Precision Auto-Tuning and Control of Accuracy in Numerical Applications

Fabienne Jézéquel

Sorbonne Université, Laboratoire d'Informatique de Paris 6 (LIP6), France

Workshop on Large-scale Parallel Numerical Computing Technology
RIKEN Center for Computational Science, Kobe, Japan, 29-30 January 2020

Overview

- How to estimate rounding errors?

  ⇒ Discrete Stochastic Arithmetic and its implementations (CADNA, SAM)

Overview

- How to estimate rounding errors?

  ⇒ Discrete Stochastic Arithmetic and its implementations (CADNA, SAM)

- How to use less precision taking into account accuracy requirements?

  ⇒ Precision auto-tuning: the PROMISE software

Overview

- How to estimate rounding errors?

  ⇒ Discrete Stochastic Arithmetic and its implementations (CADNA, SAM)

- How to use less precision taking into account accuracy requirements?

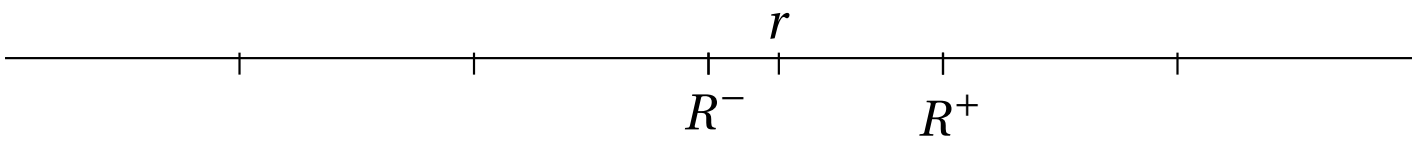
  ⇒ Precision auto-tuning: the PROMISE software

- Can we avoid controlling accuracy and still get trustful results?

  ⇒ Yes, if BLAS 3 routines are used with perturbed data.

How to estimate rounding error propagation?

The exact result r of an arithmetic operation is approximated by a floating-point number R^- or R^+.



The random rounding mode

Approximation of r by R^- or R^+ with the probability 1/2

The CESTAC method [La Porte & Vignes 1974]

The same code is run several times with the random rounding mode.
Then different results are obtained.

Briefly, the part that is common to all the different results is assumed to be reliable and the part that is different in the results is affected by round-off errors.

Implementation of the CESTAC method

The implementation of the CESTAC method in a code providing a result R consists in:

- performing N times this code with the random rounding mode to obtain N samples R_i of R,
- choosing as the computed result the mean value $\overline{R}$ of R_i, $i = 1, ..., N$,
- estimating the number of exact significant decimal digits of $\overline{R}$ with

$$C_{\overline{R}} = \log_{10}\left(\frac{\sqrt{N}\left|\overline{R}\right|}{\sigma\tau_\beta}\right)$$

where

$$\overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i \quad \text{and} \quad \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(R_i - \overline{R}\right)^2.$$

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1-\beta$.

In pratice, $N = 3$ and $\beta = 5\%$.

The problem of stopping criteria

Let us consider a general iterative algorithm: $U_{n+1} = F(U_n)$.

```
while (fabs(X-Y) > EPSILON) {
    X = Y;
    Y = F(X);
}
```

ε too low $\Longrightarrow$ risk of infinite loop
ε too high $\Longrightarrow$ too early termination.

The problem of stopping criteria

Let us consider a general iterative algorithm: $U_{n+1} = F(U_n)$.

```
while (fabs(X-Y) > EPSILON) {
    X = Y;
    Y = F(X);
}
```

ε too low $\Longrightarrow$ risk of infinite loop
ε too high $\Longrightarrow$ too early termination.

It would be optimal to stop when $X - Y$ is an **insignificant value**.

Such a stopping criterion

- would enable one to develop new numerical algorithms
- is possible thanks to the concept of *computational zero*.

The concept of computational zero

Definition [Vignes, 1986]

A result R computed using the CESTAC method is a computational zero, denoted by @.0, if

$$\forall i, R_i = 0 \quad \text{or} \quad C_{\overline{R}} \leq 0.$$

It means that R is a computed result which, because of round-off errors, cannot be distinguished from 0.

The stochastic definitions

Let X and Y be two results computed using the CESTAC method (N-samples).

- X is stochastically equal to Y, noted $X \stackrel{s}{=} Y$, iff

$$X - Y = @.0.$$

- X is stochastically strictly greater than Y, noted $X \stackrel{s}{>} Y$, iff

$$\overline{X} > \overline{Y} \quad \text{and} \quad X \stackrel{s}{\neq} Y$$

- X is stochastically greater than or equal to Y, noted $X \stackrel{s}{\geq} Y$, iff

$$\overline{X} \geq \overline{Y} \quad \text{or} \quad X \stackrel{s}{=} Y$$

The stochastic definitions

Let X and Y be two results computed using the CESTAC method (N-samples).

- X is stochastically equal to Y, noted $X \stackrel{s}{=} Y$, iff

$$X - Y = @.0.$$

- X is stochastically strictly greater than Y, noted $X \stackrel{s}{>} Y$, iff

$$\overline{X} > \overline{Y} \quad \text{and} \quad X \stackrel{s}{\neq} Y$$

- X is stochastically greater than or equal to Y, noted $X \stackrel{s}{\geq} Y$, iff

$$\overline{X} \geq \overline{Y} \quad \text{or} \quad X \stackrel{s}{=} Y$$

These stochastic relations require that each arithmetic operation is performed N times before the next one is executed.

Discrete Stochastic Arithmetic [Vignes, 2004]

Discrete Stochastic Arithmetic (DSA) is defined as the joint use of

- the CESTAC method
- the computational zero
- the stochastic relation definitions.

Implementation of DSA

- CADNA: for programs in half, single, double, and/or quadruple precision
  `http://cadna.lip6.fr`
  support for wide range of codes (vectorised, MPI, OpenMP, GPU)
- SAM: for arbitrary precision programs (based on MPFR)
  `http://www-pequan.lip6.fr/~jezequel/SAM`

The CADNA library http://cadna.lip6.fr

CADNA allows one to estimate round-off error propagation in any scientific program written in C, C++ or Fortran.

CADNA enables one to:

- estimate the numerical quality of any result
- detect numerical instabilities
- take into account uncertainty on data.

The CADNA library http://cadna.lip6.fr

CADNA allows one to estimate round-off error propagation in any scientific program written in C, C++ or Fortran.

CADNA enables one to:

- estimate the numerical quality of any result
- detect numerical instabilities
- take into account uncertainty on data.

CADNA provides new numerical types, the stochastic types, which consist of:

- 3 floating point variables
- an integer variable to store the accuracy.

All operators and mathematical functions are redefined for these types.
⇒ CADNA requires only a few modifications in user programs.

Cost of CADNA

The cost of CADNA is about 4 in memory, 10 in run time.

CADNA overhead:

Memory Bound Add	Compute Bound Add	Memory Bound Multiply	Compute Bound Multiply
$7.89\times$	$8.92\times$	$11.6\times$	$9.19\times$

(Intel Xeon E3-1275 at 3.5 GHz, gcc version 4.9.2, single precision, self-validation)

An example without/with CADNA

Computation of $P(x, y) = 9x^4 - y^4 + 2y^2$ [S.M. Rump, 1983]

```
#include <stdio.h>

double rump(double x, double y) {
  return 9.0*x*x*x*x - y*y*y*y + 2.0*y*y;
}
int main(int argc, char **argv) {
  double x, y;
  x = 10864.0;
  y = 18817.0;
  printf("P1=%.14e\n", rump(x, y));
  x = 1.0/3.0;
  y = 2.0/3.0;
  printf("P2=%.14e\n", rump(x, y));
  return 0;
}
```

An example without/with CADNA

Computation of $P(x, y) = 9x^4 - y^4 + 2y^2$ [S.M. Rump, 1983]

```
#include <stdio.h>

double rump(double x, double y) {
  return 9.0*x*x*x*x - y*y*y*y + 2.0*y*y;
}
int main(int argc, char **argv) {
  double x, y;
  x = 10864.0;
  y = 18817.0;
  printf("P1=%.14e\n", rump(x, y));
  x = 1.0/3.0;
  y = 2.0/3.0;
  printf("P2=%.14e\n", rump(x, y));
  return 0;
}
```

P1=2.00000000000000e+00

P2=8.02469135802469e–01

```
#include <stdio.h>

double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {

  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"

  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {

  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"

  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"

  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double   rump(double   x, double   y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double   x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double_st rump(double_st x, double_st y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double_st x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double_st rump(double_st x, double_st y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double_st x, y;
  x=10864.0; y=18817.0;
  printf("P1=%.14e\n",      rump(x, y) );"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%.14e\n",      rump(x, y) );"
  cadna_end();
  return 0;
}
```

```
#include <stdio.h>
#include <cadna.h>
double_st rump(double_st x, double_st y) {
  return 9.0*x*x*x*x-y*y*y*y+2.0*y*y;
}
int main(int argc, char **argv) {
  cadna_init(-1);
  double_st x, y;
  x=10864.0; y=18817.0;
  printf("P1=%s\n", strp(rump(x, y)));"
  x=1.0/3.0; y=2.0/3.0;
  printf("P2=%s\n", strp(rump(x, y)));"
  cadna_end();
  return 0;
}
```

Results with CADNA

only correct digits are displayed

Self-validation detection: ON
Mathematical instabilities detection: ON
Branching instabilities detection: ON
Intrinsic instabilities detection: ON
Cancellation instabilities detection: ON

P1= @.0 (no more correct digits)
P2= 0.80246913580246 9E+000

There are 2 numerical instabilities
2 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S)

Tools related to CADNA

http://cadna.lip6.fr

- CADNAIZER
  automatically transforms C codes to be used with CADNA

- CADTRACE
  identifies the instructions responsible for numerical instabilities

  Example:

  There are 11 numerical instabilities.

  10 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S).
  5 in <ex> file "ex.f90" line 58
  5 in <ex> file "ex.f90" line 59

  1 INSTABILITY IN ABS FUNCTION.
  1 in <ex> file "ex.f90" line 37

The SAM library:

Stochastic Arithmetic in Multiprecision

The SAM library

http://www-pequan.lip6.fr/~jezequel/SAM

- The SAM library implements in arbitrary precision the features of DSA:
  - the stochastic types
  - the concept of computational zero
  - the stochastic operators.
- The SAM library is based on MPFR.
- Classical variables → stochastic variables (of `mp_st` type) consisting of
  - three variables of MPFR type
  - an integer to store the accuracy.
- All operators are overloaded
  ⇨ for a program in C++ to be used with SAM, only a few modifications are needed.

Example of SAM code

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y}$$

is computed with $x = 77617,\ y = 33096$.
[S. Rump, 1988]

```
#include "sam.h"
#include <stdio.h>
int main() {
   sam_init(-1,122);
   mp_st x = 77617; mp_st y = 33096; mp_st res;
   res=333.75*y*y*y*y*y*y+x*x*(11*x*x*y*y-y*y*y*y*y*y
      -121*y*y*y*y-2.0)+5.5*y*y*y*y*y*y*y*y+x/(2*y);
   printf("res=%s\n",strp(res));
   sam_end();
}
```

Output of the SAM code

Using SAM with 122-bit mantissa length, one obtains:

```
Self-validation detection:  ON
Mathematical instabilities detection:  ON
Branching instabilities detection:  ON
Intrinsic instabilities detection:  ON
Cancellation instabilities detection:  ON
--------------
res=-0.82739605994682136814116509547981629 2
--------------
No instability detected
```

Output of the SAM code

Using SAM with 122-bit mantissa length, one obtains:

```
Self-validation detection:  ON
Mathematical instabilities detection:  ON
Branching instabilities detection:  ON
Intrinsic instabilities detection:  ON
Cancellation instabilities detection:  ON
--------------
res=-0.82739605994682136814116509547981629 2
--------------
No instability detected
```

[S. Graillat, F. Jézéquel, S. Wang, Y. Zhu, Stochastic Arithmetic in Multiprecision, 2011]

work in progress: a version of SAM mixing different precisions

Precision auto-tuning

The PROMISE tool

Precision optimization

- mixed precision often leads to better performance
- some existing tools:
  - CRAFT HPC [Lam & al., 2013]
    - binary modifications on the operations
  - Precimonious [Rubio-Gonzàlez & al., 2013]
    - source modification with LLVM

  They rely on comparisons with the highest precision result.

Precision optimization

- mixed precision often leads to better performance
- some existing tools:
  - CRAFT HPC [Lam & al., 2013]
    - binary modifications on the operations
  - Precimonious [Rubio-Gonzàlez & al., 2013]
    - source modification with LLVM

  They rely on comparisons with the highest precision result.

⚠ [Rump, 1988] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

Precision optimization

- mixed precision often leads to better performance
- some existing tools:
  - CRAFT HPC [Lam & al., 2013]
    - binary modifications on the operations
  - Precimonious [Rubio-Gonzàlez & al., 2013]
    - source modification with LLVM

  They rely on comparisons with the highest precision result.

⚠ [Rump, 1988] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$
float: P =2.571784e+29

Precision optimization

- mixed precision often leads to better performance
- some existing tools:
  - CRAFT HPC [Lam & al., 2013]
    - binary modifications on the operations
  - Precimonious [Rubio-Gonzàlez & al., 2013]
    - source modification with LLVM

  They rely on comparisons with the highest precision result.

[Rump, 1988] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

float: P =2.571784e+29

double: P =1.17260394005318

Precision optimization

- mixed precision often leads to better performance
- some existing tools:
  - CRAFT HPC [Lam & al., 2013]
    - binary modifications on the operations
  - Precimonious [Rubio-Gonzàlez & al., 2013]
    - source modification with LLVM

  They rely on comparisons with the highest precision result.

⚠ [Rump, 1988] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

float: P =2.571784e+29
double: P =1.17260394005318
quad: P =1.172603940053178631858834904520 18

Precision optimization

- mixed precision often leads to better performance
- some existing tools:
  - CRAFT HPC [Lam & al., 2013]
    - binary modifications on the operations
  - Precimonious [Rubio-Gonzàlez & al., 2013]
    - source modification with LLVM

  They rely on comparisons with the highest precision result.

[Rump, 1988] $P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
with $x = 77617$ and $y = 33096$

float:	P =2.571784e+29
double:	P =1.17260394005318
quad:	P =1.172603940053178631858834904520l8
exact:	$P \approx$-0.827396059946821368141165095479816292

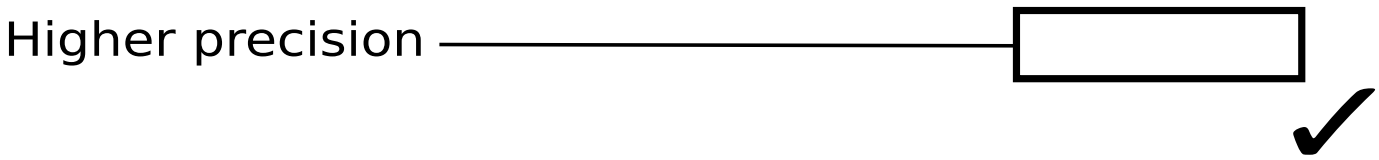
PROMISE (PRecision OptiMISE)

promise.lip6.fr

- Taking into account a required accuracy, PROMISE provides a mixed precision configuration (float, double, quad)
- 2 ways to validate a configuration:
  - validation of every execution using CADNA
  - validation of a reference using CADNA and comparison to this reference

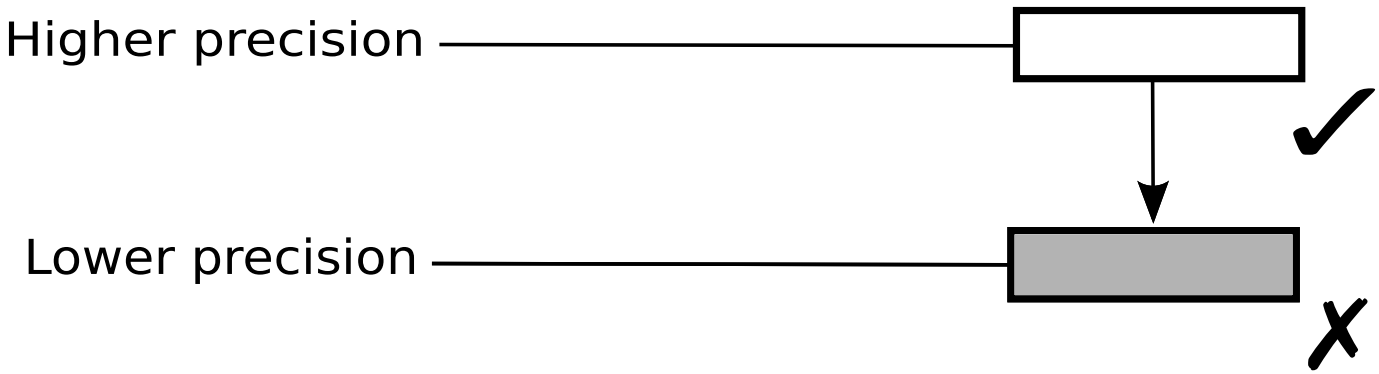
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller, 2009]



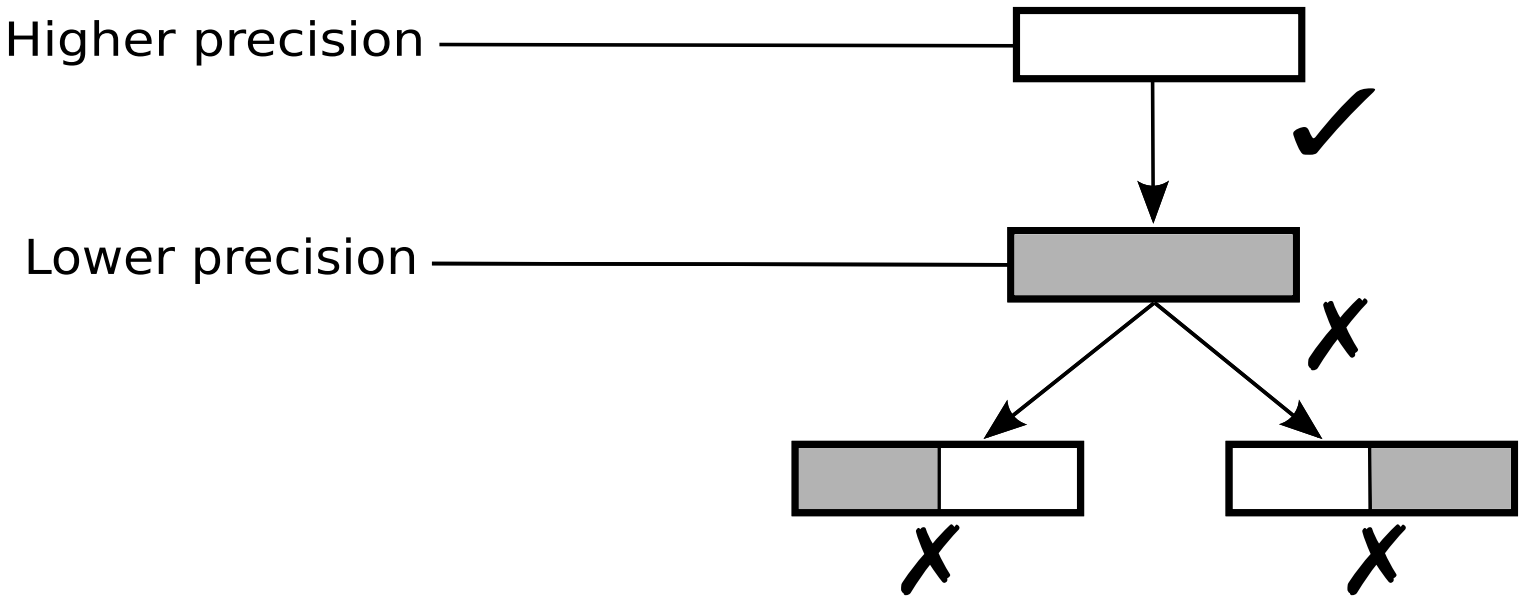
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller, 2009]



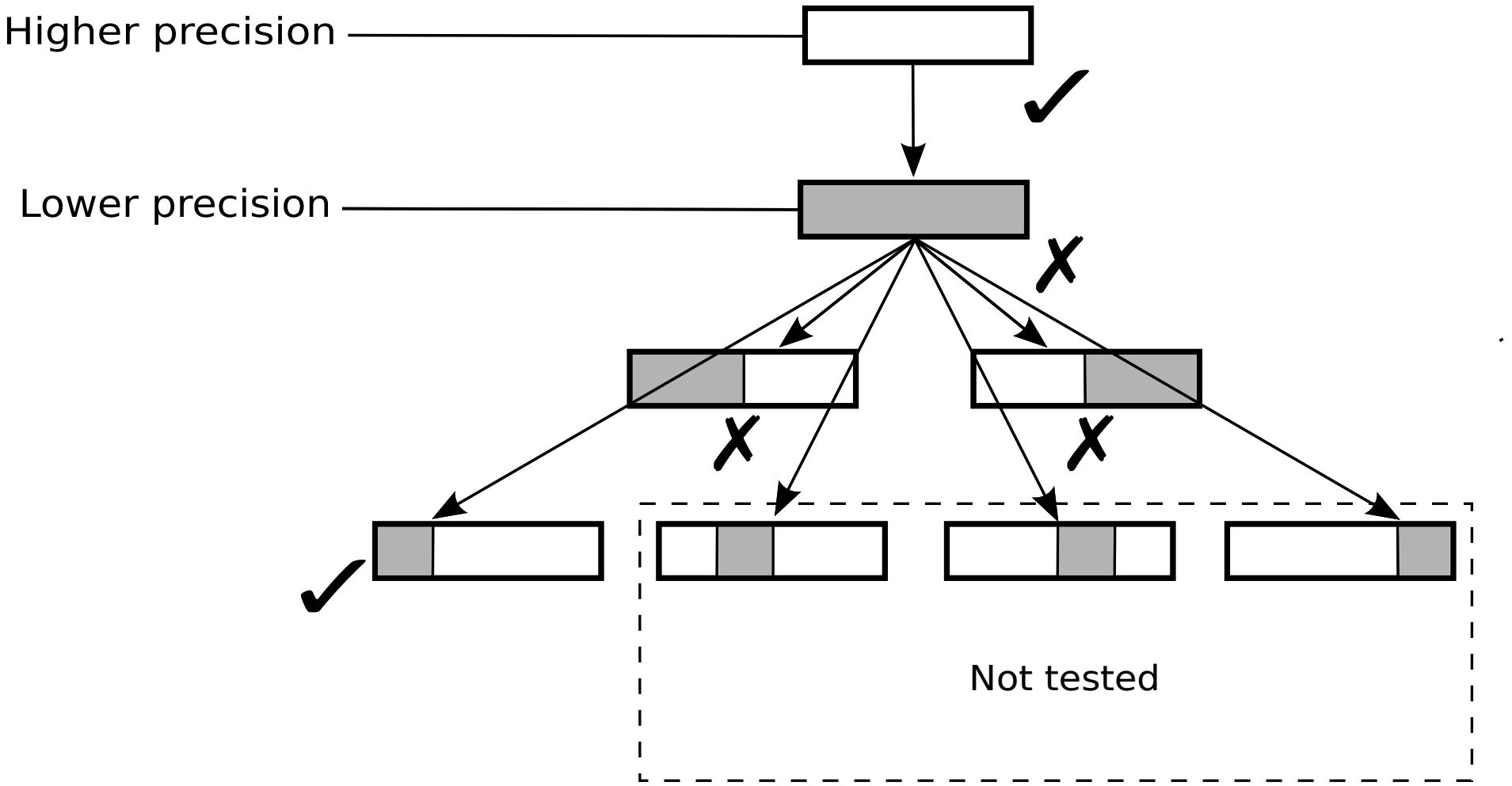
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller, 2009]



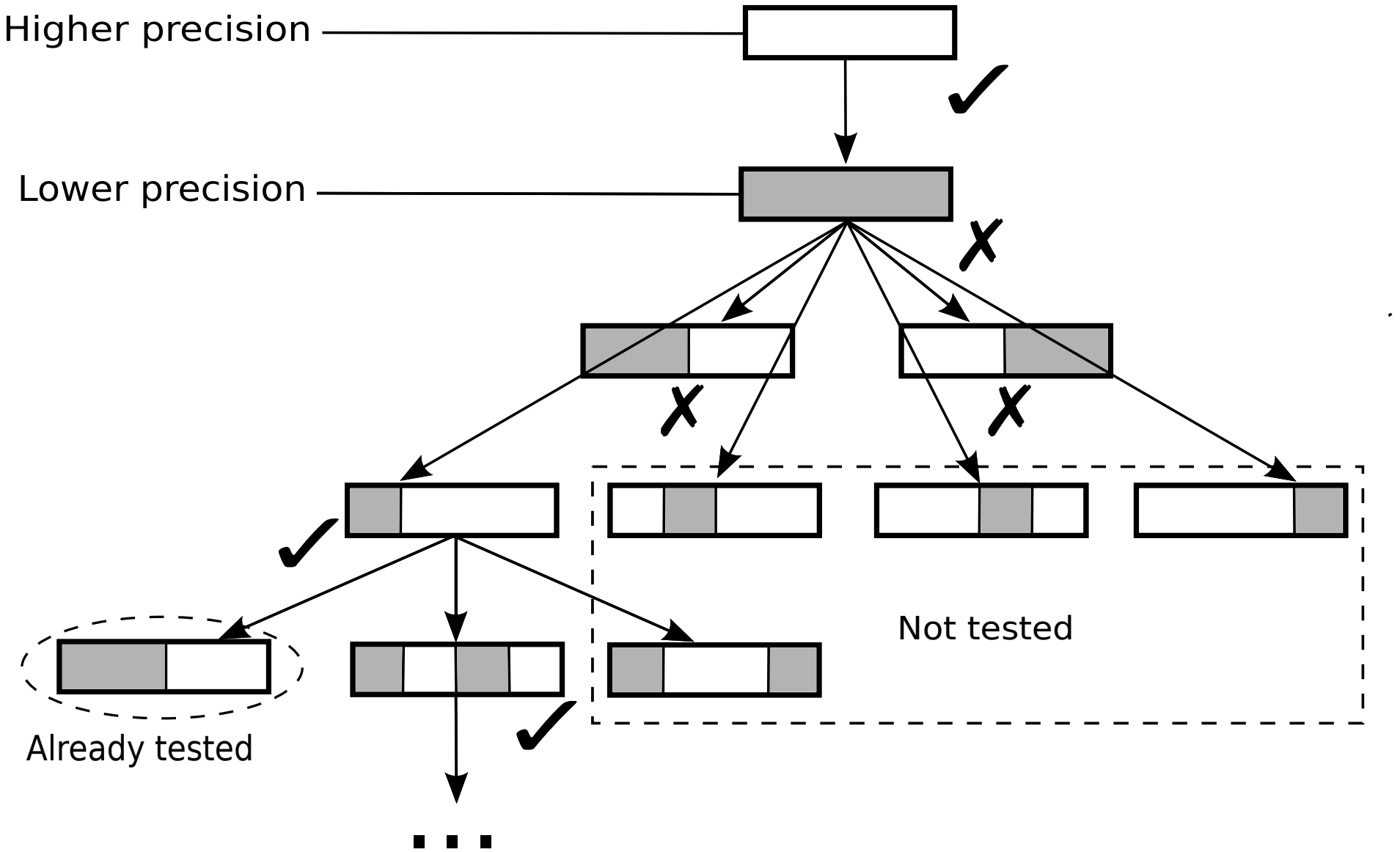
Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller, 2009]



Searching for a valid configuration with 2 types

Method based on the Delta Debug algorithm [Zeller, 2009]



Searching for a valid configuration: complexity

- We will not have the *best* configuration.
- But the mean complexity is $O(n\log(n))$ and in the worst case $O(n^2)$

Searching for a valid configuration: complexity

- We will not have the *best* configuration.
- But the mean complexity is $O(n\log(n))$ and in the worst case $O(n^2)$

Efficient way for finding a local maximum configuration

Searching for a valid configuration with 3 types

PROMISE with 2 types

- from a C/C++ program and an accuracy requirement on the results, provides a new program mixing single and double precision
- based on CADNA and the DeltaDebug (DD) algorithm

C: set of variables in double precision $\longrightarrow$ DD $\longrightarrow$ bipartition (C^s, C^d)

PROMISE with 3 types

- 2 executions of DD to provide a program mixing single, double, and quadruple precision

C: set of variables in quadruple precision $\longrightarrow$ DD $\longrightarrow$ (C_0^d, C^q) $\longrightarrow$ DD $\longrightarrow$ (C^s, C^d, C^q)

Experimental results

Benchmarks

- **Short programs:**
  - arclength computation
  - rectangle method for the computation of integrals
  - Babylonian method for square root
  - matrix multiplication
- **GNU Scientific Library:**
  - Fast Fourier Transform
  - sum of Taylor series terms
  - polynomial evaluation/solver
- **SNU NPB Suite:**
  - Conjugate Gradient method
  - Scalar Penta-diagonal solver

Requested accuracy: 4, 6, 8 and 10 digits

$\Rightarrow$ PROMISE has found a new configuration each time.

MICADO: simulation of nuclear cores (EDF)

- neutron transport iterative solver
- 11,000 C++ code lines

# Digits	# comp - # exec	# double - # float	Time (mm:ss)	Speed up	memory gain
10	83-51	19-32	88:56	1.01	1.00
8	80-48	18-33	85:10	1.01	1.01
6	69-37	13-38	71:32	1.20	1.44
5 4	3-3	0-51	9:58	1.32	1.62

MICADO: simulation of nuclear cores (EDF)

- neutron transport iterative solver
- 11,000 C++ code lines

# Digits	# comp - # exec	# double - # float	Time (mm:ss)	Speed up	memory gain
10	83-51	19-32	88:56	1.01	1.00
8	80-48	18-33	85:10	1.01	1.01
6	69-37	13-38	71:32	1.20	1.44
5 4	3-3	0-51	9:58	1.32	1.62

- Speed-up up to 1.32 and memory gain 1.62
- Mixed precision approach successful: speed-up 1.20 and memory gain 1.44

[S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière, Auto-tuning for floating-point precision with Discrete Stochastic Arithmetic, 2019]

Numerical validation of BLAS routines with perturbed data

Numerical validation is crucial... but costful ☹.

Can we accelarate the execution?

Can we avoid numerical validation
...and still get trustful results?

work carried out from discussions with Stef Graillat, Roman Iakymchuk, Toshiyuki Imamura, and Daichi Mukunoki

Definitions

Let $y = f(x)$ be an exact result and $\hat{y} = \hat{f}(x)$ be the associated computed result.

- The forward error is the difference between y and $\hat{y}$.
- The backward analysis tries to seek for Δx s.t. $\hat{y} = f(x + \Delta x)$.

  Δx is the backward error associated with $\hat{y}$.

  It measures the distance between the problem that is solved and the initial one.
- The condition number C of the problem is defined as:

$$C := \lim_{\varepsilon \to 0^+} \sup_{|\Delta x| \leq \varepsilon} \left[\frac{|f(x + \Delta x) - f(x)|}{|f(x)|} / \frac{|\Delta x|}{|x|} \right].$$

  It measures the effect on the result of data perturbation.

Error induced by perturbed data

The relative rounding error is denoted by $\mathbf{u}$.

- *binary64* format (double precision): $\mathbf{u} = 2^{-53}$
- *binary32* format (single precision): $\mathbf{u} = 2^{-24}$.

If the algorithm is backward-stable (*i.e.* the backward error is of the order of $\mathbf{u}$)

$$|f(x) - \hat{f}(x)|/|f(x)| \lesssim C\mathbf{u}.$$

If the input data are perturbed, *i.e.* the input data are not x but $\hat{x} = x(1+\delta)$, then one computes $\hat{f}(\hat{x})$ with

$$|f(x) - \hat{f}(\hat{x})|/|f(x)| \lesssim C(\mathbf{u} + |\delta|).$$

If $|\delta| \gg \mathbf{u}$, the rounding error generated by $\hat{f}$ is negligible w.r.t. $C|\delta|$.

Error induced by perturbed data

The relative rounding error is denoted by $\mathbf{u}$.

- *binary64* format (double precision): $\mathbf{u} = 2^{-53}$
- *binary32* format (single precision): $\mathbf{u} = 2^{-24}$.

If the algorithm is backward-stable (*i.e.* the backward error is of the order of $\mathbf{u}$)

$$|f(x) - \hat{f}(x)|/|f(x)| \lesssim C\mathbf{u}.$$

If the input data are perturbed, *i.e.* the input data are not x but $\hat{x} = x(1+\delta)$, then one computes $\hat{f}(\hat{x})$ with

$$|f(x) - \hat{f}(\hat{x})|/|f(x)| \lesssim C(\mathbf{u} + |\delta|).$$

If $|\delta| \gg \mathbf{u}$, the rounding error generated by $\hat{f}$ is negligible w.r.t. $C|\delta|$.

$\Rightarrow$ Estimating this rounding error may be avoided.

Using a classic routine in a CADNA code

A computation routine is executed in a code that is controlled using DSA.

Its input data are affected by errors (rounding errors and/or measurement errors).

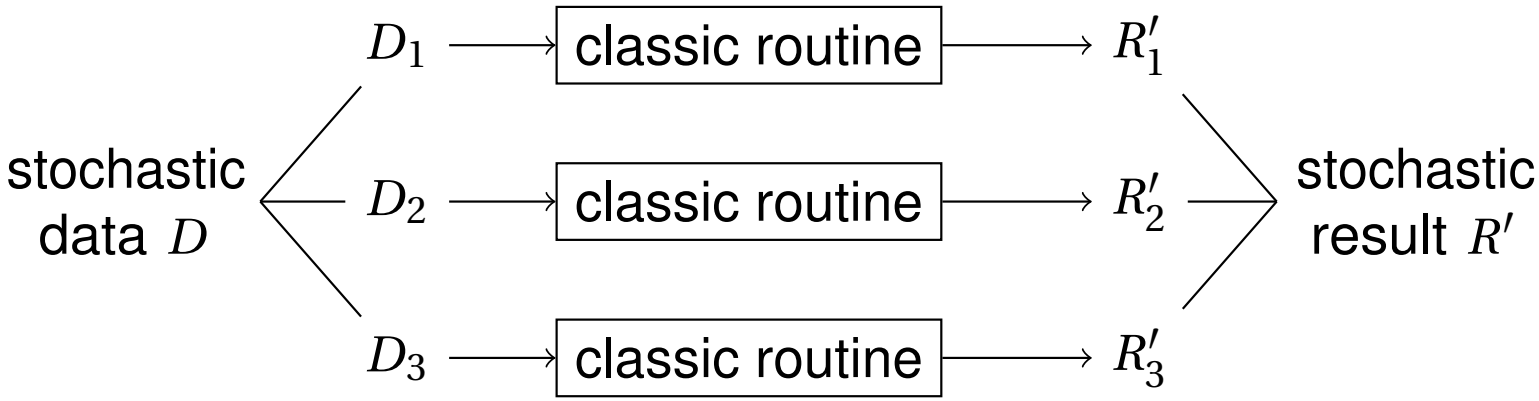
We compare 2 kinds of computation:

- with a call to a CADNA routine
- with 3 calls to a classic routine.

Computation with a call to a CADNA routine

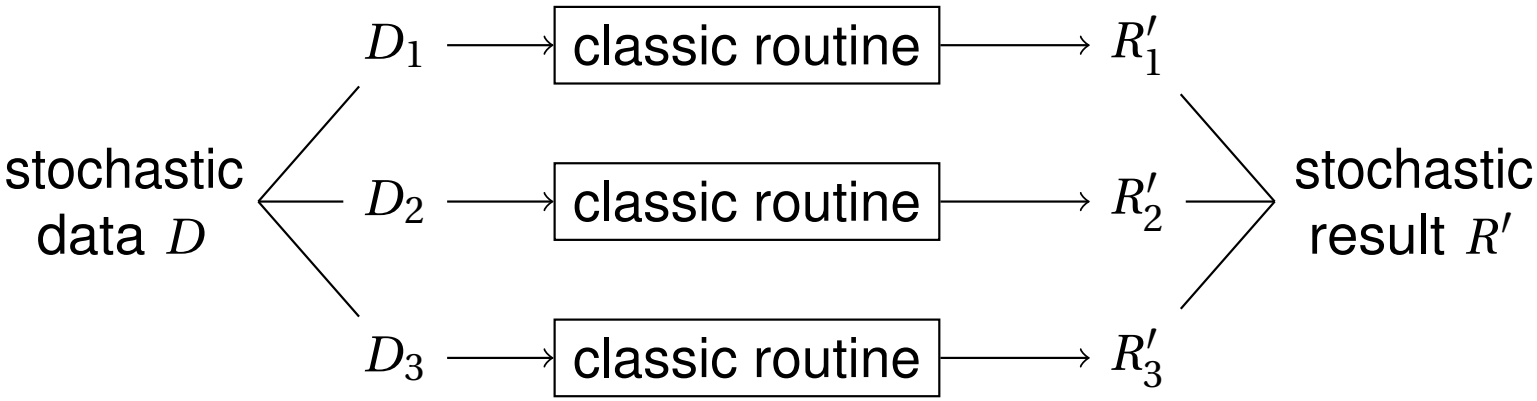
- D and R consist in stochastic arrays (each element is a triplet).
- Every arithmetic operation is performed 3 times with the random rounding mode.

Computation with 3 calls to a classic routine



- input data: 3 classic floating-point arrays D_1, D_2, D_3 created from the triplets of D
- We get 3 classic floating-point arrays R'_1, R'_2, R'_3.
- A stochastic array R' created from R'_1, R'_2, R'_3 can be used in the next parts of the code.

Computation with 3 calls to a classic routine



- input data: 3 classic floating-point arrays D_1, D_2, D_3 created from the triplets of D
- We get 3 classic floating-point arrays R'_1, R'_2, R'_3.
- A stochastic array R' created from R'_1, R'_2, R'_3 can be used in the next parts of the code.

$\Rightarrow$ we compare the number of correct digits (estimated by CADNA) in R and R'

Accuracy comparison for matrix multiplication

- Multiplication in double precision of square random matrices of size 500
- Each input value is perturbed using a CADNA function with a relative error δ.

δ	accuracy of R		accuracy difference between R & R'	
	mean	min-max	mean	max
1.e-14	13.9	9-15	2.5e-02	2
1.e-13	12.8	8-15	5.8e-03	1
1.e-12	11.9	7-14	4.2e-04	1
1.e-11	10.9	6-13	2.4e-05	1

- As the order of magnitude of δ $\nearrow$ the mean accuracy $\searrow$ by 1 digit.
- Low difference between the accuracy of R & R'

Accuracy comparison for matrix-vector multiplication

- Multiplication in double precision of a square random matrix of size 1000 with a vector

δ	accuracy of R		accuracy difference between R & R'	
	mean	min-max	mean	max
1.e-14	13.9	12-15	4.6e-02	1
1.e-13	12.7	11-14	7.0e-03	1
1.e-12	11.8	10-13	0	0
1.e-11	10.9	9-12	0	0

- As the order of magnitude of δ $\nearrow$ the mean accuracy $\searrow$ by 1 digit.
- The accuracy difference between R & R' remains low.
- All the results have the same accuracy if $\delta \geq 10^{-12}$

Performance comparison

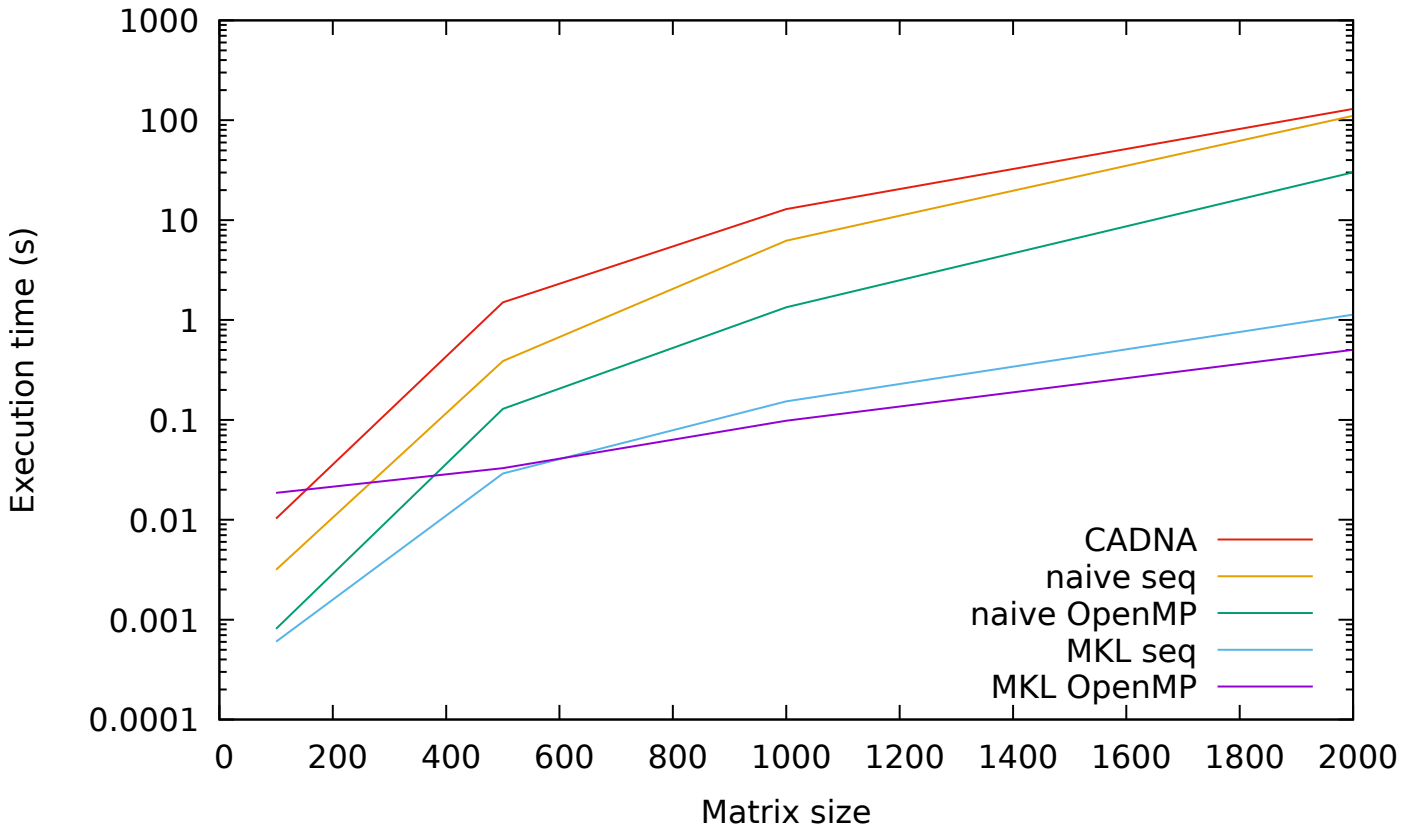
We compare the performance of the CADNA routine with codes using:

- a naive floating-point algorithm
- the MKL implementation.

In both cases: sequential and OpenMP 4 cores

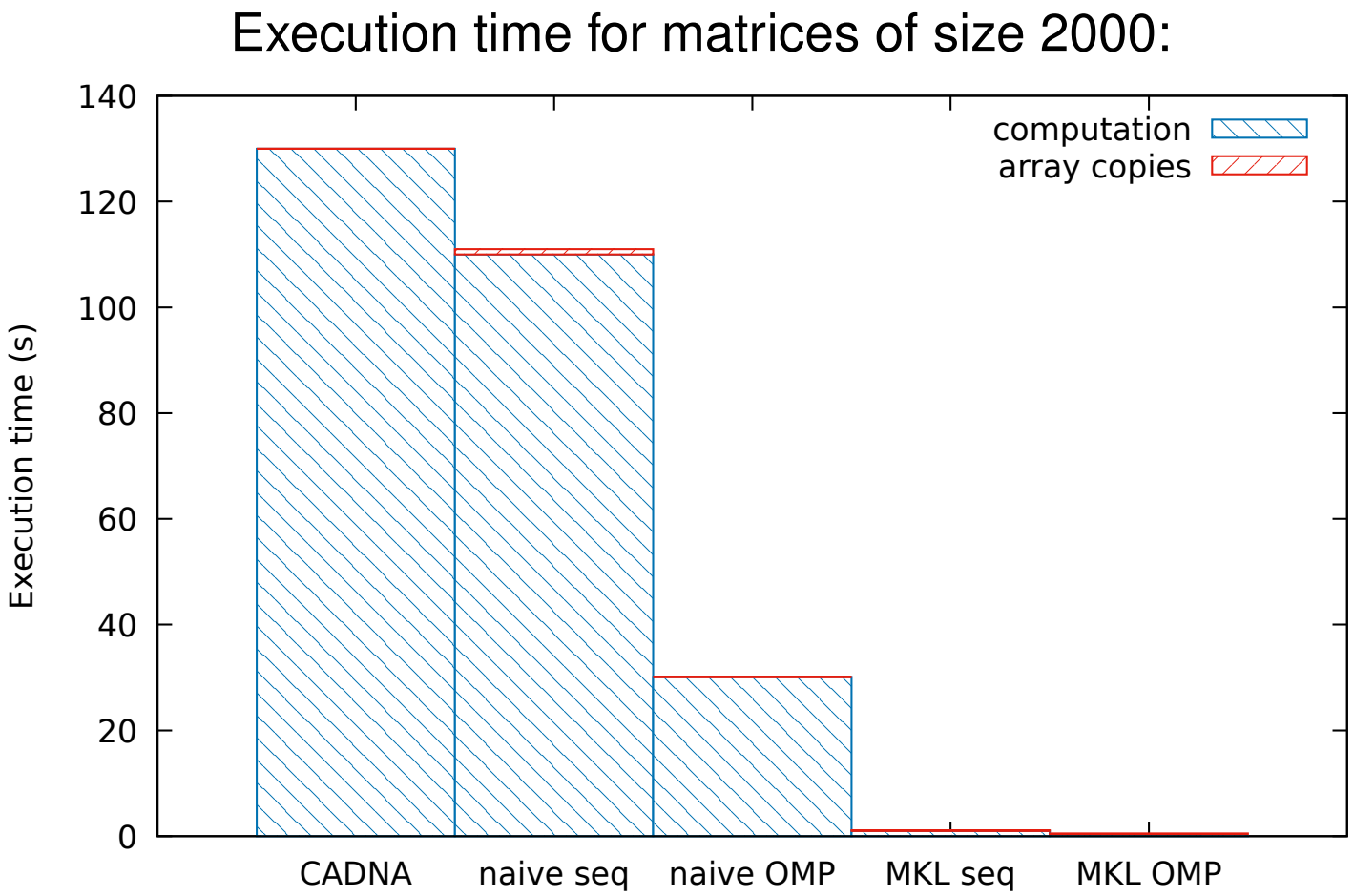
Performance for matrix multiplication

Execution time including matrix multiplications and array copies:



- Despite memory copies, the codes using 3 classic matrix multiplications perform better than the CADNA routine.
- For matrices of size 2000, the MKL OpenMP implementation outperforms the CADNA routine by a factor 250.

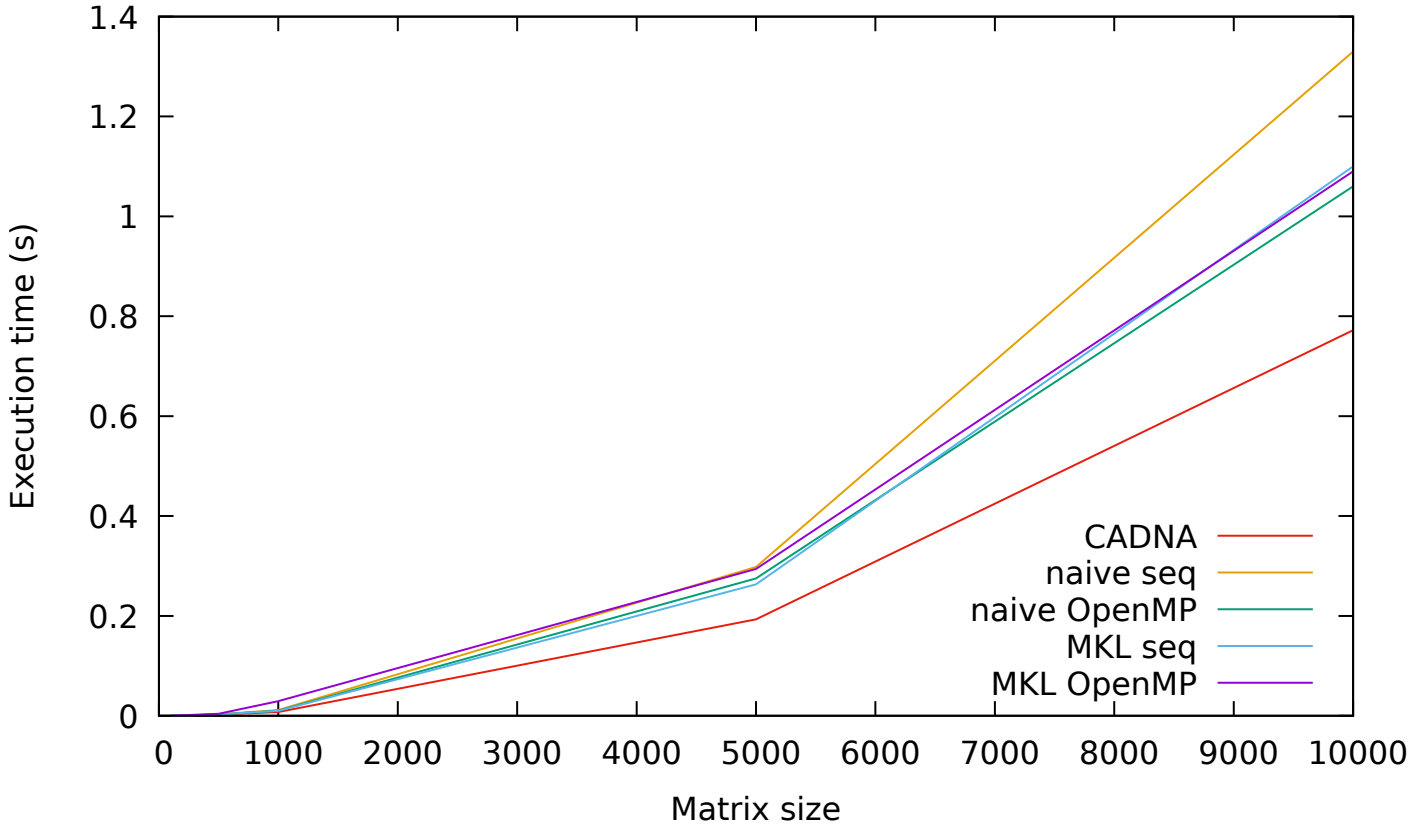
Performance for matrix multiplication



- The impact of array copies on the execution time is negligible.

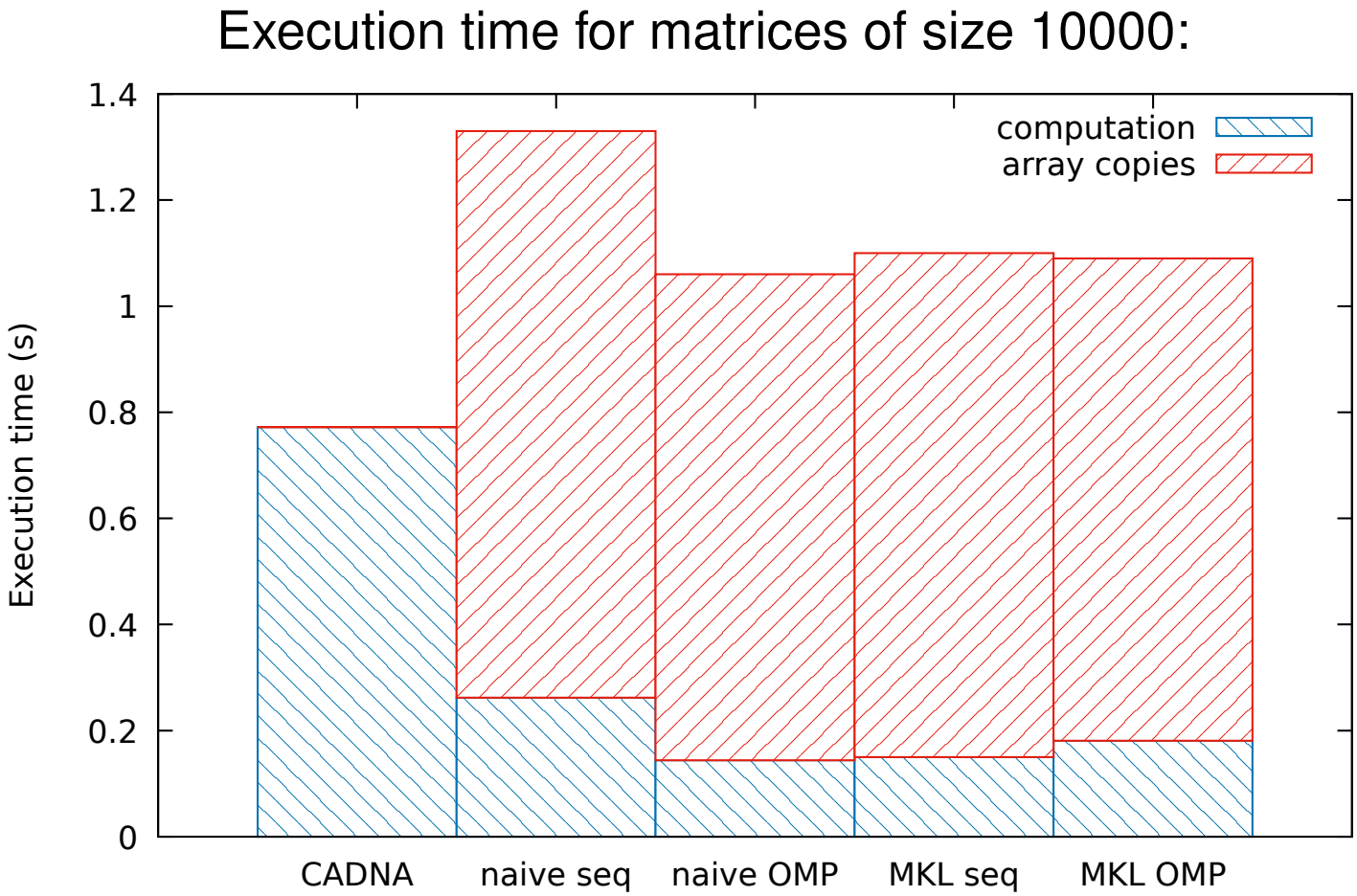
Performance for matrix-vector multiplication

Execution time including matrix-vector multiplications and array copies:



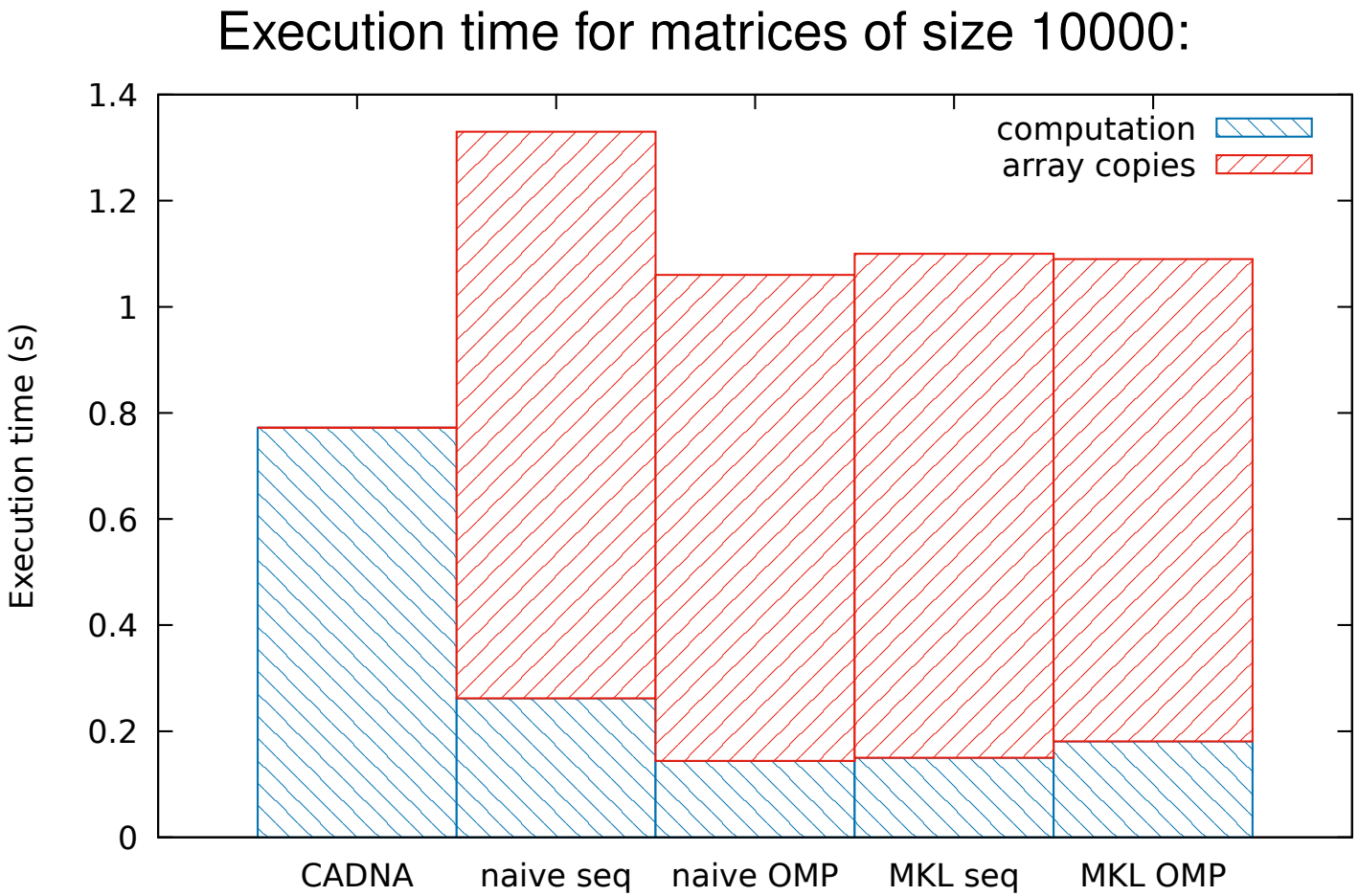
- The CADNA routine performs better than the other codes.

Performance for matrix-vector multiplication



- Except with the CADNA routine, the main part of the execution time is spent in array copies.

Performance for matrix-vector multiplication



⇒ Performance gain if a computation-intensive CADNA routine (BLAS 3) is replaced by classic floating-point routines.

Instability detection

Without CADNA:

- numerical instabilities are not detected ☹
- results with no correct digits appear as numerical noise ☺

Example: matrix multiplication with catastrophic cancellations

Input data: square matrices A & B of size 10 in double precision

- 1st line of A: $[1,...,1,-1,...,-1]$ (1st half: 1, 2nd half: -1)
- each element of B set to 1
- A and B pertubed with a relative error $\delta = 10^{-12}$

Results: $C = A * B$ with CADNA, $C' = A * B$ without CADNA

- 1st line of C and C': @.0 (numerical noise, triplet with no common digits)

With CADNA:

- 10 catastrophic cancellations are detected.

Accuracy improvement with CADNA

Example: Gauss algorithm with pivoting

Input data:

We solve in single precision the system $Ax = b$ with

$$A = \begin{pmatrix} 21 & 130 & 0 & 2.1 \\ 13 & 80 & 4.74\ 10^{8} & 752 \\ 0 & -0.4 & 3.9816\ 10^{8} & 4.2 \\ 0 & 0 & 1.7 & 9\ 10^{-9} \end{pmatrix} \quad b = \begin{pmatrix} 153.1 \\ 849.74 \\ 7.7816 \\ 2.6\ 10^{-8} \end{pmatrix}$$

A and b pertubed with a relative error $\delta = 10^{-6}$

Results: x with CADNA, x' without CADNA

$$x = \begin{pmatrix} 0.100\text{E+}001 \\ 0.999\text{E+}000 \\ 0.999999\text{E-}008 \\ 0.999999\text{E+}000 \end{pmatrix} \quad x' = \begin{pmatrix} @.0 \\ @.0 \\ @.0 \\ 0.999999\text{E+}000 \end{pmatrix} \quad x_{exact} = \begin{pmatrix} 1 \\ 1 \\ 10^{-8} \\ 1 \end{pmatrix}$$

Accuracy improvement with CADNA

Example: Gauss algorithm with pivoting

Results: x with CADNA, x' without CADNA

$$x = \begin{pmatrix} 0.100\text{E+001} \\ 0.999\text{E+000} \\ 0.999999\text{E-008} \\ 0.999999\text{E+000} \end{pmatrix} \quad x' = \begin{pmatrix} @.0 \\ @.0 \\ @.0 \\ 0.999999\text{E+000} \end{pmatrix} \quad x_{exact} = \begin{pmatrix} 1 \\ 1 \\ 10^{-8} \\ 1 \end{pmatrix}$$

Test for pivoting: if $(|A_{i,j}| > p_{max})$ ...

With CADNA a non-significant element is not chosen as a pivot.

Instabilities detected by CADNA:

```
There are 3 numerical instabilities
1 UNSTABLE BRANCHING(S)
1 UNSTABLE INTRINSIC FUNCTION(S)
1 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S)
```

Conclusions & Perspectives

In a code controlled using CADNA, if a computation-intensive routine (BLAS 3) is run with perturbed data,

- a classic BLAS routine can be executed 3 times instead of the CADNA routine with almost no accuracy difference on the results
- the performance gain can be high with a BLAS routine from an optimized library
- but we loose the instability detection.

The same conclusions would be valid with an HPC code using MPI.

In the same conditions (computation-intensive routine & perturbed data) CADNA-MPI routine $\Rightarrow$ optimized floating-point MPI routines.

Thanks for your attention!

On the number of runs

2 or 3 runs are enough. To increase the number of runs is not necessary.

From the model, to increase by 1 the number of exact significant digits given by $C_{\overline{R}}$, we need to multiply the size of the sample by 100.

Such an increase of N will only point out the limit of the model and its error without really improving the quality of the estimation.

It has been shown that $N = 3$ is the optimal value. [Chesneaux & Vignes, 1988]

On the probability of the confidence interval

With $\beta = 0.05$ and $N = 3$,

- the probability of overestimating the number of exact significant digits of at least 1 is 0.054%
- the probability of underestimating the number of exact significant digits of at least 1 is 29%.

By choosing a confidence interval at 95%, we prefer to guarantee a minimal number of exact significant digits with high probability (99.946%), even if we are often pessimistic by 1 digit.