

Outline

- 1 Tuning the adjoint of an unknown code
- 2 User driven placement of checkpoints
- 3 Adjoint code profiling
- 4 Conclusion

Outline

- 1 Tuning the adjoint of an unknown code
- 2 User driven placement of checkpoints**
- 3 Adjoint code profiling
- 4 Conclusion

Example: STICS

- STICS is an agronomy modeling program.
- 21.010 LOC, and 46.921 LOC generated in the adjoint.

Problem: very bad performance using the default (checkpoint-all) strategy, **100 x ORIG**

Example: STICS

- STICS is an agronomy modeling program.
- 21.010 LOC, and 46.921 LOC generated in the adjoint.

Problem: very bad performance using the default (checkpoint-all) strategy, **100 x ORIG**

NOCHECKPOINT directive

We introduce a new feature in Tapenade.

For each subroutine call, decide whether to checkpoint it or not:

- call site per call site
- globally for one procedure
- alternatively with a `-nocheckpoint` command line option

NOCHECKPOINT directive

We introduce a new feature in Tapenade.

For each subroutine call, decide whether to checkpoint it or not:

- call site per call site
- globally for one procedure
- alternatively with a `-nocheckpoint` command line option

NOCHECKPOINT directive

We introduce a new feature in Tapenade.

For each subroutine call, decide whether to checkpoint it or not:

- call site per call site
- globally for one procedure
- alternatively with a `-nocheckpoint` command line option

Experiments on STICS

Experiment		Time		Memory	
Id	Description	Total [s]	% gain	Peak [Mb]	% gain
01	Checkpoint-all strategy	38.56		229.23	
02	NOCHECKPOINT BIOMAER	36.15	6.3	229.23	0.0
03	NOCHECKPOINT MINERAL	35.78	7.2	229.28	0.0
04	NOCHECKPOINT DENSIRAC	30.02	22.1	229.23	0.0
05	NOCHECKPOINT CROIRA	24.45	36.6	229.23	0.0
06	NOCHECKPOINT ONEBIGLOOP	23.75	38.4	229.75	-0.2
07	04 and 05	16.79	56.5	229.23	0.0
08	04 and 06	15.64	59.4	229.75	-0.2
08	05 and 06	11.71	69.6	206.81	9.8
09	04, 05 and 06	3.93	89.8	149.11	34.9
09	03, 04, 05 and 06	3.92	89.8	149.11	34.9
09	01 to 06	3.90	89.9	149.11	34.9

Outline

- 1 Tuning the adjoint of an unknown code
- 2 User driven placement of checkpoints
- 3 Adjoint code profiling**
- 4 Conclusion

Why a procedure should not be checkpointed ?

- On STICS, candidate subroutines have **snapshot » tape**.
- On other examples, deeply nested calltrees need not be checkpointed systematically.
- Other heuristics. . .

Profiling required!

Why a procedure should not be checkpointed ?

- On STICS, candidate subroutines have **snapshot » tape**.
- On other examples, deeply nested calltrees need not be checkpointed systematically.
- Other heuristics...

Profiling required!

Why a procedure should not be checkpointed ?

- On STICS, candidate subroutines have **snapshot » tape**.
- On other examples, deeply nested calltrees need not be checkpointed systematically.
- Other heuristics. . .

Profiling required!

What to profile?

For each procedure, average and standard deviation of:

- size of snapshot and local tape
- time of snapshot
- time of original, forward and backward procedure
- number of duplicate executions

What to profile?

For each procedure, average and standard deviation of:

- size of snapshot and local tape
- time of snapshot
- time of original, forward and backward procedure
- number of duplicate executions

What to profile?

For each procedure, average and standard deviation of:

- size of snapshot and local tape
- time of snapshot
- time of original, forward and backward procedure
- number of duplicate executions

What to profile?

For each procedure, average and standard deviation of:

- size of snapshot and local tape
- time of snapshot
- time of original, forward and backward procedure
- number of duplicate executions

Implementation of profiling

The `-profile` option adds instrumenting code in the adjoint code. This code monitors tape size and execution time for these events:

- beginning of a snapshot
- end of a snapshot
- beginning of original, forward and backward procedure
- end of original, forward and backward procedure

Implementation of profiling

The `-profile` option adds instrumenting code in the adjoint code. This code monitors tape size and execution time for these events:

- beginning of a snapshot
- end of a snapshot
- beginning of original, forward and backward procedure
- end of original, forward and backward procedure

Implementation of profiling

The `-profile` option adds instrumenting code in the adjoint code. This code monitors tape size and execution time for these events:

- beginning of a snapshot
- end of a snapshot
- beginning of original, forward and backward procedure
- end of original, forward and backward procedure

Implementation of profiling

The `-profile` option adds instrumenting code in the adjoint code. This code monitors tape size and execution time for these events:

- beginning of a snapshot
- end of a snapshot
- beginning of original, forward and backward procedure
- end of original, forward and backward procedure

Profiling results: STICS

Name	Average tape	Average snap	SNAP / TAPE
humirac_b	32	0.43764	0.013676
biomaer_b	125.22	1692.6	13.517
densirac_b	43474	1.465e+06	33.699
fertil_b	40.117	1411.4	35.183
croira_b	515.46	1.8456e+05	358.05
onebigloopiter_b	660.48	2.5995e+05	393.58
n1bigloopiters_b	118.86	3.0046e+06	25279

Exploiting the profiling results

The result of profiling on STICS confirms our intuitive checkpointing placements.

Profiling suggests additional candidates:

⇒ an additional 15 % gain

Further work

- Integrate to AD process as much as possible
- Look at the behaviour of the code with respect to the cache
- Extend possible placement of checkpoint to loops (treeverse, dichotomy, ...)

Further work

- Integrate to AD process as much as possible
- Look at the behaviour of the code with respect to the cache
- Extend possible placement of checkpoint to loops (treeverse, dichotomy, ...)

Further work

- Integrate to AD process as much as possible
- Look at the behaviour of the code with respect to the cache
- Extend possible placement of checkpoint to loops (treeverse, dichotomy, ...)