Debugging and optimization of HPC programs with the Verrou tool

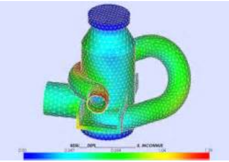
Software Correctness
for HPC Applications
18/11/2019

François Févotte
Bruno Lathuilière*

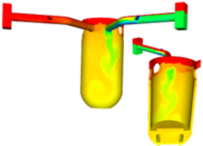
EDF R&D
PERICLES / I23
Analysis and Numerical Modeling

Industrial context – Numerical Verification

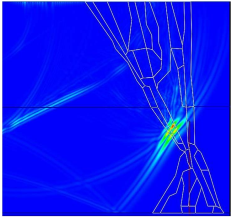
In-house development of Scientific Computing Codes



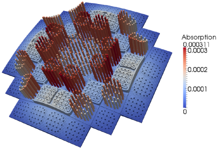
Structures



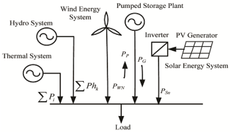
Fluid dynamics



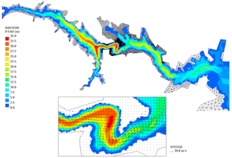
Wave propagation



Neutronics



Power Systems



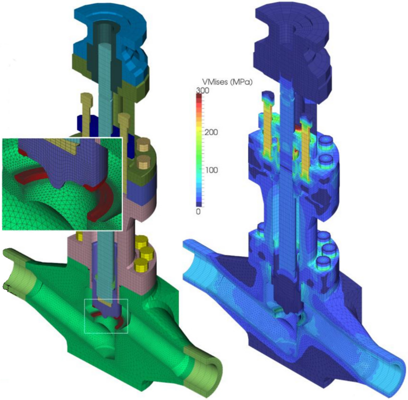
Free surface hydraulics

Industrial context – Numerical Verification

Code_aster

Mechanics

- Seismic
- Acoustic
- Thermo-mechanics



Code_Aster

- 1.5M code lines
- Fortran 90, C, Python
- thousands of test cases
- Large number of dependencies :
  - Linear solvers (MUMPS. . .)
  - Mesh generator and partitioning tools (Metis, Scotch. . .)

Goals

- understand the non-reproducibility between test computers

Industrial context – Numerical Verification

Objectives / presentation outline

Diagnostics

- verify a code / show the presence of FP-related errors
- quantify the magnitude of issues

Debugging

- locate the origin of FP-related issues in the source code
  - unstable algorithms
  - unstable tests
- track the origin of issues during program execution
  - context of calls
  - temporal information (*e.g.* iteration number...)

Optimization

- use mixed-precision implementations

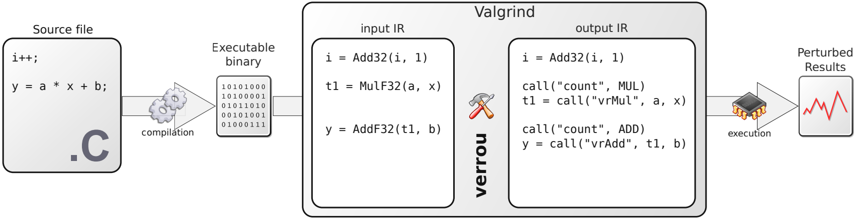
Available on github (latest version: v2.1.0)
http://github.com/edf-hpc/verrou

Documentation:
http://edf-hpc.github.io/verrou/vr-manual.html

Industrial context – Numerical Verification

Dynamic binary analysis with Valgrind

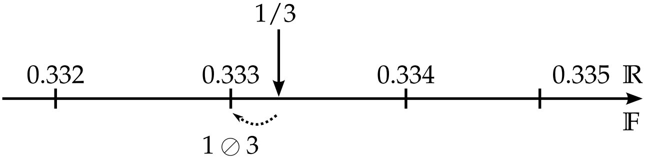
```
$ valgrind --tool=verrou [VERROU_ARGS]    PROGRAM [ARGS...]
```



Diagnostics: detect and assess instabilities

Verrou back-end: random rounding

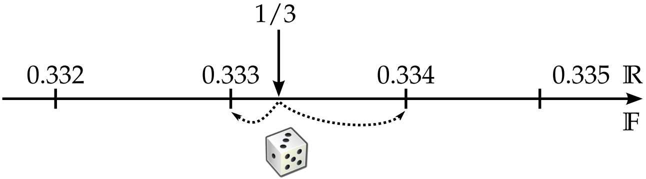
IEEE-754: nearest rounding mode



Diagnostics: detect and assess instabilities

Verrou back-end: random rounding

CESTAC: random rounding mode `--rounding-mode=random`



Basic use

Run the code:

```
$ PROG='python -c "print(1+10**6-sum([0.1 for i in range(10**7)]))"'
$ eval $PROG
1.00016102463
```

Verify the instrumentation process:

```
$ CMD="valgrind --tool=verrou --rounding-mode=nearest $PROG"
$ eval "$CMD"  2>/dev/null
1.00016102463
```

Run stochastic samples:

```
$ CMD="valgrind --tool=verrou --rounding-mode=random $PROG"
$ eval "$CMD; $CMD; $CMD"  2>/dev/null
0.999830178451
0.999830617453
0.999830234447
```

Post-process:

$$\hat{s} = -\log_2\left(\frac{\max|X_i - X_{\text{ieee}}|}{|X_{\text{ieee}}|}\right)$$

$\approx$ 11.6 significant bits

(3.5 significant digits)

Really few false positive

Libm is the noticeable exception

Issue

```
$ PROG='python -c "import math;print(math.cos(42.))"'

$ CMD="valgrind --tool=verrou --rounding-mode=random $PROG"

$ eval "$CMD; $CMD; $CMD"  2>/dev/null
-0.399985314988
-0.399985314988
-1.00505077023
```

Baseline: ignore FP operation inside libm

```
$ cat libm.ex
* /lib/x86_64-linux-gnu/libm-2.19.so

$ CMD="valgrind --tool=verrou --rounding-mode=random --exclude=libm.ex $PROG"

$ eval "$CMD; $CMD; $CMD"  2>/dev/null
-0.399985314988
-0.399985314988
-0.399985314988
```

Interlibm

An interposition library

Use:

```
VERROU_ROUNDING_MODE=random LD_PRELOAD=$VERROU_LIBM $PROG
```

Implementation for each libm function:

- To reuse the verrou rounding mode simulation we need:
  - the round to nearest result
  - and an estimation of error

The implementation uses the estimation provided by libquadmath.

Exemple extracted from code_aster

for $a = 4.208003496301644 \times 10^{-5}$ and $b = a + 6\,ulp(a)$.

$$f(a,b) = \frac{b-a}{\log(b)-\log(a)}$$

Number of significant bits for both implementations.

		$\frac{b-a}{\log(b)-\log(a)}$	$a\,\frac{\frac{b}{a}-1}{\log(\frac{b}{a})}$
	IEEE Error	16.60	53.00
(i)	interlibm	14.12	52.46
(ii)	verrou	52.46	51.46
(iii)	verrou+interlibm	14.12	50.88

Estimation with 459 samples and following estimator :

$$\hat{s} = -\log_2\left(\frac{\max|X_i - X_{\text{ieee}}|}{|X_{\text{ieee}}|}\right). \tag{1}$$

Plan

1. Diagnostics
2. Debugging
3. Conclusions – perspectives

Debugging: code coverage

Instable tests detection

```
$ make CFLAGS="-fprofile-arcs -ftest-coverage"
$ make check
$ gcov *.c *.f
```

"standard" coverage

```
120:subroutine fun1(area, a1, a2, n)
  -:    implicit none
  -:    integer :: n
  -:    real(kind=8) :: area, a1, a2
120:    if (a1 .eq. a2) then
 13:        area = a1
  -:    else
107:        if (n .lt. 2) then
107:            area = (a2-a1) / (log(a2)-log(a1))
###:        else if (n .eq.2) then
###:            area = sqrt (a1*a2)
  -:        else
###:            ! ...
  -:        endif
  -:    endif
120:end subroutine
```

Debugging: code coverage

Instable tests detection

```
$ make CFLAGS="-fprofile-arcs -ftest-coverage"
$ make check
$ gcov *.c *.f
```

"standard" coverage

```
120:subroutine fun1(area, a1, a2, n)
  -:    implicit none
  -:    integer :: n
  -:    real(kind=8) :: area, a1, a2
120:    if (a1 .eq. a2) then
 13:        area = a1
  -:    else
107:        if (n .lt. 2) then
107:            area = (a2-a1) / (log(a2)-log(a1))
###:        else if (n .eq.2) then
###:            area = sqrt (a1*a2)
  -:        else
###:            ! ...
  -:        endif
  -:    endif
120:end subroutine
```

"Verrou" coverage

```
120:subroutine fun1(area, a1,...
  -:    implicit none
  -:    integer :: n
  -:    real(kind=8) :: area,...
120:    if (a1 .eq. a2) then
  4:        area = a1
  -:    else
116:        if (n .lt. 2) then
116:            area = (a2-a1...
###:        else if (n .eq.2)...
###:            area = sqrt (...
  -:        else
###:            ! ...
  -:        endif
  -:    endif
120:end subroutine
```

Debugging: locate issues in the source code

Delta-debugging

```
log.L               .../aster.release
volum2_             .../aster.release
bilpla_             .../aster.release
ecrval_             .../aster.release
print_plath_        .../aster.release
classer_groupes_    .../aster.release
etupla_             .../aster.release
couhyd_pi_          .../aster.release
ecrplr_             .../aster.release
imovi_              .../aster.release
resopt_             .../aster.release
getgrp_marginal_    .../aster.release
ecrpla_             .../aster.release
fin_exec_main_      .../aster.release
decopt_pi_          .../aster.release
paraend_            .../aster.release
resopt_cnt_zones_   .../aster.release
apstop_             .../aster.release
ihyd_               .../aster.release
impression_info_    .../aster.release
coupla_             .../aster.release
gere_print_plath_   .../aster.release
log                 .../aster.release
thepla_             .../aster.release
coutot_             .../aster.release
iprit_              .../aster.release
```

- relies on the Verrou ability to restrict the scope of instrumentation/perturbations

→ ✓

Debugging: locate issues in the source code

Delta-debugging

```
# log.L                 .../aster.release
# volum2_               .../aster.release
# bilpla_               .../aster.release
# ecrval_               .../aster.release
# print_plath_          .../aster.release
# classer_groupes_      .../aster.release
# etupla_               .../aster.release
# couhyd_pi_            .../aster.release
# ecrplr_               .../aster.release
# imovi_                .../aster.release
# resopt_               .../aster.release
# getgrp_marginal_      .../aster.release
# ecrpla_               .../aster.release
# fin_exec_main_        .../aster.release
# decopt_pi_            .../aster.release
# paraend_              .../aster.release
# resopt_cnt_zones_     .../aster.release
# apstop_               .../aster.release
# ihyd_                 .../aster.release
# impression_info_      .../aster.release
# coupla_               .../aster.release
# gere_print_plath_     .../aster.release
# log                   .../aster.release
# thepla_               .../aster.release
# coutot_               .../aster.release
# iprit_                .../aster.release
```

- relies on the Verrou ability to restrict the scope of instrumentation/perturbations

$\longrightarrow$ $\times$

Debugging: locate issues in the source code

Delta-debugging

```
# log.L                .../aster.release
# volum2_              .../aster.release
# bilpla_              .../aster.release
# ecrval_              .../aster.release
# print_plath_         .../aster.release
# classer_groupes_     .../aster.release
# etupla_              .../aster.release
# couhyd_pi_           .../aster.release
# ecrplr_              .../aster.release
# imovi_               .../aster.release
# resopt_              .../aster.release
# getgrp_marginal_     .../aster.release
# ecrpla_              .../aster.release
fin_exec_main_         .../aster.release
decopt_pi_             .../aster.release
paraend_               .../aster.release
resopt_cnt_zones_      .../aster.release
apstop_                .../aster.release
ihyd_                  .../aster.release
impression_info_       .../aster.release
coupla_                .../aster.release
gere_print_plath_      .../aster.release
log                    .../aster.release
thepla_                .../aster.release
coutot_                .../aster.release
iprit_                 .../aster.release
```

- relies on the Verrou ability to restrict the scope of instrumentation/perturbations
- Delta-Debugging [A. Zeller, 1999] adapted for stochastic evaluation

$\rightarrow$ ✗

Debugging: locate issues in the source code

Delta-debugging

```
# log.L               .../aster.release
# volum2_             .../aster.release
# bilpla_             .../aster.release
# ecrval_             .../aster.release
# print_plath_        .../aster.release
# classer_groupes_    .../aster.release
# etupla_             .../aster.release
couhyd_pi_            .../aster.release
ecrplr_               .../aster.release
imovi_                .../aster.release
resopt_               .../aster.release
getgrp_marginal_      .../aster.release
ecrpla_               .../aster.release
fin_exec_main_        .../aster.release
decopt_pi_            .../aster.release
paraend_              .../aster.release
resopt_cnt_zones_     .../aster.release
apstop_               .../aster.release
ihyd_                 .../aster.release
impression_info_      .../aster.release
coupla_               .../aster.release
gere_print_plath_     .../aster.release
log                   .../aster.release
thepla_               .../aster.release
coutot_               .../aster.release
iprit_                .../aster.release
```

- relies on the Verrou ability to restrict the scope of instrumentation/perturbations
- Delta-Debugging [A. Zeller, 1999] adapted for stochastic evaluation

$\longrightarrow$ ✓

Debugging: locate issues in the source code

Delta-debugging

```
log.L                  .../aster.release
volum2_                .../aster.release
bilpla_                .../aster.release
ecrval_                .../aster.release
print_plath_           .../aster.release
classer_groupes_       .../aster.release
etupla_                .../aster.release
# couhyd_pi_           .../aster.release
# ecrplr_              .../aster.release
# imovi_               .../aster.release
# resopt_              .../aster.release
# getgrp_marginal_     .../aster.release
# ecrpla_              .../aster.release
fin_exec_main_         .../aster.release
decopt_pi_             .../aster.release
paraend_               .../aster.release
resopt_cnt_zones_      .../aster.release
apstop_                .../aster.release
ihyd_                  .../aster.release
impression_info_       .../aster.release
coupla_                .../aster.release
gere_print_plath_      .../aster.release
log                    .../aster.release
thepla_                .../aster.release
coutot_                .../aster.release
iprit_                 .../aster.release
```

- relies on the Verrou ability to restrict the scope of instrumentation/perturbations
- Delta-Debugging [A. Zeller, 1999] adapted for stochastic evaluation

$\rightarrow$ ✗

Debugging: locate issues in the source code

Delta-debugging

```
log.L                   .../aster.release
volum2_                 .../aster.release
bilpla_                 .../aster.release
ecrval_                 .../aster.release
print_plath_            .../aster.release
classer_groupes_        .../aster.release
etupla_                 .../aster.release
# couhyd_pi_            .../aster.release
ecrplr_                 .../aster.release
imovi_                  .../aster.release
resopt_                 .../aster.release
getgrp_marginal_        .../aster.release
ecrpla_                 .../aster.release
fin_exec_main_          .../aster.release
# decopt_pi_            .../aster.release
paraend_                .../aster.release
resopt_cnt_zones_       .../aster.release
apstop_                 .../aster.release
# ihyd_                 .../aster.release
impression_info_        .../aster.release
coupla_                 .../aster.release
gere_print_plath_       .../aster.release
log                     .../aster.release
thepla_                 .../aster.release
# coutot_               .../aster.release
# iprit_                .../aster.release
```

- relies on the Verrou ability to restrict the scope of instrumentation/perturbations
- Delta-Debugging [A. Zeller, 1999] adapted for stochastic evaluation

- Inputs :
  - run script
  - comparison script
- Output:
  - “unstable” code parts
- Also works at the source line granularity:
  - if the code was compiled with -g

Conclusions

Verrou as a tool to help with FP issues

Diagnostics

- show the presence of FP-related errors (random-rounding back-end)
- quantify the magnitude of issues (post-processing)

Debugging

- locate the origin of FP-related issues in the source code
  - unstable algorithms (Delta-Debugging)
  - unstable tests (code coverage analysis)
- track the origin of issues during program execution
  - context of calls
  - temporal information (*e.g.* iteration number. . .)

Optimization

- emulate mixed-precision implementations (reduced precision back-end)
- re-use debugging features (Delta-Debugging)

Outlooks

Interflop

- Verrou is no silver bullet
  - multiply techniques & tools

Interflop (toolbox)

Common interface for Verificarlo & Verrou

- share Stochastic Arithmetic back-ends
- share accompanying tools (Delta-Debugging. . .)
- improve performance of instrumentation front-ends

Interflop (larger consortium)

- Explore different analysis methods and the links between them
  - Stochastic Arithmetic
  - Interval Arithmetic & Affine Forms

VERROU

Thank you !

Questions ?

Get Verrou on github:
http://github.com/edf-hpc/verrou

Documentation:
http://edf-hpc.github.io/verrou/vr-manual.html

Relevant references I

Jean-Marie Chesneaux and Jean Vignes, *On the robustness of the cestac method*, C. R. Acad.Sci. Paris **1** (1988), 855–860.

Christophe Denis, Pablo de Oliveira Castro, and Eric Petit, *Verificarlo: checking floating point accuracy through Monte Carlo Arithmetic*, 23rd IEEE Internatinal Symposium on Computer Arithmetic (ARITH'23), 2016.

François Févotte and Bruno Lathuilière, *VERROU: Assessing Floating-Point Accuracy Without Recompiling*, `https://hal.archives-ouvertes.fr/hal-01383417`, October 2016.

François Févotte and Bruno Lathuilière, *Studying the numerical quality of an industrial computing code: A case study on code_aster*, 10th International Workshop on Numerical Software Verification (NSV) (Heidelberg, Germany), July 2017, pp. 61–80.

Relevant references II

- Stef Graillat, Fabienne Jézéquel, and Romain Picot, *Numerical validation of compensated algorithms with stochastic arithmetic*, Applied Mathematics and Computation **329** (2018), 339 – 363.
- Fabienne Jézéquel, Jean-Marie Chesneaux, and Jean-Luc Lamotte, *A new version of the CADNA library for estimating round-off error propagation in Fortran programs*, Computer Physics Communications **181** (2010), no. 11, 1927–1928.
- William Kahan, *How futile are mindless assessments of roundoff in floating-point computations?*, `https://people.eecs.berkeley.edu/~wkahan/Mindless.pdf`, 2006.
- Jean-Luc Lamotte, Jean-Marie Chesneaux, and Fabienne Jézéquel, *CADNA_C: A version of CADNA for use with C or C++ programs*, Computer Physics Communications **181** (2010), no. 11, 1925–1926.

Relevant references III

- Devan Sohier, Pablo De Oliveira Castro, François Févotte, Bruno Lathuilière, Eric Petit, and Olivier Jamond, *Confidence Intervals for Stochastic Arithmetic*, preprint, `https://hal.archives-ouvertes.fr/hal-01827319`.
- Douglas Stott Parker, *Monte Carlo arithmetic: exploiting randomness in floating-point arithmetic*, Tech. Report CSD-970002, University of California, Los Angeles, 1997.
- Jean Vignes, *A stochastic arithmetic for reliable scientific computation*, Mathematics and Computers in Simulation **35** (1993), 233–261.
- Andreas Zeller, *Yesterday, My Program Worked. Today, It Does Not. Why?*, SIGSOFT Softw. Eng. Notes **24** (1999), no. 6, 253–267.

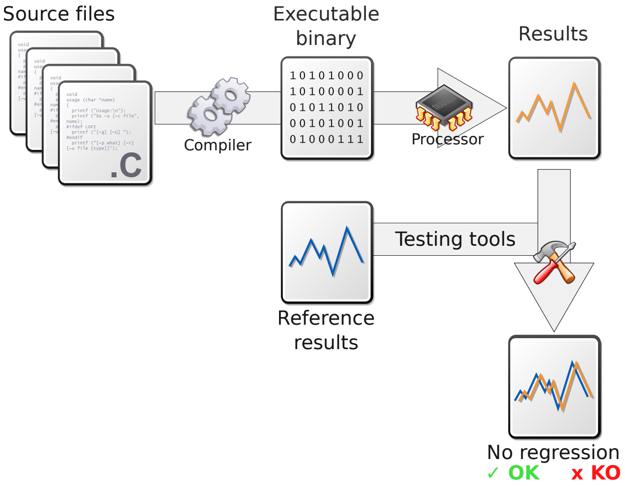
Annexes

1. Verrou
2. Application : Athena

The Verrou tool

Development and QA process

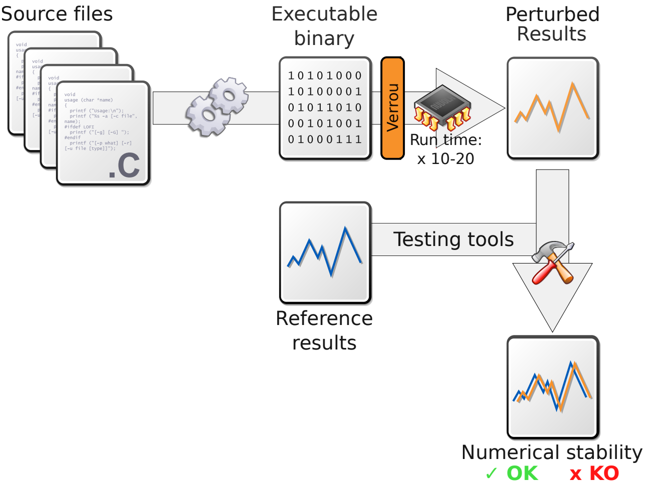
```
$ myProg in out
```



The Verrou tool

The Verrou tool: test the robustness w.r.t changes in the arithmetic

$ valgrind --tool=verrou --rounding-mode=random `myProg in out`



The Verrou tool

Output exemple

```
$ valgrind --tool=verrou --rounding-mode=random PROGRAM [ARGS...]
```

```
==4683== Verrou, Check floating-point rounding errors
==4683== Copyright (C) 2014, F. Fevotte & B. Lathuiliere.
...
==4683== First seed : 1430818339
==4683== Simulating AVERAGE rounding mode
==4683== Instrumented operations :
==4683==  add : yes
...
==4683==  ---------------------------------------------------------------------
==4683==  Operation                    Instructions count
==4683==   `- Precision      Instrumented              Total
==4683==  ---------------------------------------------------------------------
==4683==  add                500869335                 500869335          (100%)
==4683==   `- flt                400695468                 400695468      (100%)
==4683==   `- dbl                100173867                 100173867      (100%)
==4683==  ---------------------------------------------------------------------
==4683==  sub                763127658                 763127658          (100%)
==4683==   `- flt                763127658                 763127658      (100%)
==4683==  ---------------------------------------------------------------------
==4683==  mul               1202086563                1202086563          (100%)
==4683==   `- flt               1101912537                1101912537      (100%)
==4683==   `- dbl                100174026                 100174026      (100%)
==4683==  ---------------------------------------------------------------------
...
```

The Verrou tool

Output exemple

```
$ valgrind --tool=verrou --rounding-mode=random PROGRAM [ARGS...]
```

```
==4683== Verrou, Check floating-point rounding errors
==4683== Copyright (C) 2014, F. Fevotte & B. Lathuiliere.
...
==4683== First seed : 1430818339
==4683== Simulating AVERAGE rounding mode
==4683== Instrumented operations :
==4683==   add : yes
...
==4683==                                                   -----------------
==4683==
==4683==
==4683==                                                   -----------------
==4683==                                                                (100%)
==4683==                                                   5468         (100%)
==4683==    `- dbl              100173867              100173867        (100%)
==4683==   -------------------------------------------------------------------
==4683==   sub               763127658              763127658           (100%)
==4683==    `- flt              763127658              763127658        (100%)
==4683==   -------------------------------------------------------------------
==4683==   mul              1202086563             1202086563           (100%)
==4683==    `- flt             1101912537             1101912537        (100%)
==4683==    `- dbl              100174026              100174026        (100%)
==4683==   -------------------------------------------------------------------
...
```

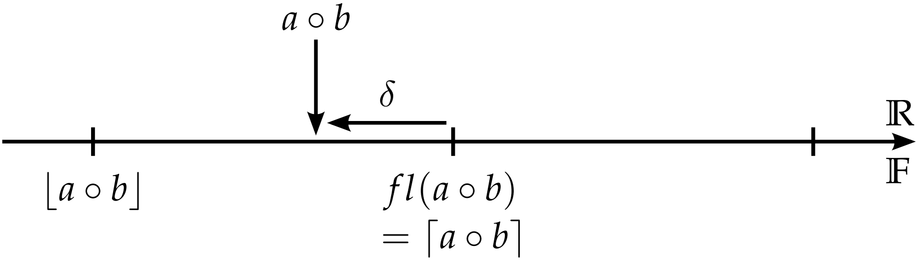
normal output of the program

+ **Warnings for “dangereous” instructions**

(ex : x87)

The Verrou tool

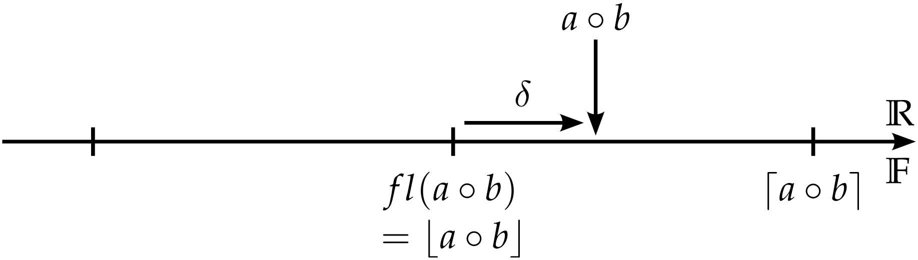
Rounding mode simulation



- Error Free Transformation (the division is more complicated):
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$
- If $\delta < 0$:
  - $\lfloor a \circ b \rfloor = fl(a \circ b) - ulp$,
  - $\lceil a \circ b \rceil = fl(a \circ b)$.

The Verrou tool

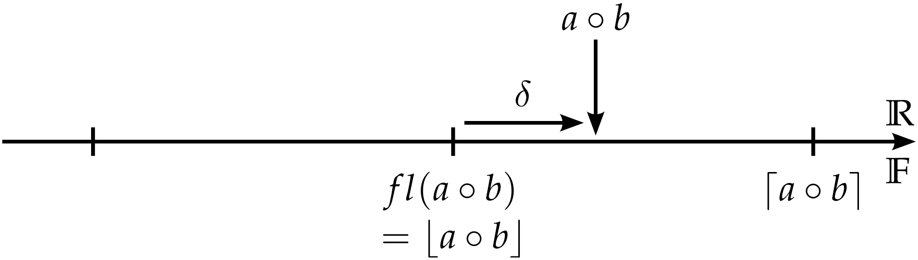
Rounding mode simulation



- Error Free Transformation (the division is more complicated):
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$
- If $\delta > 0$:
  - $\lfloor a \circ b \rfloor = fl(a \circ b)$,
  - $\lceil a \circ b \rceil = fl(a \circ b) + ulp$.

The Verrou tool

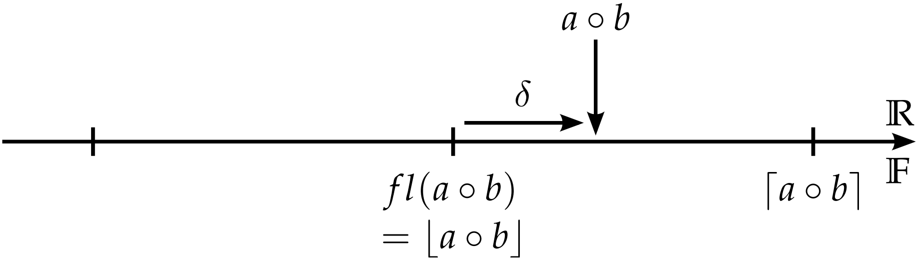
Rounding mode simulation



- `random` (CESTAC)
  - $p(\lceil a \circ b \rceil) = 0.5$
  - $p(\lfloor a \circ b \rfloor) = 0.5$

The Verrou tool

Rounding mode simulation



- `average`
  - $p(\lceil a \circ b \rceil) = \dfrac{\delta}{ulp}$ (if $\delta > 0$)
  - $p(\lfloor a \circ b \rfloor) = \dfrac{ulp - \delta}{ulp}$

Approximate transformation for division

- What we want:

$$\frac{a}{b} = q + r,$$

with $q = \mathrm{fl}(a/b)$.

- proposed algorithm:

> **Input** : a, b
> **Output** : $\tilde{r}$ such as
> $a/b \simeq \mathrm{fl}(a/b) + \tilde{r}$.
> $q \leftarrow \mathrm{fl}(a/b)$
> $(p, s) \leftarrow \mathrm{twoprod}(b, q)$
> $t \leftarrow \mathrm{fl}(a - p)$
> $u \leftarrow \mathrm{fl}(t - s)$
> $\tilde{r} \leftarrow \mathrm{fl}(u/b)$

- Idea of proof :

$$q = \frac{a}{b}\,(1 + \epsilon_1)$$

$$\begin{aligned} p &= b\,q\,(1 + \epsilon_2) \\ &= a\,(1 + \epsilon_1)\,(1 + \epsilon_2) \end{aligned}$$

$$t = a - p \qquad \text{(Sterbenz lemma)}$$

$$\begin{aligned} u &= (t - s)\,(1 + \epsilon_3) \\ &= \Big(a - (p + s)\Big)\,(1 + \epsilon_3) \\ &= (a - bq)\,(1 + \epsilon_3) \\ &= b\,r\,(1 + \epsilon_3) \end{aligned}$$

$$\begin{aligned} \tilde{r} &= \frac{u}{b}\,(1 + \epsilon_4) \\ &= r\,(1 + \epsilon_3)\,(1 + \epsilon_4)\,. \end{aligned}$$

The Verrou tool

Using Verrou and Random Rounding

Test case	nearest
ssls108i	OK
ssls108j	OK
ssls108k	OK
ssls108l	OK
sdnl112a	OK
ssnp130a	OK
ssnp130b	OK
ssnp130c	OK
ssnp130d	OK

The Verrou tool

Using Verrou and Random Rounding

△

Test case	nearest	rnd_1
`ssls108i`	OK	OK
`ssls108j`	OK	OK
`ssls108k`	OK	OK
`ssls108l`	OK	OK
`sdnl112a`	OK	KO
`ssnp130a`	OK	OK
`ssnp130b`	OK	OK
`ssnp130c`	OK	OK
`ssnp130d`	OK	OK

The Verrou tool

Using Verrou and Random Rounding

Test case	Status nearest	rnd_1	rnd_2	rnd_3
`ssls108i`	OK	OK	OK	OK
`ssls108j`	OK	OK	OK	OK
`ssls108k`	OK	OK	OK	OK
`ssls108l`	OK	OK	OK	OK
`sdnl112a`	OK	KO	KO	KO
`ssnp130a`	OK	OK	OK	OK
`ssnp130b`	OK	OK	OK	OK
`ssnp130c`	OK	OK	OK	OK
`ssnp130d`	OK	OK	OK	OK

10 minutes 20 minutes each

(72 test cases)

The Verrou tool

Using Verrou and Random Rounding

Test case	Status nearest	rnd_1	rnd_2	rnd_3	# common decimal digits $C(\mathsf{rnd}_1, \mathsf{rnd}_2, \mathsf{rnd}_3)$
`ssls108i`	OK	OK	OK	OK	10
`ssls108j`	OK	OK	OK	OK	10
`ssls108k`	OK	OK	OK	OK	10
`ssls108l`	OK	OK	OK	OK	9
`sdnl112a`	OK	KO	KO	KO	3
`ssnp130a`	OK	OK	OK	OK	9
`ssnp130b`	OK	OK	OK	OK	9
`ssnp130c`	OK	OK	OK	OK	9
`ssnp130d`	OK	OK	OK	OK	9

10 minutes

(72 test cases)

20 minutes each

$$C(x) = \log_{10} \left| \frac{\mu(x)}{\sigma(x)} \right|$$

The Verrou tool

Delta-Debugging variants and improvements

Definition of “unstable” program parts

- DDmax: straws that breaks the camels back
- DDmin: parts that cause instabilities on their own

Preliminary filtering of the list

- only consider functions performing FP computations
  - code_aster test case (`ttlv300a`): 4459 $\rightarrow$ 154 symbols (96.5% reduction)
- a preliminary basic search might be beneficial for such small lists

Account for the stochastic nature of tests

- progressively increase the number of correct runs required to validate a set

The Verrou tool

Comparison of Delta-Debugging algorithms

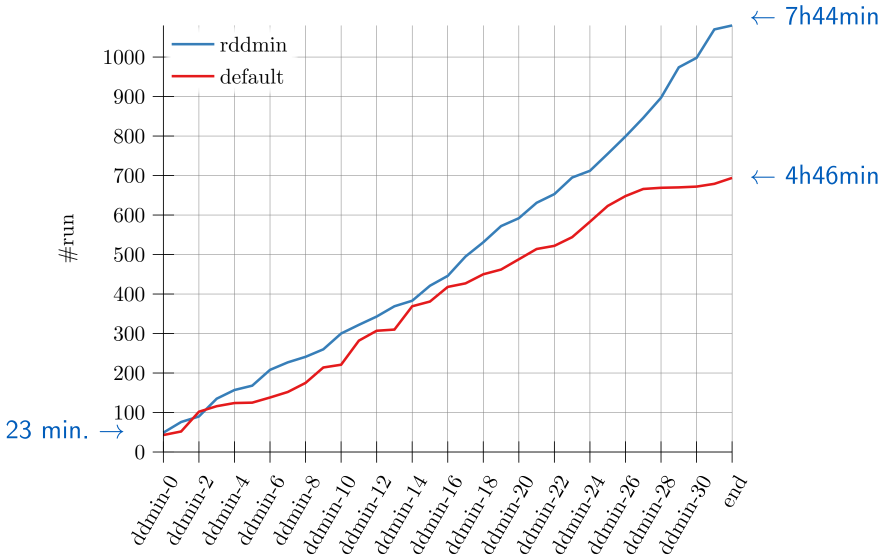
- Code_aster test case (ttlv300a)
  - 10 correct runs to validate a set
  - pre-filtering of FP functions

Algo	Total time	Speed-up
DDmax	12h 57min	
rDDmin	7h 44min	1.7
new default	4h 46min	2.7

- Tested algorithms
  - standard DDmax
  - standard DDmin
  - Verrou's improved DDmin (new default)
    - preliminary binary search with 5 correct runs requirement
    - exponential progression of the number of runs (1,2,5,10)

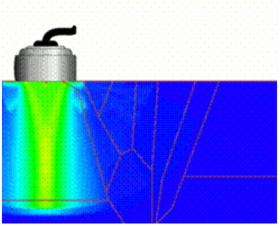
The Verrou tool

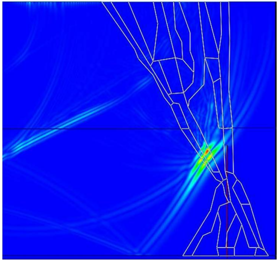
Standard DDmin algorithm vs Verrou's new default



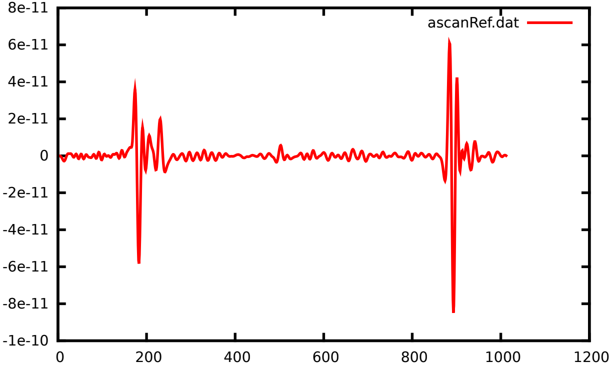
Application : NDT by US

Code Athena 2D



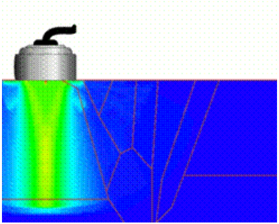


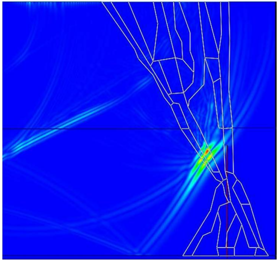
Produced result : "A-scan"



Application : NDT by US

Code Athena 2D





Non destructive testing by Ultra-Sonic technics

Code Athena 2D

- 36k code lines
- Fortran 77 + Fortran 90
- Dependencies : BLAS, LAPACK
- Code “known”

Goal

- No identified problem
- “routine” test

Application : NDT by US △

Non-regression test with `verrou`

	random 1
Cas-Test A	
ins1.dat	0
ascan.dat	1.8e-12
Cas-Test B	
sismo.dat	7.9e-69
ascan.dat	1.2e-10
Cas-Test C	
ins1.dat	4.6e-06
sismo.dat	8.0e-28
ascan.dat	2.0e-11
Cas-Test D	
ins1.dat	1.5e-18
enerloc.dat	0
sismo.dat	0
ascan.dat	0

Application : NDT by US

Non-regression test with `verrou`

	random 1	random 2
Cas-Test A		
ins1.dat	0	6.1e-06
ascan.dat	1.8e-12	5.9e-12
Cas-Test B		
sismo.dat	7.9e-69	7.9e-69
ascan.dat	1.2e-10	2.0e-11
Cas-Test C		
ins1.dat	4.6e-06	4.6e-06
sismo.dat	8.0e-28	2.8e-28
ascan.dat	2.0e-11	1.2e-11
Cas-Test D		
ins1.dat	1.5e-18	**4.1e-01**
enerloc.dat	0	**2.3e-01**
sismo.dat	0	**1.6e-01**
ascan.dat	0	**1.5e-01**

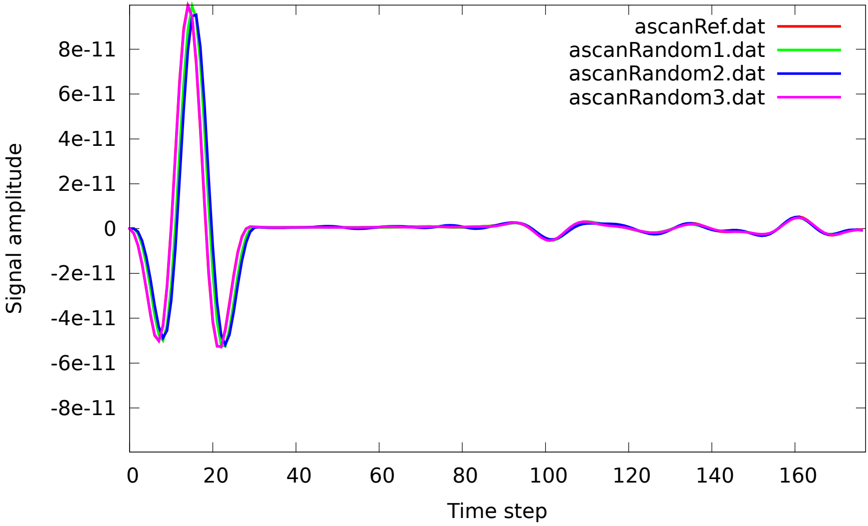
Application : NDT by US

Non-regression test with `verrou`

	random 1	random 2	random 3	random 4
Cas-Test A				
ins1.dat	0	6.1e-06	6.1e-06	6.1e-06
ascan.dat	1.8e-12	5.9e-12	5.9e-12	5.9e-12
Cas-Test B				
sismo.dat	7.9e-69	7.9e-69	4.3e-69	4.3e-69
ascan.dat	1.2e-10	2.0e-11	2.8e-10	1.1e-11
Cas-Test C				
ins1.dat	4.6e-06	4.6e-06	4.6e-06	0
sismo.dat	8.0e-28	2.8e-28	8.0e-28	0
ascan.dat	2.0e-11	1.2e-11	1.8e-11	0
Cas-Test D				
ins1.dat	1.5e-18	**4.1e-01**	**2.0e-01**	0
enerloc.dat	0	**2.3e-01**	**1.2e-01**	0
sismo.dat	0	**1.6e-01**	**3.2e-02**	0
ascan.dat	0	**1.5e-01**	**3.6e-01**	**6.5e-03**

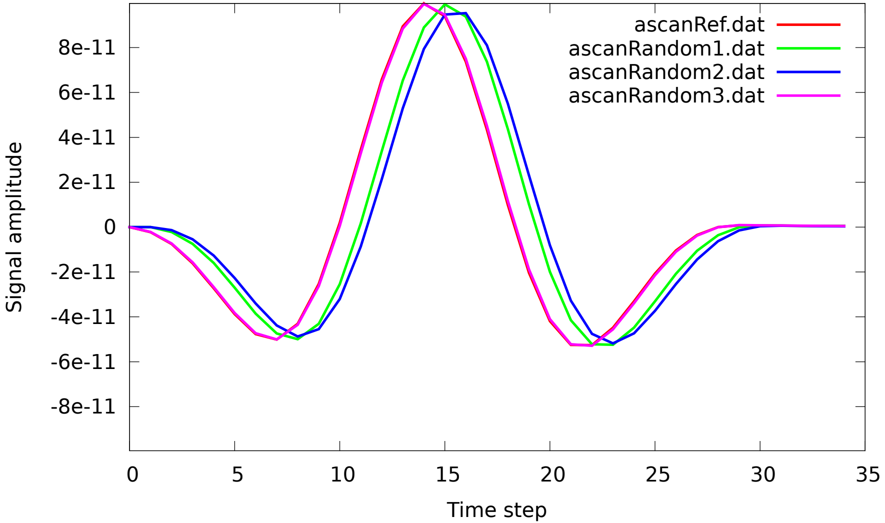
Application : NDT by US

Test case D



Application : NDT by US

Test case D



Application : NDT by US

Verrou performances

	reference	random	average
Cas-Test A	4.70s	83.23s (x17)	90.49s (x19)
Cas-Test B	29.79s	969.54s (x32)	1042.02s (x34)
Cas-Test C	21.15s	326.81s (x15)	358.08s (x16)
Cas-Test D	1.99s	24.20s (x12)	25.87s (x12)
Cas-Test E	0.46s	7.88s (x17)	8.88s (x19)
Cas-Test F	0.38s	4.54s (x11)	4.95s (x12)
Cas-Test G	6.16s	100.31s (x16)	109.70s (x17)
Cas-Test H	14.09s	503.90s (x35)	549.50s (x39)
Cas-Test I	1.48s	14.34s (x9)	14.85s (x10)

Slow down between $\times 9$ and $\times 39$

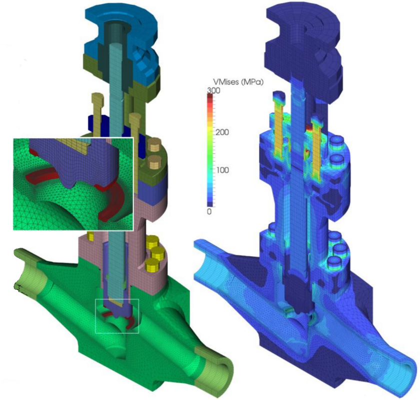
Additional slides

High Performance Computing

- **Performance** is the primary concern
  - if more **powerful hardware** can be used, it will be used, ...
  - even if **new developments** are needed to harness such hardware
  - and **no holds are barred**:
    - as low as possible precision
    - unsafe compiler optimizations (such as `-ffast-math`)
    - loop vectorization / unrolling / both
    - hand-tuned assembly code
    - ...
- Every high-performance developer knows these techniques are unsafe
  - and probably learned the hard way,
  - but it is part of the trade.

Additional slides

V&V process: ad-hoc numerical instability detection methods



Physical input: affects the result
Simulation parameter: should be neutral

INPUTS

geometry →
materials →
mesh →

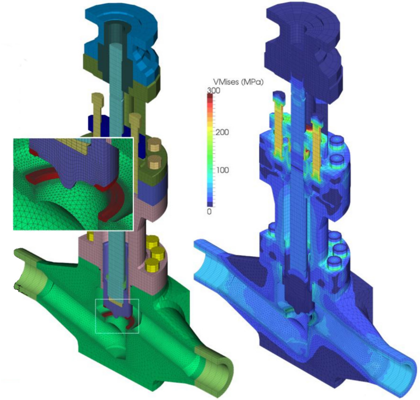
Code → results

OUTPUTS

- Idea: measure the sensitivity of the results w.r.t "neutral" parameters
  - easy to do
  - *ad hoc*, no localization

Additional slides

V&V process: ad-hoc numerical instability detection methods



Physical input: affects the result
Simulation parameter: should be neutral

INPUTS

- geometry
- materials
- mesh
- compiler
- hardware specification
- ...

Code

OUTPUTS

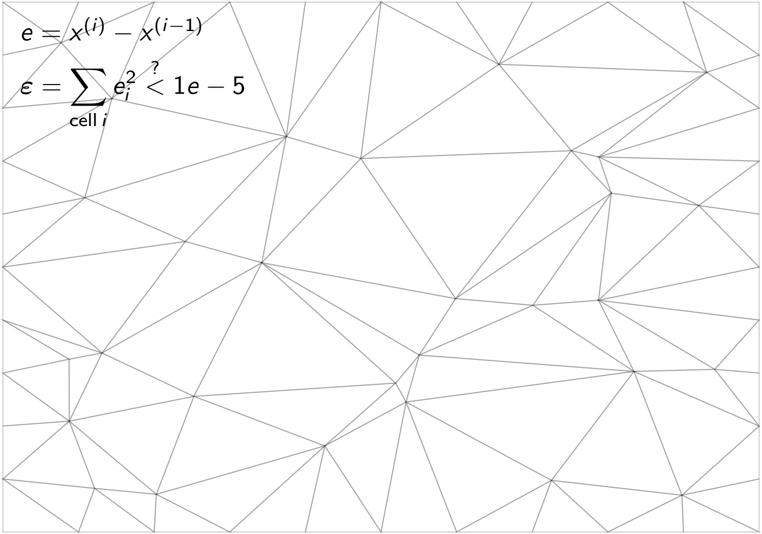
- results

Additional slides

Example 1: failing computation in free surface hydraulics (Telemac 2D)

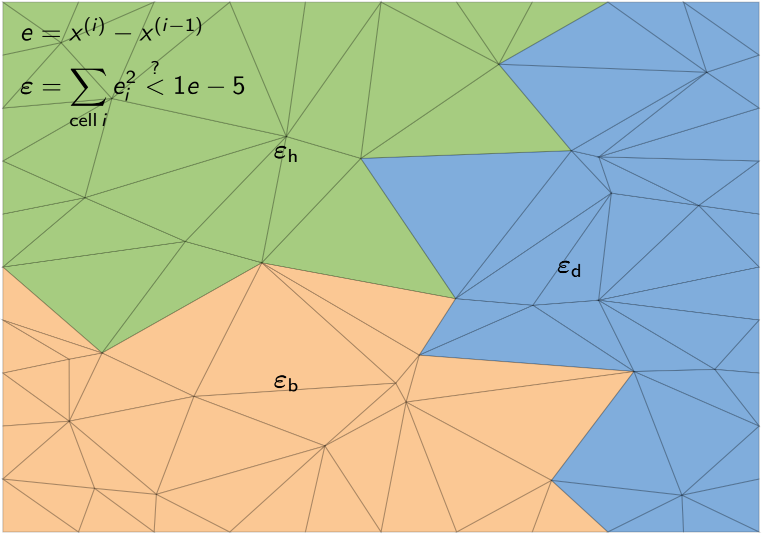
$$e = x^{(i)} - x^{(i-1)}$$

$$\varepsilon = \sum_{\text{cell } i} e_i^2 \overset{?}{<} 1e-5$$



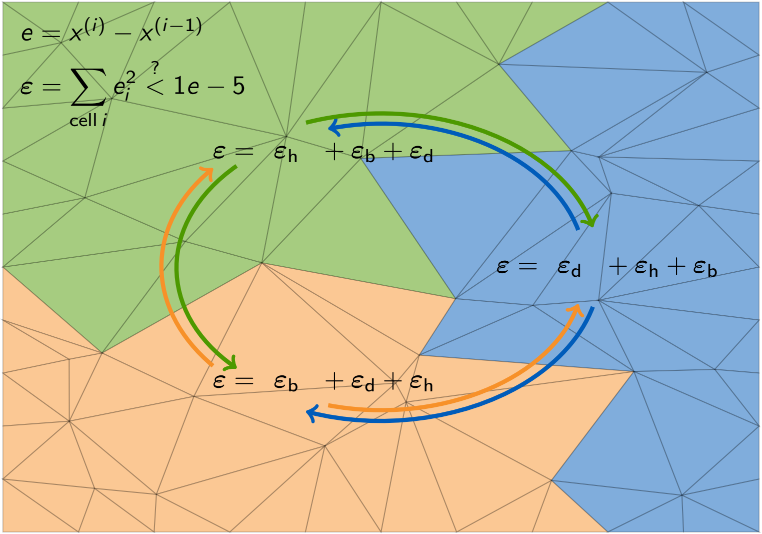
Additional slides

Example 1: failing computation in free surface hydraulics (Telemac 2D)



Additional slides

Example 1: failing computation in free surface hydraulics (Telemac 2D)



Additional slides

Example 2: performances of an optimization problem

Initial problem formulation

$$\max_{p,v} \sum_{t\in T} \sum_{i\in I} \lambda_t p_t^i + \sum_{r\in R} \omega_r \left(v_t^r - V_0^r - \sum_{t\in T} \Gamma_t^r \right)$$

Objective functional computation

$$\begin{array}{lr} \displaystyle\sum_{t\in T} \sum_{i\in I} \lambda_t p_t^i + \sum_{r\in R} \omega_r v_t^r & 1534019.677371745 \\ \displaystyle -\sum_{r\in R} \omega_r \left(V_0^r + \sum_{t\in T} \Gamma_t^r \right) & -1534019.677282780 \\ \hline & 0.000088965\underbrace{00000000000}_{\text{11 digits}} \end{array}$$

Proposed re-formulation

$$\left[\max_{p,v} \sum_{t\in T}\sum_{i\in I}\lambda_t p_t^i + \sum_{r\in R}\omega_r v_t^r\right] - \sum_{r\in R}\omega_r\left(V_0^r + \sum_{t\in T}\Gamma_t^r\right)$$

Objective functional computation

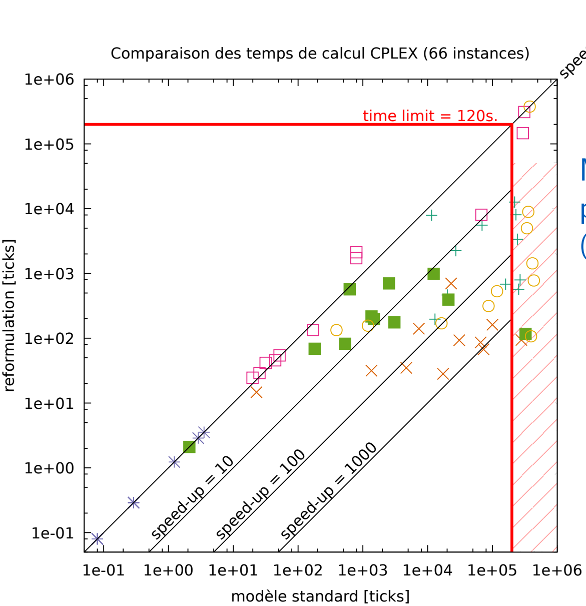
$$\sum_{t\in T}\sum_{i\in I}\lambda_t p_t^i + \sum_{r\in R}\omega_r v_t^r \qquad 1534019.677371745$$

$$-\sum_{r\in R}\omega_r\left(V_0^r + \sum_{t\in T}\Gamma_t^r\right) \qquad -1534019.677282780$$

$$0.000088965\underbrace{00000000000}_{\text{11 digits}}$$

Additional slides

Example 2: performances of an optimization problem



MILP resolution performances (66 instances)

- 2 (3%) slowed down (×2)
- 15 (23%) unchanged
- 49 (74%) speed up (×2–1100)

Additional slides

Case study on ACTS: emulation of reduced precision + NaN detection

```
$ valgrind --tool=verrou --rounding-mode=float --exclude=excludes.ex --demangle=no --trace-children=yes --num-callers=50 \
>          ctest -V -R "BFieldMapUtils"
                                                    [...]
==1863== NaN:
==1863==    at 0x5847E7F: _ZN5Eigen8internal4pmulIDv2_dEET_RKS3_S5_ (emmintrin.h:271)
==1863==    by 0x585B570: _ZNK5Eigen8internal17scalar_product_opIddE8packetOpIDv2_dEEKT_RS6_S7_ (BinaryFunctors.h:89)
==1863==    by 0x59A31C9: _ZNK5Eigen8internal16binary_evaluatorINS_13CwiseBinaryOpINS0_17scalar_product_opIddEEKNS_14
                          CwiseNullaryOpINS0_18scalar_constant_opIdEEKNS_6MatrixIdLi2ELi1ELi0ELi2ELi1EEEEEESA_EENS0_10
                          IndexBasedESE_ddE6packetILi16EDv2_dEET0_ll (CoreEvaluators.h:727)
                                                    [...]
==1863==    by 0x5BDE2D4: _ZN5Eigen6MatrixIdLi2ELi1ELi0ELi2ELi1EEaSINS_13CwiseBinaryOpINS_8internal13scalar_sum_opIdd
                          EEKNS3_INS4_17scalar_product_opIddEEKNS_14CwiseNullaryOpINS4_18scalar_constant_opIdEEKS1_EE
                          SC_EESG_EEEERS1_RKNS_9DenseBaseIT_EE (Matrix.h:225)
==1863==    by 0x5BDAD82: _ZN4Acts6detail16interpolate_implIN5Eigen6MatrixIdLi2ELi1ELi0ELi2ELi1EEES4_St5arrayIdLm2EES
                          6_Lm1ELm4EE3runERKS4_RKS6_SB_RKS5_IS4_Lm4EE (interpolation_impl.hpp:133)
==1863==    by 0x5BD59E8: _ZN4Acts11interpolateIN5Eigen6MatrixIdLi2ELi1ELi0ELi2ELi1EEELm4ES3_St5arrayIdLm2EES5_vEET_R
                          KT1_RKT2_RKT3_RKS4_IS6_XT0_EE (Interpolation.hpp:95)
                                                    [...]
==1863==    by 0x4F6D54:  _ZNK4Acts21InterpolatedBFieldMap11FieldMapperILj2ELj2EE8getFieldERKN5Eigen6MatrixIdLi3ELi1EL
                          i0ELi3ELi1EEE (InterpolatedBFieldMap.hpp:166)
==1863==    by 0x477030:  _ZN4Acts4Test15bfield_creation11test_methodEv (BFieldMapUtilsTests.cpp:88)
==1863==    by 0x474767:  _ZN4Acts4TestL23bfield_creation_invokerEv (BFieldMapUtilsTests.cpp:25)
==1863==    by 0x549B9B:  _ZN5boost6detail8function22void_function_invoker0IPFvvEvE6invokeERNS1_15function_bufferE (fu
                          nction_template.hpp:118)
                                                    [...]
==1863==    by 0x43F2FE:  _ZN5boost9unit_test9framework3runEmb (framework.ipp:1629)
==1863==    by 0x463F10:  _ZN5boost9unit_test14unit_test_mainEPFPNS0_10test_suiteEiPPcEiS4_ (unit_test_main.ipp:247)
==1863==    by 0x4648CB:  main (unit_test_main.ipp:303)
                                                    [...]
Running 12 test cases...
acts-core/Tests/Utilities/BFieldMapUtilsTests.cpp(105): error: in "bfield_creation": difference{-nan} between
        value2_rz.perp(){-nan} and bField2_rz.perp(){8} exceeds 1e-09%
```