University of Nice Sophia-Antipolis

On two developments of the adjoint model of Algorithmic Differentiation to meet the needs of computationally intensive applications

A. TAFTAF

Thesis Defense Presentation

17 January 2017

Algorithmic Differentiation (AD)

- Derivatives of functions are required in many areas of computational science, e.g. design optimization.
- For complex function $F : X \in \boldsymbol{R}^N \rightarrow Y \in \boldsymbol{R}^M$, expressed as a computer program P, a method known as Algorithmic Differentiation produces code that computes the derivatives of this functions.
- AD considers the original program P as a set of successive instructions $\{I_1; I_2; ...I_p;\}$ that computes a set of elementary differentiable functions $\{g_1, g_2, ...g_p\}$, each by instruction I_k, so that:

$$F(X) = g_p(...(g_2(g_1(X))...)) \tag{1}$$

Tangent and Adjoint modes of AD

- Applying the chain rule to $F(X)$ gives a new function $F'(X)$ that computes the first order full derivatives, i.e. the Jacobian:

$$F'(X) = g'_p(X_{p-1}) * \ldots * g'_2(X_1) * g'_1(X) \tag{2}$$

- We may extend every instruction I_k, so that it computes in addition to the elementary function g_k, its derivative g'_k.
- Computing the whole Jacobian $F'(X)$ may require too much time and memory.
- In practice, we compute one of these two projections:

$$F'(X) * \dot{X} \quad or \quad \overline{Y} * F'(X)$$

where $\dot{X}$ is a vector in $\boldsymbol{R}^N$ and $\overline{Y}$ is a row-vector in $\boldsymbol{R}^M$.

- The particular modes of AD that compute these projections are called respectively the tangent and adjoint modes.

Adjoint mode of AD

- When the number of inputs is much larger then the number of outputs, e.g. gradients, the adjoint mode is recommended.
- The adjoint mode computes $\overline{X} = \overline{Y} * F'(X)$. Recalling equation 2, we get:

$$\overline{X} = \overline{Y} * g'_p(X_{p-1}) * ... * g'_2(X_1) * g'_1(X) \tag{3}$$

- Equation 3 is best computed from left to right.
- Primal values are needed in the opposite of their computation order, i.e. X_{p-1} is needed first, then X_{p-2}, then X_{p-3}, etc.

Dealing with Data-Flow reversal of Adjoint mode

In the Store-All context, the adjoint code consist of two sweeps:

- First sweep (the forward sweep) runs the original program and saves the intermediate values.
- Second sweep (the backward sweep) computes the partial derivatives, using the intermediate primal values already stored during the first sweep..

$$\textit{SUBROUTINE FOO}(x,\ y\ )$$

$$w = x * x$$

$$w = 2 * w^3$$

$$w = sin(w)$$

$$y = w$$

$$\textit{END}$$

Dealing with Data-Flow reversal of Adjoint mode

$SUBROUTINE\ \overline{FOO}(x, \overline{x}, y, \overline{y})$

$push(w)$

$w = x * x$

$push(w)$

$w = 2 * w^3$

$push(w)$

$w = sin(w)$

$push(y)$

$y = w$

$pop(y)$

$\overline{w} = \overline{y}; \overline{y} = 0$

$pop(w)$

$\overline{w} = \overline{w} * cos(w)$

$pop(w)$

$\overline{w} = 6 * \overline{w} * w^2$

$pop(w)$

$\overline{x} = 2 * \overline{w} * x$

$\overline{w} = 0$

END

Motivations

- This thesis is part of AboutFlow project which focuses on methods of optimization gradient-based.
- Adjoint mode of Algorithmic Differentiation (AD) is particularly attractive for computing gradients
- Adjoint mode requires data-flow reversal which is very expensive.

$\Rightarrow$ How to reduce the cost of Adjoint mode of AD ?

Contents

1 Introduction

2 An efficient Adjoint of Fixed-Point Loops

3 Summary and Further Work

Fixed-Point (FP) loops

- Many equations $F(z, x) = 0$ may be solved by using iterative methods:

```
z = initial_guess
Do while z not converged
    z = φ(z,x)
end
y = f(z,x)
```

- state : variable that reaches a stationary value: `z`
- parameters : variables that influence the result but never modified: `x`

Standard adjoint of Fixed-Point loops: Black Box

`z = initial_guess`

Do while `z` *not converged*

`push(z)`

`z = ϕ(z,x)`

end

`y = f(z,x)`

$$\overline{x} = \overline{y} * \frac{\partial f}{\partial x}(z,x)$$

$$\overline{z} = \overline{y} * \frac{\partial f}{\partial z}(z,x)$$

Do "as many times"

`pop(z)`

$$\overline{x} \mathrel{+}= \overline{z} * \frac{\partial \phi}{\partial x}(z,x)$$

$$\overline{z} = \overline{z} * \frac{\partial \phi}{\partial z}(z,x)$$

end

Efficient adjoint of Fixed-Point loops: Piggyback

- Many FP loops satisfy a FP equation of the form: $\mathtt{z} = \mathtt{z} - \mathtt{P} * \mathtt{F}(\mathtt{z}, \mathtt{x})$.
- The adjoint of these loops may be written as:

Repeat until $\overline{\mathtt{w}}$ *and* $\overline{\mathtt{z}}$ *stables*

```
w= F(z,x)
```

$$\overline{\mathtt{z}} = \overline{\mathtt{w}} * \frac{\partial \mathtt{F}}{\partial \mathtt{z}}(\mathtt{z},\mathtt{x}) + \overline{\mathtt{y}} * \frac{\partial \mathtt{f}}{\partial \mathtt{z}}(\mathtt{z},\mathtt{x})$$

```
z = z-P*w
```

$$\overline{\mathtt{w}} = \overline{\mathtt{w}} - \overline{\mathtt{z}} * \mathtt{P}$$

end

$$\overline{\mathtt{x}} = \overline{\mathtt{w}} * \frac{\partial \mathtt{F}}{\partial \mathtt{x}}(\mathtt{z},\mathtt{x}) + \overline{\mathtt{y}} * \frac{\partial \mathtt{f}}{\partial \mathtt{x}}(\mathtt{z},\mathtt{x})$$

Refinements of the Piggyback approach

- **Delayed Piggyback**: Consists in applying Piggyback after that the original Fixed-Point loop has "sufficiently" converged to the solution.
- **Blurred Piggyback**: Piggyback can be applied in a parallel way: a process that computes the original values and another process that computes the adjoint values. The Blurred Piggyback proposes to run these processes in an asynchronous way.

Efficient adjoint of Fixed-Point loops: Two-Phases

Do while z *not converged*

$$\texttt{z} = \phi(\texttt{z},\texttt{x})$$

end

$$\texttt{z} = \phi(\texttt{z},\texttt{x})$$

store(z)

$$\texttt{y} = \texttt{f}(\texttt{z},\texttt{x})$$

$$\overline{\texttt{x}} = \overline{\texttt{y}} * \frac{\partial \texttt{f}}{\partial \texttt{x}}(\texttt{z},\texttt{x})$$

$$\overline{\texttt{z}} = \overline{\texttt{y}} * \frac{\partial \texttt{f}}{\partial \texttt{z}}(\texttt{z},\texttt{x})$$

$$\overline{\texttt{w}} = \overline{\texttt{z}};\ \overline{\texttt{x}}_0 = \overline{\texttt{x}}$$

Do while $\overline{\texttt{w}}$ *not converged*

restore(z)

$$\overline{\texttt{w}} = \overline{\texttt{w}} * \frac{\partial \phi}{\partial \texttt{z}}(\texttt{z},\texttt{x}) + \overline{\texttt{z}}$$

$$\overline{\texttt{x}} = \overline{\texttt{w}} * \frac{\partial \phi}{\partial \texttt{x}}(\texttt{z},\texttt{x}) + \overline{\texttt{x}}_0$$

end

Refinement of the Two-Phases approach

Do while z *not converged*

$$z = \phi(z,x)$$

end

$$z = \phi(z,x)$$

store(z)

$$y = f(z,x)$$

$$\overline{x} = \overline{y} * \frac{\partial f}{\partial x}(z,x)$$

$$\overline{z} = \overline{y} * \frac{\partial f}{\partial z}(z,x)$$

$$\overline{w} = \overline{z};\ \overline{x}_0 = \overline{x}$$

Do while $\overline{w}$ *not converged*

restore(z)

$$\overline{w} = \overline{w} * \frac{\partial \phi}{\partial z}(z,x) + \overline{z}$$

end

$$\overline{x} = \overline{w} * \frac{\partial \phi}{\partial x}(z,x) + \overline{x}_0$$

Refinement of the Black Box approach

Do while z *not converged*

```
z = φ(z,x)
```

end

```
z = φ(z,x)
store(z)
y = f(z,x)
```

$$\overline{x} = \overline{y} * \frac{\partial f}{\partial x}(z,x)$$

$$\overline{z} = \overline{y} * \frac{\partial f}{\partial z}(z,x)$$

Do "as many times"

```
restore(z)
```

$$\overline{x} \mathrel{+}= \overline{z} * \frac{\partial \phi}{\partial x}(z,x)$$

$$\overline{z} = \overline{z} * \frac{\partial \phi}{\partial z}(z,x)$$

end

Our choices

We prefer the approach that:

- Does not make assumptions on the shape of the FP loop and easy to implement. This is not the case of Piggyback class of methods.
- Considers that the adjoint of the FP loop is a FP loop itself. This is not the case of Black Box class of methods.
- Computes $\overline{x}$ outside the adjoint loop. This is not the case of the Two-Phases approach.

$\Rightarrow$ We select the Refined Two-Phases approach to be implemented in our AD tool Tapenade.

Automatic Detection of Fixed-Point elements

- One directive to identify FP loop.
- State and parameters detected automatically from static data-flow analysis :

$$\begin{aligned} \text{state} &= \textbf{out}(\text{FP loop}) \cap \textbf{live} \\ \text{parameters} &= \textbf{use}(\text{FP loop}) \backslash \textbf{out}(\text{FP loop}) \end{aligned} \tag{4}$$

use : differentiable variables read by the FP loop
out : differentiable variables written by the FP loop
live : differentiable variables used in the sequel of FP loop

Implementation: Enable a repeated access to the stack

Do while z *not converged*

$$z = \phi(z,x)$$

end

$$z = \phi(z,x)$$

push(z)

$$y = f(z,x)$$

$$\overline{x} = \overline{y} * \frac{\partial f}{\partial x}(z,x)$$

$$\overline{z} = \overline{y} * \frac{\partial f}{\partial z}(z,x)$$

$$\overline{w} = \overline{z}$$

CALL start_repeat_stack()

Do while $\overline{w}$ *not converged*

pop(z)

$$\overline{w} = \overline{w} * \frac{\partial \phi}{\partial z}(z,x) + \overline{z}$$

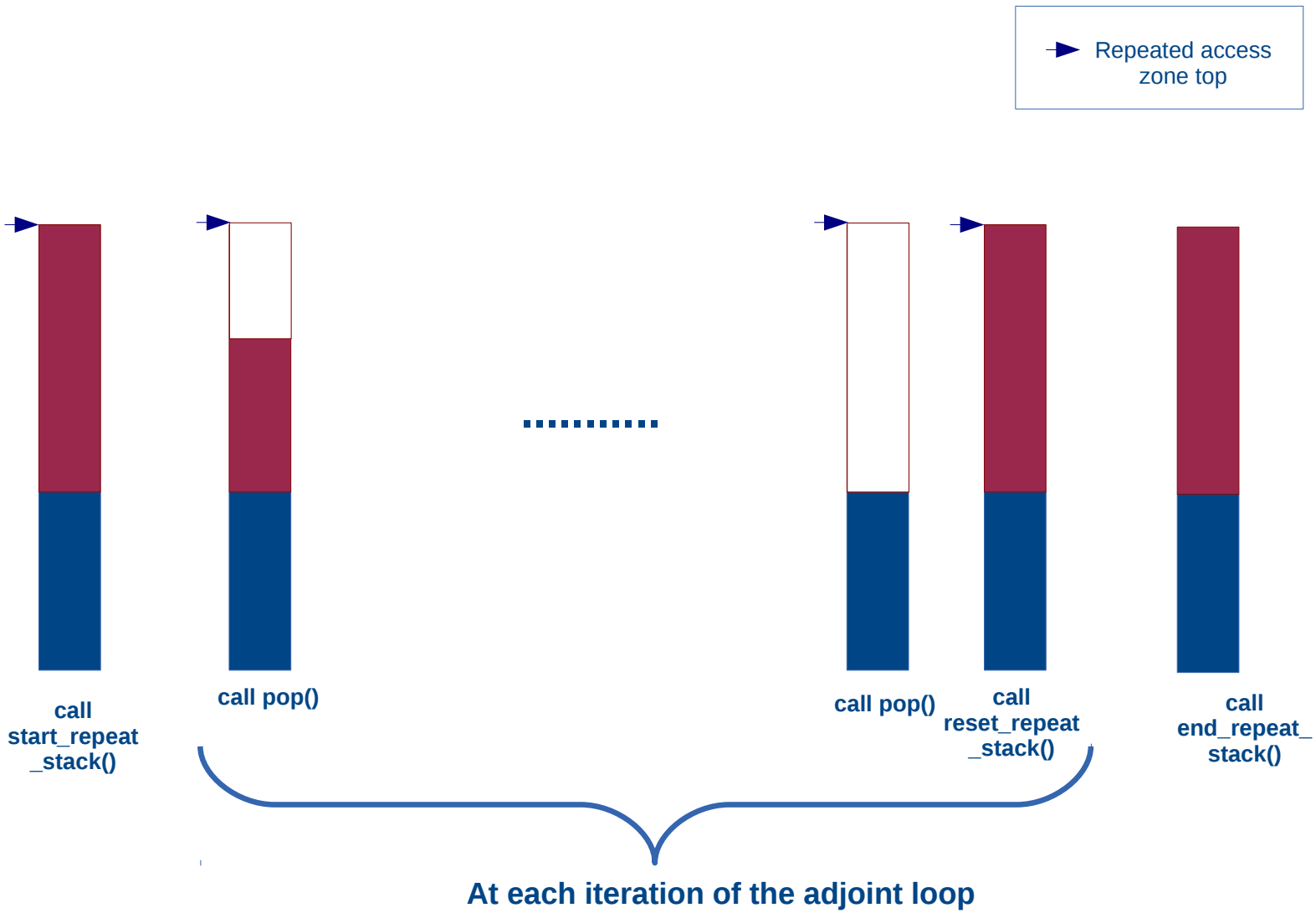
CALL reset_repeat_stack()

end

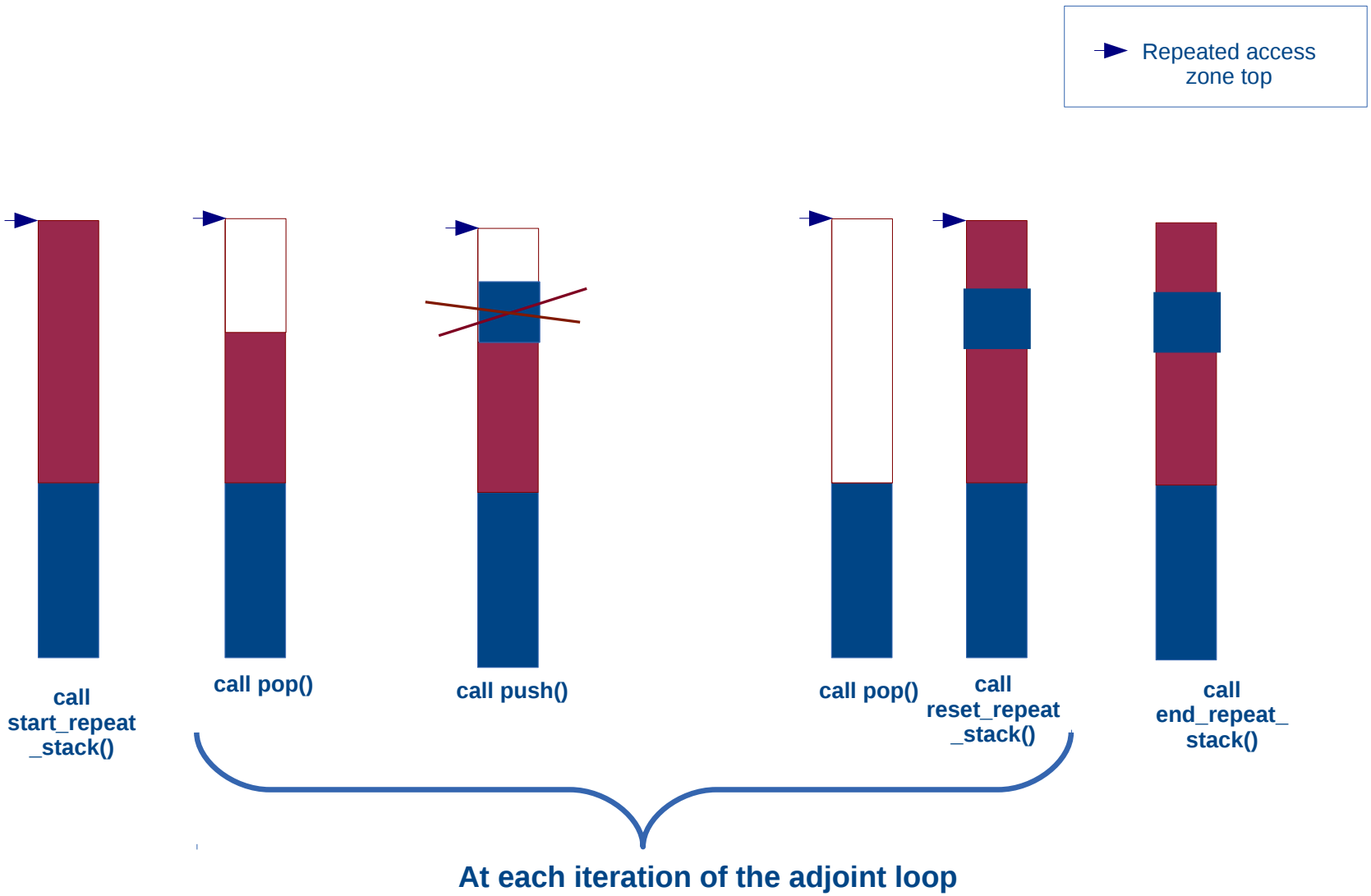
CALL end_repeat_stack()

$$\overline{x} = \overline{w} * \frac{\partial \phi}{\partial x}(z,x) + \overline{x}$$

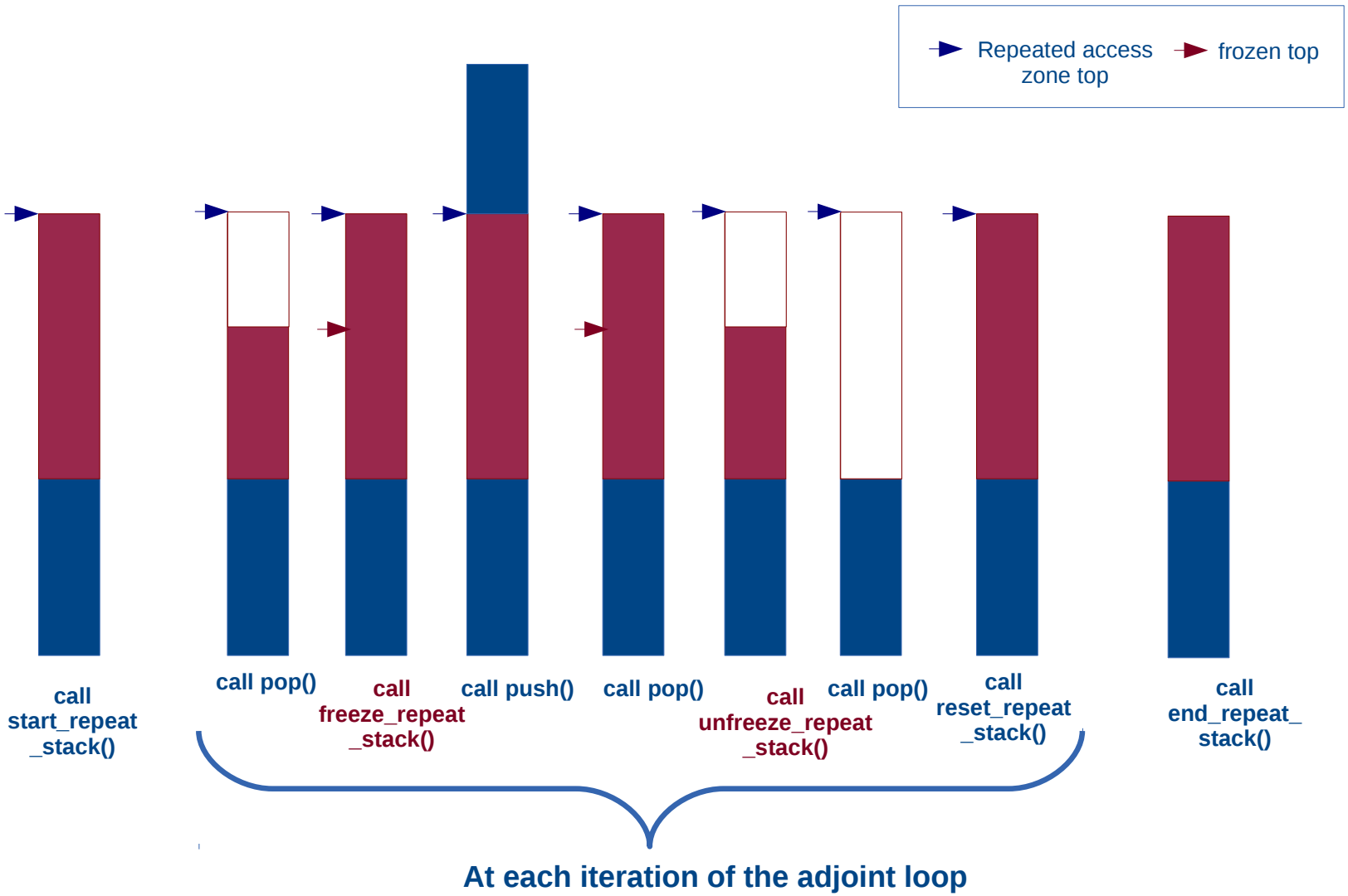
Extension of the stack mechanism



Extension of the stack mechanism



Extension of the stack mechanism



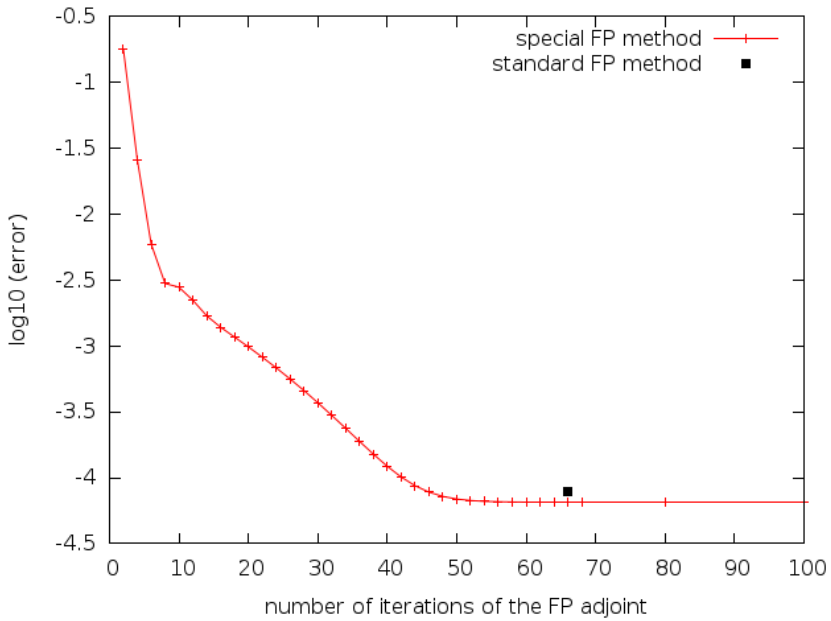
Experiment on real-medium size code

We validated our implementation on a medium-size code called GPDE:

- A Fortran90 program developed at Queen Mary University Of London (QMUL)
- An unstructured pressure-based steady-state Navier-Stokes solver with finite volume spatial discretization
- Contains a Fixed-Point loop that computes the pression and velocity (the state variables) of an incompressible flow.

Results

- Small improvement of the accuracy of the result



- Major reduction of the memory needed to store the trajectories.

	Two-Phases adjoint	Black Box adjoint
Memory Storage	10.1 MBytes	605.5 MBytes

Nested FP loops: constant intial guess for the inner loop

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

$\mathtt{z_{in}} = \mathtt{const}$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{z} = \mathtt{z_{in}}$

end

$\mathtt{y} = \mathtt{f}(\mathtt{z})$

	Two-Phases adjoint	Black Box adjoint
Number of iterations	10132	14491
Accuracy	$12 * 10^{-5}\%$	$2.1 * 10^{-5}\%$
Accuracy: 14491 iterations	$1.1 * 10^{-5}\%$	$2.1 * 10^{-5}\%$
Memory storage	268 bytes	86 Kbytes

Nested FP loops: smart intial guess for the inner loop

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{z} = \mathtt{z_{in}}$

end

$\mathtt{y} = \mathtt{f}(\mathtt{z})$

	Two-Phases adjoint	Black Box adjoint
Number of iterations	10132	8788
Accuracy	$12 * 10^{-5}\%$	$2 * 10^{-5}\%$
Memory storage	268 bytes	86 Kbytes

Black Box on a nested FP loops with constant intial guess

...

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

$\mathtt{z_{in}} = \mathtt{const}$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{push}(\mathtt{z_{in}})$

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{push}(\mathtt{z})$

$\mathtt{z} = \mathtt{z_{in}}$

end

...

Do "as many times"

$\mathtt{pop}(\mathtt{z})$

$\overline{\mathtt{z}}_{\mathtt{in}} = \overline{\mathtt{z}} + \overline{\mathtt{z}}_{\mathtt{in}}$; $\overline{z} = 0$

Do "as many times"

$$\overline{z}_{in} += \overline{z}_{in} * \frac{\partial Jacobi}{\partial z_{in}}(z_{in}, M, x)$$

$$\overline{x} += \overline{z}_{in} * \frac{\partial Jacobi}{\partial x}(z_{in}, M, x)$$

$\mathtt{pop}(\mathtt{z_{in}})$

end

$\overline{\mathtt{z}}_{\mathtt{in}} = 0$

$$\overline{z} += \overline{x} * \frac{\partial g}{\partial z}(z, F)$$

$$\overline{F} += \overline{x} * \frac{\partial g}{\partial F}(z, F)$$

end

Black Box on a nested FP loops with smart intial guess

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{push}(\mathtt{z_{in}})$

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{push}(\mathtt{z})$

$\mathtt{z} = \mathtt{z_{in}}$

end

...

...

Do "as many times"

$\mathtt{pop}(\mathtt{z})$

$\overline{\mathtt{z}}_{\mathtt{in}} = \overline{\mathtt{z}} + \overline{\mathtt{z}}_{\mathtt{in}}$; $\overline{z} = 0$

Do "as many times"

$$\overline{z}_{in} += \overline{z}_{in} * \frac{\partial Jacobi}{\partial z_{in}}(z_{in}, M, x)$$

$$\overline{x} += \overline{z}_{in} * \frac{\partial Jacobi}{\partial x}(z_{in}, M, x)$$

$\mathtt{pop}(\mathtt{z_{in}})$

end

$$\overline{z} += \overline{x} * \frac{\partial g}{\partial z}(z, F)$$

$$\overline{F} += \overline{x} * \frac{\partial g}{\partial F}(z, F)$$

end

Nested FP loops: smart intial guess for the inner loop

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{z} = \mathtt{z_{in}}$

end

$\mathtt{y} = \mathtt{f}(\mathtt{z})$

	Two-Phases adjoint	Black Box adjoint
Number of iterations	10132	8788
Accuracy	$12 * 10^{-5}\%$	$2 * 10^{-5}\%$
Memory storage	268 bytes	86 Kbytes

Two-Phases on a nested FP loops with smart intial guess

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{z} = \mathtt{z_{in}}$

end

last iteration + $\mathtt{store}(\mathtt{z}, \mathtt{z_{in}})$

...

Do while $\overline{\mathtt{w}}$ *not converged*

$\mathtt{restore}(\mathtt{z})$

$\overline{\mathtt{z}}_{\mathtt{in}} = \overline{\mathtt{w}} + \overline{\mathtt{z}}_{\mathtt{in}}$; $\overline{w} = 0$

$\overline{w}_{in} = \overline{z}_{in}$

Do $\overline{\mathtt{w}}_{\mathtt{in}}$ *not converged*

$\overline{w}_{in} += \overline{w}_{in} * \frac{\partial Jacobi}{\partial z_{in}}(z_{in}, M, x)$

$\overline{w}_{in} = \overline{w}_{in} + \overline{z}_{in}$; $\mathtt{restore}(\mathtt{z_{in}})$

end

$\overline{x} += \overline{w}_{in} * \frac{\partial Jacobi}{\partial x}(z_{in}, M, x)$

$\overline{w} += \overline{x} * \frac{\partial g}{\partial z}(z, F)$

$\overline{w} = \overline{w} + \overline{z}$

end

$\overline{F} += \overline{x} * \frac{\partial g}{\partial F}(z, F)$

Two-Phases with smart intial guess for the inner adjoint

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{z} = \mathtt{z_{in}}$

end

last iteration + $\mathtt{store}(\mathtt{z}, \mathtt{z_{in}})$

...

Do while $\overline{\mathtt{w}}$ *not converged*

$\mathtt{restore(z)}$

$\overline{z}_{\mathtt{in}} = \overline{\mathtt{w}} + \overline{z}_{\mathtt{in}}$; $\overline{w} = 0$

$\overline{w}_{in} = \overline{z}_{in} + \overline{w}_{inConv}$

Do $\overline{\mathtt{w}}_{\mathtt{in}}$ *not converged*

$\overline{w}_{in} += \overline{w}_{in} * \frac{\partial Jacobi}{\partial z_{in}}(z_{in}, M, x)$

$\overline{w}_{in} = \overline{w}_{in} + \overline{z}_{in}$; $\mathtt{restore}(\mathtt{z_{in}})$

end

$\overline{x} += \overline{w}_{in} * \frac{\partial Jacobi}{\partial x}(z_{in}, M, x)$

$\overline{w}_{inConv} = \overline{w}_{in} * \frac{\partial Jacobi}{\partial z_{in}}(z_{in}, M, x)$

$\overline{w} += \overline{x} * \frac{\partial g}{\partial z}(z, F)$; $\overline{w} = \overline{w} + \overline{z}$

end

$\overline{F} += \overline{x} * \frac{\partial g}{\partial F}(z, F)$

Nested FP loops: smart intial guess for the inner loop

Do while $\mathtt{z}$ *not converged*

$\mathtt{x} = \mathtt{g}(\mathtt{z}, \mathtt{F})$

Do while $\mathtt{z_{in}}$ *not converged*

$\mathtt{z_{in}} = \mathtt{Jacobi}(\mathtt{z_{in}}, \mathtt{M}, \mathtt{x})$

end

$\mathtt{z} = \mathtt{z_{in}}$

end

$\mathtt{y} = \mathtt{f}(\mathtt{z})$

	Two-Phases with smart initial	Two-Phases	Black Box
Number of iterations	5219	10132	8788
Accuracy	$10 * 10^{-5}\%$	$12 * 10^{-5}\%$	$2 * 10^{-5}\%$
Accuracy for 8788 iterations	$0.98 * 10^{-5}\%$	$340 * 10^{-5}\%$	$2 * 10^{-5}\%$

Summary

- We are seeking to improve performance of adjoint codes produced by the adjoint mode of Automatic Differentiation
- We explained why we consider the strategy initially proposed by Christianson as the best suited for our needs.
- We described the way we implemented this strategy in Tapenade.
- We experimented this strategy on a some a real medium size code and quantified its benefits.
- We studied the related question of the initial guess in the case of nested FP iterations.

Further Work

- The restrictions we imposed on the structure of Fixed-Point loops can be lifted
- Study the choice of the intial guess of the adjoint loop in the general case of nested structures of FP loops.
- Derive the stopping criterion of the adjoint loop form the original loop's stopping criterion

Acknowledgements

This work has been conducted within the **About Flow** project on "Adjoint-based optimisation of industrial and unsteady flows".

`http://aboutflow.sems.qmul.ac.uk`

About Flow has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under Grant Agreement No. 317006.

Rolls-Royce engys esi get it right Inria informatics mathematics Queen Mary University of London RWTH AACHEN UNIVERSITY National Technical University of Athens