The data-flow equations of Checkpointing in Reverse Automatic Differentiation

Benjamin Dauvergne, Laurent Hascoët

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

ICCS'2006, Reading, UK

Context: Reverse Automatic Differentiation

- **gradients !**
  Reverse AD by program transformation
  ($\Rightarrow$ opportunity for data-flow analysis: activity,. . .)
- **reversed data flow !**
  Store-All approach ($\Rightarrow$ needs optimized taping, TBR)
- **memory (tape) size !**
  Nested Checkpointing
  ($\Rightarrow$ repeated executions, Snapshots)

For the Data Flow Equations of these analyses, we need formal proofs of "correctness" and "optimality".

Checkpointing questions

When **no** checkpointing is done:

- Unique optimal Equations for activity, (adjoint-)liveness, TBR
- Data Flow Equations derived formally
- No retroaction between analyses:
  1) activity, *2)* adjoint-liveness, *3)* TBR

Our goal

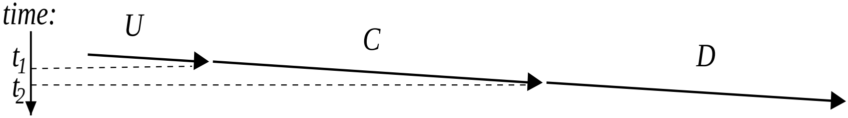
But when checkpointing **is** present:

- Still no problem for activity and adj-liveness
- Examples show many optimal answers for TBR/Snapshots
- Retroaction between TBR and Snapshot

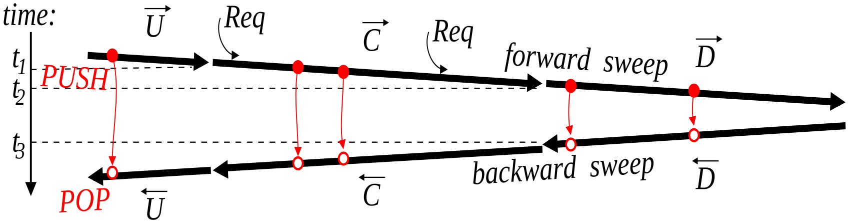
$\Rightarrow$ Our goal is to characterize all possible "optimal" strategies for TBR/Snapshot, and then experiment some of them on real applications.

Reverse AD without Checkpointing

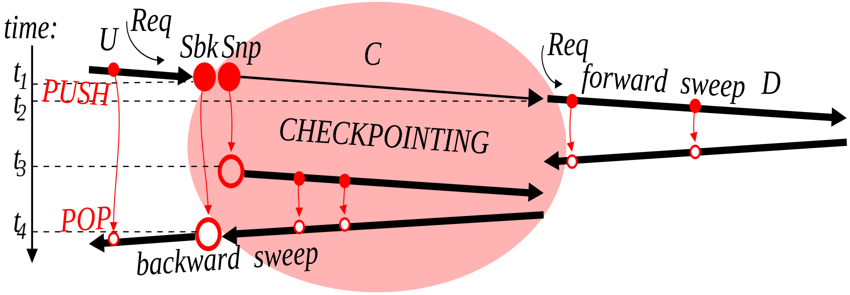
- Original program $U; C; D$:



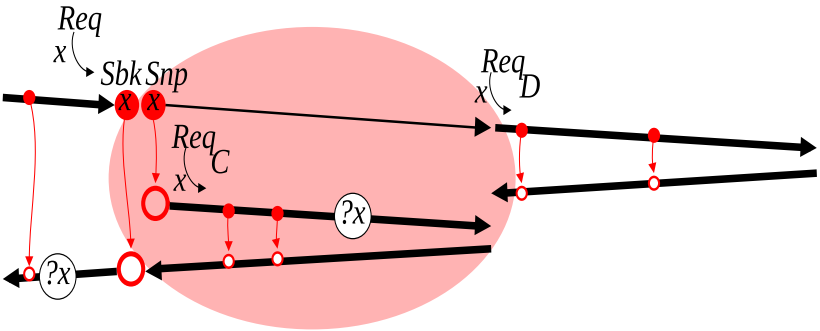
- Reverse diff program, no Checkpointing:



- Reverse diff program, with Checkpointing on C:



The retroaction problem



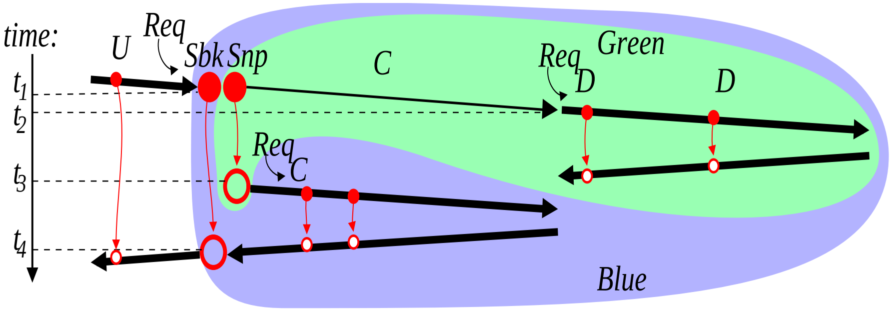
Variable x needs to be saved . . .

- either because required in $\overleftarrow{U}$ (TBR) $\Rightarrow x \in Sbk$
- or because used in $\overline{C}$ (Checkpointing) $\Rightarrow x \in Snp$

. . . but if $x \in Snp$ and $x \notin$ **out**$(\overline{C})$, then $x \notin Sbk$!

$\Rightarrow$ We have to be more systematic

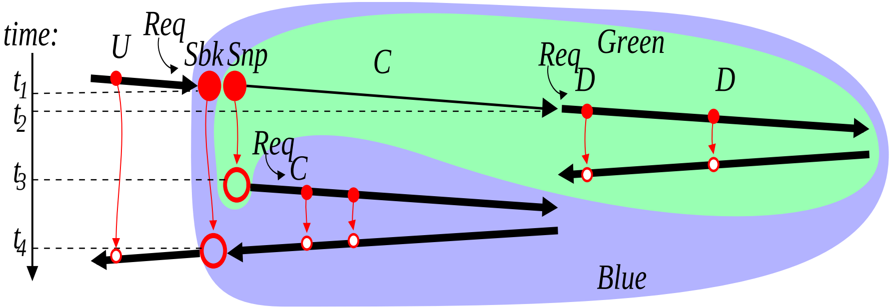
Necessary and sufficient constraints



- $\mathbf{out}(\mathit{Green}) \bigcap \mathbf{use}(\overline{C}) = \emptyset$
- $\mathbf{out}(\mathit{Blue}) \bigcap \mathrm{Req} = \emptyset$

From now on, constraints on *Snp*, *Sbk*, Req_D, and Req_C follow mechanically !

Developping the **out** sets



$$\mathbf{out}(\mathit{Green}) = (\mathbf{out}(C) \cup (\mathbf{out}(\overline{D}) \setminus Req_D)) \setminus Snp$$

$$\mathbf{out}(\mathit{Blue}) = ((\mathbf{out}(C) \cup (\mathbf{out}(\overline{D}) \setminus Req_D)) \setminus Snp \\ \cup(\mathbf{out}(\overline{C}) \setminus Req_C)) \setminus Sbk$$

Equations for the minimal solutions

$$
\begin{aligned}
Sbk &\supseteq ((\mathbf{out}(C) \cup (\mathbf{out}(\overline{D}) \setminus Req_D)) \setminus Snp \\
&\qquad\qquad\qquad \cup\ (\mathbf{out}(\overline{C}) \setminus Req_C)) \cap Req \\
Snp &\supseteq (\mathbf{out}(C) \cup (\mathbf{out}(\overline{D}) \setminus Req_D)) \cap \\
&\qquad\qquad\qquad (\mathbf{use}(\overline{C}) \cup (Req \setminus Sbk)) \\
Req_D &\supseteq (\mathbf{out}(\overline{D}) \setminus Snp) \cap (\mathbf{use}(\overline{C}) \cup (Req \setminus Sbk)) \\
Req_C &\supseteq (\mathbf{out}(\overline{C}) \setminus Sbk) \cap Req
\end{aligned}
$$

- Retroaction is now apparent
- Hand resolution is error-prone

  $\Rightarrow$ use a symbolic computation tool

. . . for instance Maple

```
paprika$ maple
    |\^/|     Maple V Release 5 (INRIA)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(logic) ;
[bequal, bsimp, canon, convert/MOD2, convert/frominert, convert/toinert, distrib,
    dual, environ, randbool, satisfy, tautology]

> Snp := bsimp((outC &or (outDb &and &not(ReqD))) &and (useCb &or (Req \
> &and &not(Sbk)))) ;

Snp := &or(&and(useCb, &not(ReqD), outDb),
    &and(Req, &not(Sbk), &not(ReqD), outDb), outC &and useCb,
    &and(outC, Req, &not(Sbk)))

> Sbk := bsimp((((outC &or (outDb &and &not(ReqD))) &and &not(Snp)) &or\
> ((outC &and outCb) &and &not(ReqC))) &and Req) ;

Warning, recursive definition of name
Sbk := &or(&and(outC, outCb, Req, &not(ReqC)), &and(outC, Sbk, Req, &not(useCb)),
    &and(Sbk, Req, &not(ReqD), outDb, &not(useCb)))
```

The minimal solutions

Define:

$$\begin{aligned}
Snp_0 &= \mathbf{out}(C) \cap (\mathbf{use}(\overline{C}) \cup (Req \setminus \mathbf{out}(\overline{C}))) \\
Opt_1 &= Req \cap \mathbf{out}(\overline{C}) \cap \mathbf{use}(\overline{C}) \\
Opt_2 &= Req \cap \mathbf{out}(\overline{C}) \setminus \mathbf{use}(\overline{C}) \\
Opt_3 &= \mathbf{out}(\overline{D}) \cap (\mathbf{use}(\overline{C}) \cup Req) \setminus \mathbf{out}(C)
\end{aligned}$$

Every minimal solution is of the form:

$$\begin{array}{lcccccc}
Sbk & = & Opt_1^+ & \cup & Opt_2^+ & & \\
Snp & = & Snp_0 & \cup & Opt_2^- & \cup & Opt_3^+ \\
Req_D & = & & & & & Opt_3^- \\
Req_C & = & Opt_1^- & \cup & Opt_2^- & &
\end{array}$$

Eager: *save now$^+$ in Snp* **vs** **Lazy:** *delayed$^-$ for TBR*

"Eager Snapshots"

Take $Opt_1^+ = Opt_1$, $Opt_2^+ = Opt_2$, and $Opt_3^+ = Opt_3 \Rightarrow$

$$
\begin{array}{rcl}
Sbk & = & Req \cap \mathbf{out}(\overline{C}) \\
Snp & = & (Req \cap \mathbf{out}(\overline{D}) \setminus \mathbf{out}(C)) \\
 & & \cup (Req \cap \mathbf{out}(C) \setminus \mathbf{out}(\overline{C})) \\
 & & \cup (\mathbf{use}(\overline{C}) \cap \mathbf{out}(\overline{D})) \cup (\mathbf{use}(\overline{C}) \cap \mathbf{out}(C)) \\
Req_D & = & \emptyset \\
Req_C & = & \emptyset
\end{array}
$$

- Need $\mathbf{out}(\overline{C}), \mathbf{out}(\overline{D})$
- Snapshot anticipates TBR $\Rightarrow$ rarely good...

"Lazy Snapshots"

Take $Opt_1^+ = \emptyset$, $Opt_2^+ = \emptyset$, and $Opt_3^+ = \emptyset \Rightarrow$

$$
\begin{array}{lcl}
Sbk & = & \emptyset \\
Snp & = & \mathbf{out}(C) \cap (Req \cup \mathbf{use}(\overline{C})) \\
Req_D & = & \mathbf{out}(\overline{D}) \cap (Req \cup \mathbf{use}(\overline{C})) \setminus \mathbf{out}(C) \\
Req_C & = & \mathbf{out}(\overline{C}) \cap Req
\end{array}
$$

- Saves are delayed until the very last moment
- No need for $\mathbf{out}(\overline{C}), \mathbf{out}(\overline{D})$
- Best strategy in general,
  except for special (contrived) cases.

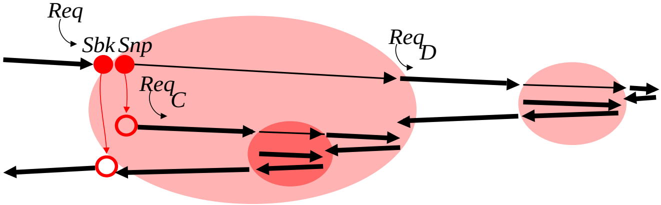
Experimental Measurements

Code	Domain	Time	adj.T.	Eager	Lazy
OPA	*oceanogr.*	110 s	780 s	480 Mb	479 Mb
STICS	*agronomy*	0.23 s	1.82 s	80 Mb	80 Mb
UNS2D	*CFD*	2.7 s	23 s	248 Mb	185 Mb
SAIL	*agronomy*	5.6 s	17 s	1.6 Mb	1.5 Mb
THYC	*thermodyn.*	2.7 s	12 s	33.7 Mb	18.3 Mb
LIDAR	*optics*	4.3 s	10 s	14.6 Mb	14.6 Mb
CURVE	*shape optim.*	0.7 s	2.7 s	1.44 Mb	0.59 Mb
SONIC	*CFD*	0.03 s	0.2 s	3.55 Mb	2.02 Mb
Contrived example		0.02 s	0.1 s	8.20 Mb	11.71 Mb

Lazy snapshots never loose on real applications.
Gain is less visible on long iterative programs.

Nested Checkpoints

What is the relative influence of nested checkpoints?



Optimal sets depend on $\mathbf{use}(\overline{C}), \mathbf{out}(\overline{C}), \mathbf{out}(\overline{D})$.
Does $\mathbf{out}(\overline{P})$ depend on the checkpoints inside P?

(Maple:) $\Rightarrow$ whatever the choice of $Opt_1^+, Opt_2^+, Opt_3^+$, the value of $\mathbf{out}(\overline{C;D})$ is the same:

$$\mathbf{out}(\overline{C;D}) = (\mathbf{out}(C) \cup \mathbf{out}(\overline{D})) \cap \mathbf{out}(\overline{C}) \setminus \mathbf{use}(\overline{C}) \setminus \mathrm{Req}$$

Conclusion

- "Minimal" Snp, Req_C, and Req_D sets.
- "Lazy" strategy best in most cases.

$\Rightarrow$ Future directions:

- Measurements should look not only at memory peak, but also at memory traffic.
- Adaptive choice of Opt_1^+, Opt_2^+, Opt_3^+, different for each checkpoint.
- Try activation and disactivation of checkpoints based on the data-flow **use** and **out** sets.
- Try moving the boundaries of the checkpoints C.