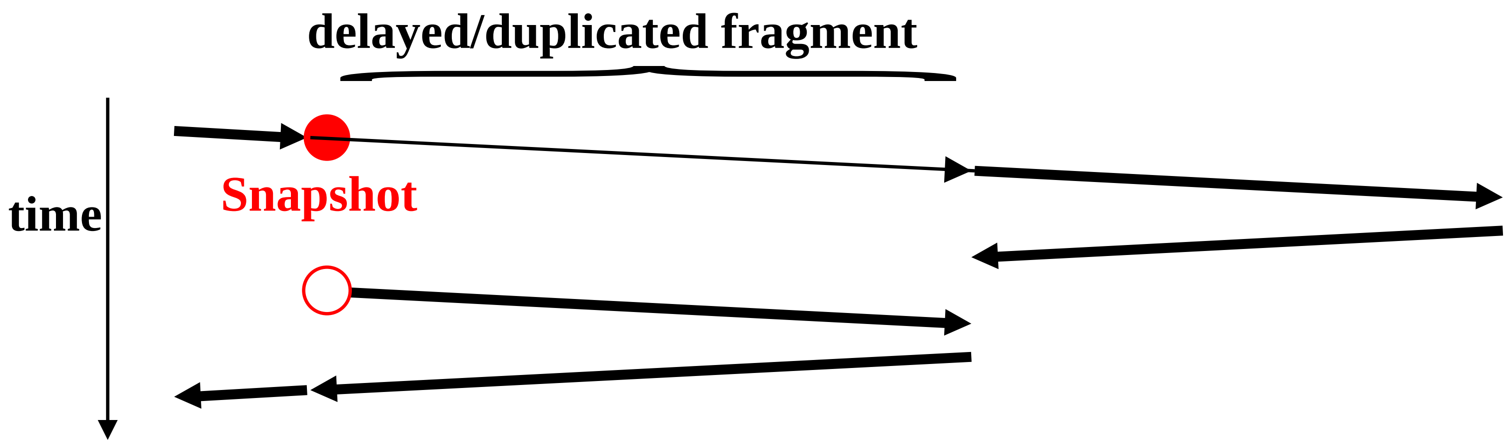
- Find unnecessary storage.

"TBR" analysis (Uwe Naumann). Now interprocedural, bottom-up then top-down.



The Storage Size Problem (2/3)

- Explore storage/recomputation tradeoffs.



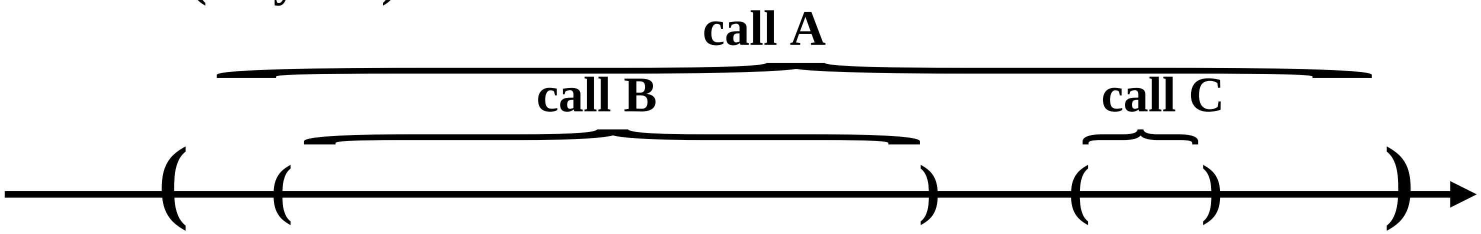
- Optimize snapshots using In-Out analysis.

Should we go into array regions?

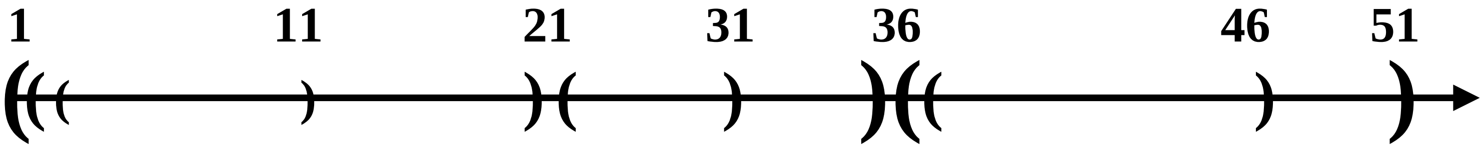
The Storage Size Problem (3/3)

- How to choose duplicated fragments?

On calls (Odyssee):



On loop iterations (Griewank):



Other choices? User's choice?.

Using Data-Dependency

Proposition:
The DDG of $\overline{P}$ is isomorphic to a subset of the DDG of P

Consequence:

$$
\begin{array}{l}
\texttt{do // i = 1,N} \\
\quad \textit{body(i)} \\
\texttt{end} \\
\texttt{do i = N,1} \\
\quad \overline{\textit{body(i)}} \\
\texttt{end}
\end{array}
\iff
\begin{array}{l}
\texttt{do i = 1,N} \\
\quad \textit{body(i)} \\
\quad \overline{\textit{body(i)}} \\
\texttt{end}
\end{array}
$$

Tape storage size divided by N.

Another use of Data-Dependency

```
y = x + y
y = 3*y
y = a*x + b
z = b*x
x = 3*t
...
t̄ += 3*x̄
x̄ = 0
x̄ += b*z̄
b̄ += x*z̄
z̄ = 0
x̄ += a*ȳ
b̄ += ȳ
ȳ = 0
ȳ = 3*ȳ
x̄ += ȳ
```

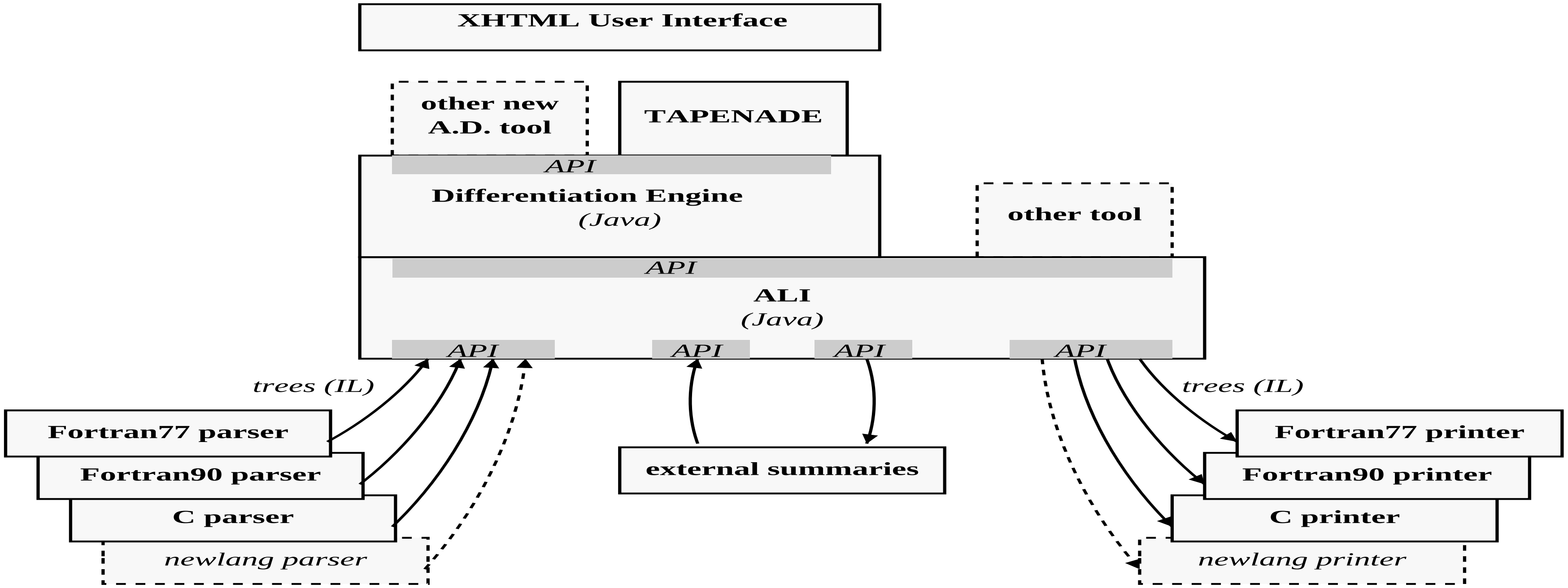
$\Longleftrightarrow$

```
...
y = x + y
y = 3*y
y = a*x + b
z = b*x
x = 3*t
...
t̄ += 3*x̄
x̄ = b*z̄ + a*ȳ
b̄ += x*z̄ + ȳ
z̄ = 0
ȳ = 0
...
```

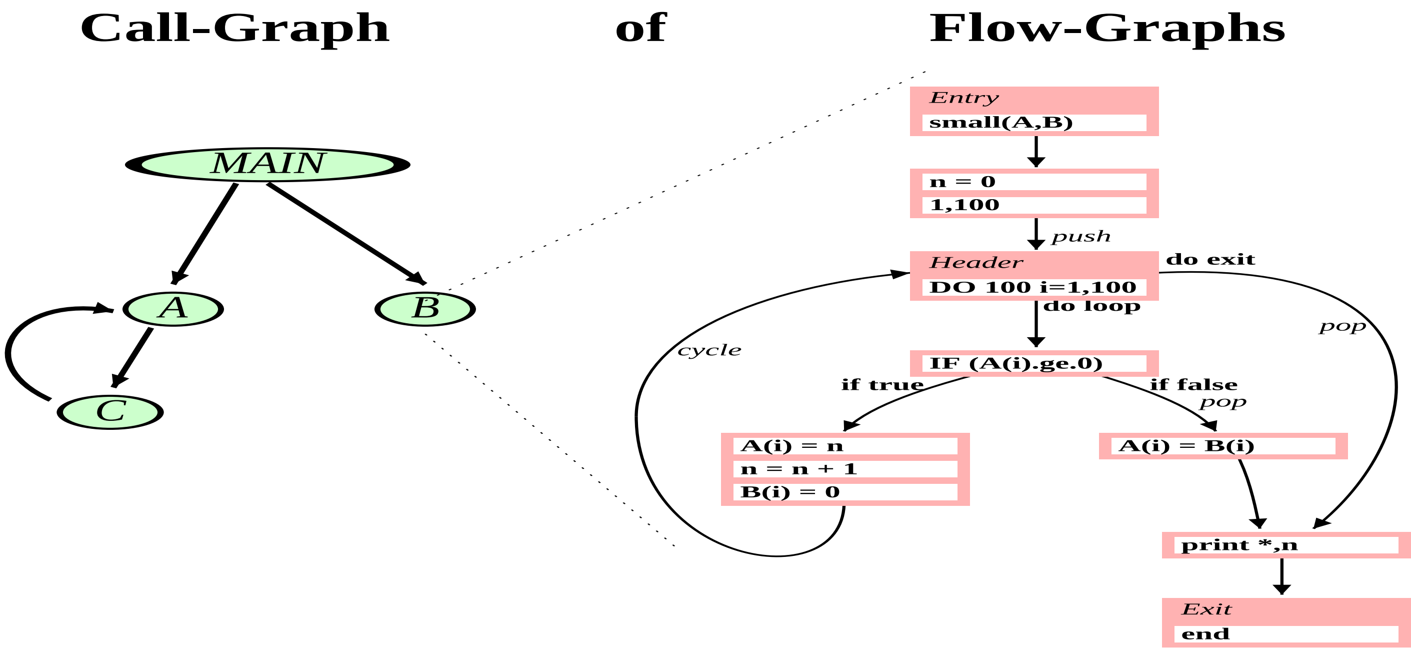
Improves data locality and legibility.

TAPENADE: An A.D. Tool on the Web

Tangent and Adjoint Automatic Differentiation Engine



TAPENADE: Internal Representation

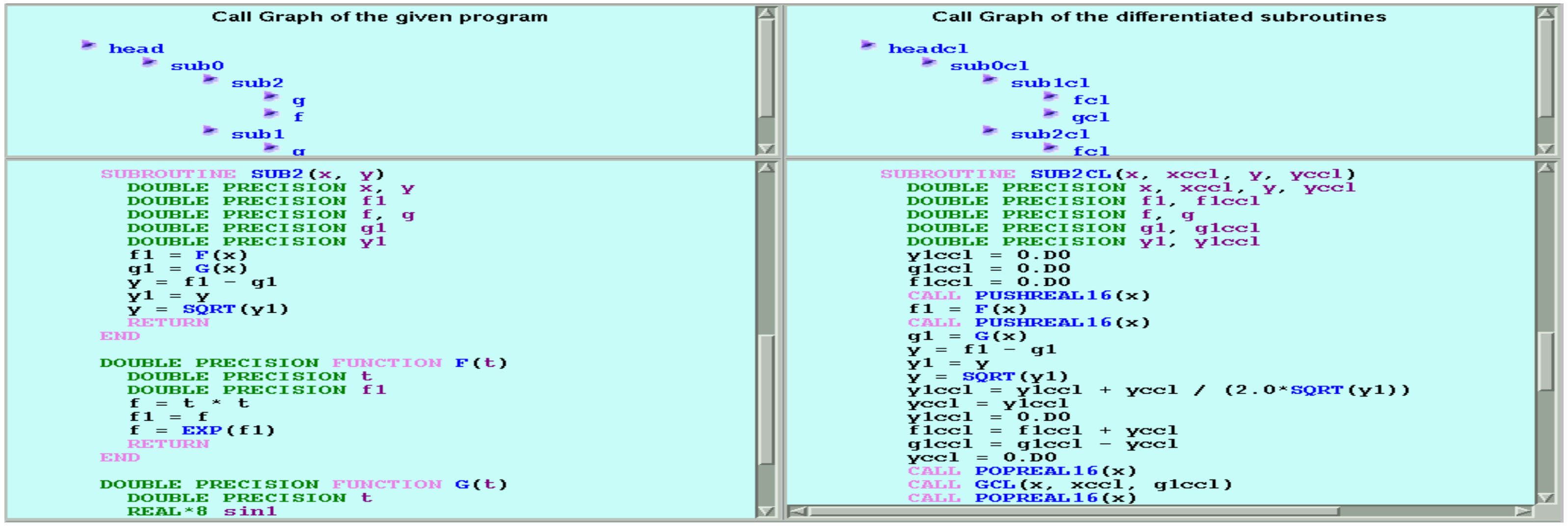


- Nested Symbol Tables for scoping
- Built-in mechanism for translation of variable-vased info across call boundaries

TAPENADE: Focus on the Reverse Mode

- Bi-directional Interprocedural Activity analysis.
- Interprocedural TBR analysis.
- Interprocedural In-Out analysis for smaller Snapshots.
- Control Reversal on the Flow Graph.
- Global Tape (reduces snapshots).
- Checkpointing on calls, with option to inhibit.
- Local optimizations on adjoint Basic Blocks.

TAPENADE : Web Interface



- Servlet on `http://www-sop.inria.fr/tropics`
- Uploads your Files and Includes
- Displays results with source-to-source link
- Batch version soon

Future

Maybe you could:

- Use the tool
- Implement drafts of your new A.D. tactics
- Experiment new analyses
- Extend the tool to your language

Possible topics:

- Other storage/recomputation
- Discontinuities warning