TAPENADE

Laurent Hascoët, Valérie Pascual

Team TROPICS
INRIA Sophia-Antipolis, France
http://www-sop.inria.fr/tropics

Berlin, november 13th, 2009
Outline

1. Presentation of TAPENADE
2. Example Applications
3. Exercise: playing with polygons
The Tapenade AD tool

**Tapenade does:**

- **Automatic Differentiation of imperative programs (F77, F90, C)**
- through **source transformation**
- provides **Tangent** and **Reverse mode**
- Reverse strategy: **Store-All + Checkpointing** on calls
- Extensive **source analysis and refinements** on the produced source.
Tapenade look and feel

Either:

```
$> tapenade -d -head "FOO(r)/(x y) BAR(q r)/(a)"
file1.f file2.f
```

... or, more colorful:
Tapenade Architecture

- 100,000 lines of Java
- Internal representation: Call Graphs of Flow Graphs
- Differentiates Fortran77-95 and C sources.
- Web server or downloadable executable.
SUBROUTINE EVAL_F (x, y)
DOUBLE PRECISION x(4), y(2), v2

v2 = TAN(x(3)*x(4))

y(1) = x(1)*v2/(x(2)-v2)
y(2) = y(1)*x(2)
END
C Differentiation of eval_f in forward (tangent) mode:
C variations of output variables:  y
C with respect to input variables:  x

SUBROUTINE EVAL_F (x, y)
DOUBLE PRECISION x(4), y(2), v2

v2 = TAN(x(3)*x(4))

y(1) = x(1)*v2/(x(2)-v2)

y(2) = y(1)*x(2)
END
C Differentiation of eval_f in forward (tangent) mode:  
C variations of output variables:  y  
C with respect to input variables:  x

SUBROUTINE EVAL_F (x, y )  
DOUBLE PRECISION x(4), y(2), v2

v2 = TAN(x(3)*x(4))  
y(1) = x(1)*v2/(x(2)-v2)  
y(2) = y(1)*x(2)  
END

C Differentiation of eval_f in forward (tangent) mode:  
C variations of output variables:  y  
C with respect to input variables:  x

v2d = (xd(3)*x(4)+x(3)*xd(4))  
& *(1.0+TAN(x(3)*x(4))**2)  
v2 = TAN(x(3)*x(4))  
yd(1) = ((xd(1)*v2+x(1)*v2d)*(x(2)-v2)  
& -x(1)*v2*(xd(2)-v2d))/(x(2)-v2)**2  
y(1) = x(1)*v2/(x(2)-v2)  
yd(2) = yd(1)*x(2) + y(1)*xd(2)  
y(2) = y(1)*x(2)  
END
C Differentiation of eval_f in forward (tangent) mode:
C variations of output variables: y
C with respect to input variables: x

SUBROUTINE EVAL_F_D (x, xd, y, yd)
DOUBLE PRECISION x(4), y(2), v2
DOUBLE PRECISION xd(4), yd(2), v2d
v2d = (xd(3)*x(4)+x(3)*xd(4))
& *(1.0+TAN(x(3)*x(4))**2)
v2 = TAN(x(3)*x(4))
yd(1) = ((xd(1)*v2+x(1)*v2d)*(x(2)-v2)
& -x(1)*v2*(xd(2)-v2d))/(x(2)-v2)**2
& yd(1)*x(2) + y(1)*xd(2)
y(1) = x(1)*v2/(x(2)-v2)
yd(2) = yd(1)*x(2) + y(1)*xd(2)
y(2) = y(1)*x(2)
END
SUBROUTINE EVAL_F_B(x, xb, y, yb)

... ...

v2 = TAN(x(3)*x(4))
y(1) = x(1)*v2/(x(2)-v2)
y(2) = y(1)*x(2)
C Differentiation of eval_f in reverse (adjoint) mode:
C gradient, with respect to input variables: x ...

SUBROUTINE EVAL_F_B(x, xb, y, yb)
...

v2 = TAN(x(3)*x(4))
y(1) = x(1)*v2/(x(2)-v2)
y(2) = y(1)*x(2)

DO ii1 = 1, 4
  xb(ii1) = 0.D0
ENDDO
yb(1) = yb(1) + x(2)*yb(2)
xb(2) = y(1)*yb(2)
yb(2) = 0.D0
tempb = yb(1)/(x(2)-v2)
tempb0 = -(x(1)*v2*tempb/(x(2)-v2))
xb(1) = xb(1) + v2*tempb
v2b = x(1)*tempb - tempb0
xb(2) = xb(2) + tempb0
yb(1) = 0.D0
tempb1 = (1.0+TAN(x(3)*x(4))**2)*v2b
xb(3) = xb(3) + x(4)*tempb1
xb(4) = xb(4) + x(3)*tempb1
C Differentiation of eval\_f in reverse (adjoint) mode:  
C gradient, with respect to input variables: x ...

SUBROUTINE EVAL\_F\_B(x, xb, y, yb)

... ... ...

\[ v_2 = \tan(x(3) \cdot x(4)) \]
\[ y(1) = x(1) \cdot v_2 / (x(2) - v_2) \]
\[ DO_{ii=1}^{4} * x(2) \]
\[ \text{xb}(ii1) = 0.0 \]
ENDDO

\[ yb(1) = yb(1) + x(2) \cdot yb(2) \]
\[ xb(2) = y(1) \cdot yb(2) \]
\[ yb(2) = 0.0 \]
\[ \text{tempb} = yb(1) / (x(2) - v_2) \]
\[ \text{tempb0} = -(x(1) \cdot v_2 \cdot \text{tempb} / (x(2) - v_2)) \]
\[ \text{xb}(1) = \text{xb}(1) + v_2 \cdot \text{tempb} \]
\[ v_2b = x(1) \cdot \text{tempb} - \text{tempb0} \]
\[ \text{xb}(2) = \text{xb}(2) + \text{tempb0} \]
\[ yb(1) = 0.0 \]
\[ \text{tempb1} = (1.0 + \tan(x(3) \cdot x(4)) \cdot 2) \cdot v_2b \]
\[ \text{xb}(3) = \text{xb}(3) + x(4) \cdot \text{tempb1} \]
\[ \text{xb}(4) = \text{xb}(4) + x(3) \cdot \text{tempb1} \]

END
Costs of Tangent and Reverse AD

\[ F : \mathbb{R}^m \rightarrow \mathbb{R}^n \]

\[ \text{Gradient} \]
\[ \text{Tangent} \]

- \( J \) costs \( m \times 4 \times P \) using the tangent mode
  Good if \( m \leq n \)
- \( J \) costs \( n \times 4 \times P \) using the reverse mode
  Good if \( m \gg n \) (e.g. \( n = 1 \) in optimization)
The Store-All strategy in Reverse mode

The **Reverse** mode uses intermediates in reverse order!

\[ y = y + \exp(a) \]

\[ y = y + a^2 \]

\[ a = 3.5z \]
The Store-All strategy in Reverse mode

The Reverse mode uses intermediates in reverse order!

\[ y = y + \exp(a) \]
\[ y = y + a**2 \]
\[ a = 3.5*z \]
\[ zb = zb + 3.5*ab \]
\[ ab = 0.0 \]
\[ ab = ab + 2*a*yb \]
\[ ab = ab + \exp(a)*yb \]
The Store-All strategy in Reverse mode

The **Reverse** mode uses intermediates in **reverse** order!

```plaintext
CALL PUSHREAL8(y)
y = y + EXP(a)
CALL PUSHREAL8(y)
y = y + a**2
CALL PUSHREAL8(a)
a = 3.5*z
CALL POPREAL8(a)
zb = zb + 3.5*ab
ab = 0.0
CALL POPREAL8(y)
ab = ab + 2*a*yb
CALL POPREAL8(y)
ab = ab + EXP(a)*yb
```

⇒ Alternative: **Recompute-All** (TAF)
⇒ Coarse-grain tradeoff: **Checkpointing** (on calls)
Refinement: To-Be-Recorded analysis

In reverse AD, not all values must be recorded for the backward sweep.

Variables occurring only in linear expressions do not appear in the differentiated instructions.
⇒ not To-Be-Recorded.

This is a global analysis that must be done on the complete program.
Illustration of TBR analysis

CALL PUSHREAL8(y)
y = y + EXP(a)
CALL PUSHREAL8(y)
y = y + a**2
CALL PUSHREAL8(a)
a = 3.5*z
CALL POPREAL8(a)
zb = zb + 3.5*ab
ab = 0.0
CALL POPREAL8(y)
ab = ab + 2*a*yb
CALL POPREAL8(y)
ab = ab + EXP(a)*yb
CALL PUSHREAL8(y)  \[\text{tbr}=\emptyset\]
y = y + \exp(a)  \[\text{tbr}=\{a\}\]
CALL PUSHREAL8(y)  \[\text{tbr}=\{a\}\]
y = y + a^2  \[\text{tbr}=\emptyset\]
CALL PUSHREAL8(a)
a = 3.5\cdot z  \[\text{tbr}=\emptyset\]
CALL POPREAL8(a)
zb = zb + 3.5\cdot ab
ab = 0.0  \[\text{tbr}=\emptyset\]
CALL POPREAL8(y)
ab = ab + 2a\cdot yb
CALL POPREAL8(y)
ab = ab + \exp(a)\cdot yb
Tapenade does/doesn’t

Tapenade does handle
- all sorts of globals and COMMON’s.
- modules, overloading, renaming, interfaces
- structured types (“records”)
- pointers and allocation

Tapenade does **not** handle
- fpp or cpp keys, templates
- deallocation in reverse more
- checkpointing of non-reentrant code
- classes and objects
Outline

1. Presentation of TAPENADE
2. Example Applications
3. Exercise: playing with polygons
CFD optimization example

- Cost function: sonic boom below + lift + drag
- Design parameters: plane skin, (2000 REAL*8)
- Specific strategy for a steady-state simulation: assembly of the adjoint linear system through AD, then specific solver.
- Performances:
  - Differentiation time: 2 s.
  - Reverse AD slowdown: 7
  - Adjoint slowdown: 4
  - Reverse AD memory use: 58 REAL*8 per mesh node
CFD optimization result

AD gradient of the cost function on the skin geometry:

(Dassault Aviation)

Sonic boom under the plane after 8 optimization cycles:
Oceanography example

- Code: OPA 9.0. 120000 lines of FORTRAN 95
- Cost function: e.g. some cumulated heat flux vs. temperature, salinity... at initial state
- Standard reverse AD of complete unsteady simulation
- Differentiation time: 20 s.
- Reverse AD slowdown: 7
Influence of $T$ at -300 metres on heat flux 20 days later across North section

Kelvin wave

Rossby wave

15° North

30° North
Gradient OPA-NEMO

2° grid cells, one year simulation
1. Presentation of TAPENADE

2. Example Applications

3. Exercise: playing with polygons
The goal of the game

- Given an initial polygon
- and a program that computes a polygon’s surface $S$ and perimeter $P$,
- move the polygon’s vertices to maximize $S$ ($P$ constant),
- using gradient-based optimization.
- Use Tapenade to build the derivative code.
Strategy 1: fixed step size

Use the gradient \((\overline{X}, \overline{Y})\) of the scalar objective function \(P^2/S\) with respect to the input parameters \((X, Y)\).

Update the polygon’s \((X, Y)\) by

\[
\Delta X = \varepsilon \times \overline{X} \\
\Delta Y = \varepsilon \times \overline{Y}
\]

with a fixed small enough step size \(\varepsilon\).

Use Tapenade to produce the code that computes the gradient. 
\(\Rightarrow\) requires the reverse mode of Tapenade.
Strategy 1: results

714 optimization steps; time 32
Strategy 2: adaptive step size

Estimate the step size using the Newton method in the gradient direction

\[ \Delta X = p \times \bar{X} \]
\[ \Delta Y = p \times \bar{Y} \]

with \( p = -\frac{k'}{k''} \), where \( k \) is the cost function along the gradient direction.

Use Tapenade to produce the code that computes \( k' \) and \( k'' \).
\( \Rightarrow \) requires the reverse mode of Tapenade and
\( \Rightarrow \) requires the 2\textsuperscript{nd}-order tangent on tangent mode of Tapenade.
Strategy 2: results

31 optimization steps; time 4.6
Strategy 3: true Newton step

The true Newton step \((\Delta X, \Delta Y)\) is in fact the solution of

\[
\frac{\partial^2 K}{\partial (X, Y) \partial (X, Y)} \times (\Delta X, \Delta Y) = -\frac{\partial K}{\partial (X, Y)}
\]

involving the gradient and the Hessian of the cost function \(K\).

Use Tapenade to produce the code that computes

\[
\frac{\partial^2 K}{\partial (X, Y) \partial (X, Y)}
\]

⇒ requires the reverse mode of Tapenade and
⇒ requires the 2\(^{nd}\)-order multi-directional tangent on reverse mode of Tapenade.
Strategy 3: results

15 optimization steps; time 7.5
... and maybe more if time permits!