Can we avoid rounding-error estimation in HPC codes and still get trustworthy results?

Fabienne Jézéquel[1], Stef Graillat[1], Daichi Mukunoki[2], Toshiyuki Imamura[2], Roman Iakymchuk[1]

[1]LIP6, Sorbonne Université, CNRS, Paris, France

[2]RIKEN Center for Computational Science, Kobe, Japan

13th International Workshop on Numerical Software Verification 2020
20-21 July 2020

Introduction

Current computers: a high number of floating-point operations performed ☺
Each of them can lead to a rounding error ☹

Introduction

Current computers: a high number of floating-point operations performed ☺
Each of them can lead to a rounding error ☹

⇒ Numerical validation is crucial

Introduction

Current computers: a high number of floating-point operations performed ☺
Each of them can lead to a rounding error ☹

⇒ Numerical validation is crucial ...but costful ☹

- execution time overhead
- development cost induced by the application of numerical validation methods to HPC codes

Introduction

Current computers: a high number of floating-point operations performed ☺
Each of them can lead to a rounding error ☹

$\Rightarrow$ Numerical validation is crucial ...but costful ☹

- execution time overhead
- development cost induced by the application of numerical validation methods to HPC codes

> Can we address this cost problem
> ...and still get trustworthy results?

Yes, when the input data is affected by rounding and/or measurement errors.

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:

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

  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 $\mathbf{u}$)

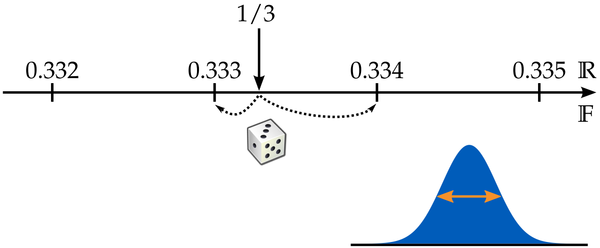
$$|f(x) - \hat{f}(x)|/|f(x)| \lesssim C\mathbf{u}.$$

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.



- each operation is executed 3 times with a random rounding mode: $R \rightarrow (R_1, R_2, R_3)$ where each result R_i is rounded up or down with the same probability
- the number of correct digits in the results is estimated using Student's test with the confidence level 95%
- operations are executed synchronously
  - $\Rightarrow$ detection of numerical instabilities

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

CADNA allows one to use DSA in any scientific program written in C, C++ or Fortran.

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

CADNA allows one to use DSA in any scientific program written in C, C++ or Fortran.

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.

Performance overhead: $\times 4$ memory, $\approx \times 10$ execution time

Combining DSA and standard floating-point arithmetic

Computation routines are executed in a code that is controlled using DSA.

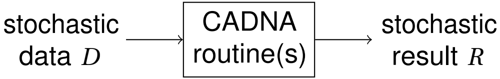
Their input data are affected by errors (rounding errors and/or measurement errors).

Combining DSA and standard floating-point arithmetic

Computation routines are executed in a code that is controlled using DSA.

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

Computation with a call to CADNA routines:



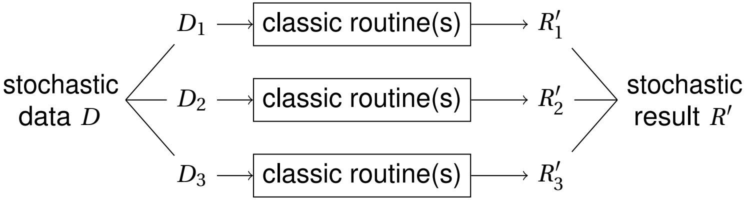
- D and R consist in stochastic arrays (each element is a triplet).

Combining DSA and standard floating-point arithmetic

Computation routines are executed in a code that is controlled using DSA.

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

Computation with 3 calls to classic routines:



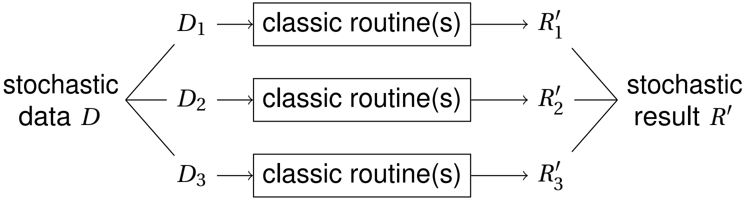
- input data: 3 classic floating-point arrays D_1, D_2, D_3 created from the triplets of D
- We get 3 classic floating-point arrays R'_1, R'_2, R'_3.
- A stochastic array R' created from R'_1, R'_2, R'_3 can be used in the next parts of the code.

Combining DSA and standard floating-point arithmetic

Computation routines are executed in a code that is controlled using DSA.

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

Computation with 3 calls to classic routines:



- 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
Experimental setup

Each random input value is perturbed with a relative error δ.

For $i = 1, \ldots, n^2$ (matrix mult.) or for $i = 1, ..., n$ (matrix-vector mult.) we analyze:

- the accuracy C_{R^i} of the element R^i of R
- the accuracy $C_{R'^i}$ of the element R'^i of R'
- $\Delta^i = \left| C_{R^i} - C_{R'^i} \right|$

Accuracy comparison

in double precision

δ	accuracy of R		accuracy difference between R & R'	
	mean	min-max	mean	max
Multiplication of matrices of size 500				
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
Multiplication of a matrix of size 1000 with a vector				
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 $\delta \nearrow$ the mean accuracy $\searrow$ by 1 digit.
- Low difference between the accuracy of R & R'

Performance comparison

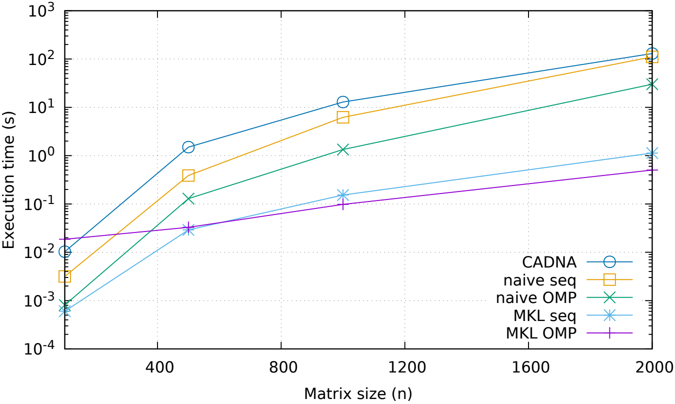
We compare the performance of the CADNA routine with codes using:

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

In both cases: sequential and OpenMP 4 cores

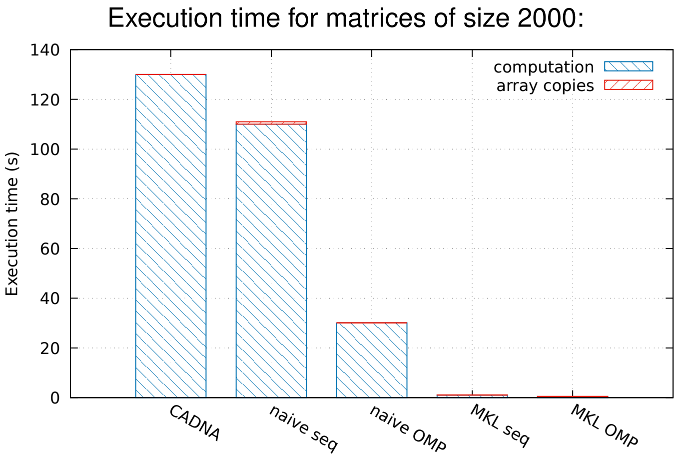
Performance for matrix multiplication

Execution time including matrix multiplications and array 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 294 (this gain increases on many-cores).

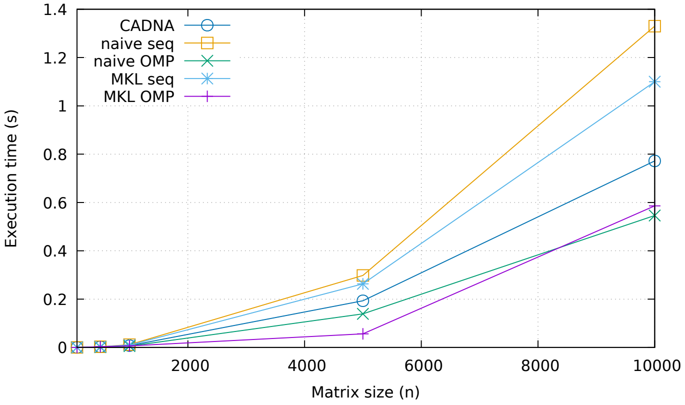
Performance for matrix multiplication



- Most of the execution time is spent in matrix multiplication.

Performance for matrix-vector multiplication

Execution time including matrix-vector multiplications and array copies:



- The CADNA routine performs better than the other sequential codes.

Performance for matrix-vector multiplication

- Except with the CADNA routine, the main part of the execution time is spent in array copies.
- Both computation and array copies are parallelized in the OpenMP codes.

Conclusions/Perspectives

- In a code controlled using CADNA, if computation-intensive routines are run with perturbed data,
  - classic BLAS routines can be executed 3 times instead of the CADNA routines with almost no accuracy difference on the results
  - the performance gain can be high with BLAS routines from an optimized library
  - but we lose the instability detection.

- The same conclusions would be valid with an HPC code using MPI.
  CADNA-MPI routines ⇒ optimized floating-point MPI routines.

- Application of our approach to real-life examples with realistic data sets.

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.