Fast rounding error estimation for compute-intensive operations using standard floating-point arithmetic

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

Rencontres Arithmétiques du GdR Informatique Mathématique (RAIM)
27-28 May 2021

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 ☹

⇒ 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.

Overview

1. Estimation of rounding errors:
   Discrete Stochastic Arithmetic (DSA) and the CADNA library
2. Error induced by perturbed data
3. Our approach: combining DSA and standard floating-point arithmetic
4. Numerical experiments
5. Pros and cons of our approach

Discrete Stochastic Arithmetic (DSA) [J. Vignes, 2004]

Classic arithmetic

$\boxed{A \oplus B \longrightarrow R}$

R =3.14237654356891

Stochastic arithmetic

Random rounding

$A_1 \oplus B_1$  $\longrightarrow R_1$

$A_2 \oplus B_2$  $\longrightarrow R_2$

$A_3 \oplus B_3$  $\longrightarrow R_3$

R_1 =**3.14**1354786390989
R_2 =**3.14**3689456834534
R_3 =**3.14**2579087356598

- each operation executed 3 times with a random rounding mode

Discrete Stochastic Arithmetic (DSA) [J. Vignes, 2004]

Classic arithmetic

$$\boxed{A \oplus B \longrightarrow R}$$

R =3.14237654356891

Stochastic arithmetic

Random rounding

$A_1 \oplus B_1$  $\longrightarrow R_1$

$A_2 \oplus B_2$  $\longrightarrow R_2$

$A_3 \oplus B_3$  $\longrightarrow R_3$

R_1 =**3.14**1354786390989
R_2 =**3.14**3689456834534
R_3 =**3.14**2579087356598

- each operation executed 3 times with a random rounding mode
- number of correct digits in the results estimated using Student's test with the confidence level 95%

Classic arithmetic

$$\boxed{A \oplus B \longrightarrow R}$$

R =3.14237654356891

Stochastic arithmetic

Random rounding

$A_1 \oplus B_1$  $\longrightarrow R_1$

$A_2 \oplus B_2$  $\longrightarrow R_2$

$A_3 \oplus B_3$  $\longrightarrow R_3$

R_1 =**3.14**1354786390989
R_2 =**3.14**3689456834534
R_3 =**3.14**2579087356598

- each operation executed 3 times with a random rounding mode
- number of correct digits in the results estimated using Student's test with the confidence level 95%
- operations executed synchronously
  - $\Rightarrow$ detection of numerical instabilities

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

CADNA enables one to estimate the numerical quality of results and detect numerical instabilities in C, C++ or Fortran codes.

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

CADNA enables one to estimate the numerical quality of results and detect numerical instabilities in C, C++ or Fortran codes.

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.

CADNA usually used to control an entire scientific application.

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

Outline

1. Discrete Stochastic Arithmetic (DSA) and the CADNA library
2. Error induced by perturbed data
3. Our approach: combining DSA and standard floating-point arithmetic
4. Numerical experiments
5. Pros and cons of our approach

Error induced by perturbed data

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.

Outline

1 Discrete Stochastic Arithmetic (DSA) and the CADNA library

2 Error induced by perturbed data

3 Our approach: combining DSA and standard floating-point arithmetic

4 Numerical experiments

5 Pros and cons of our approach

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).

We compare 2 kinds of computation:

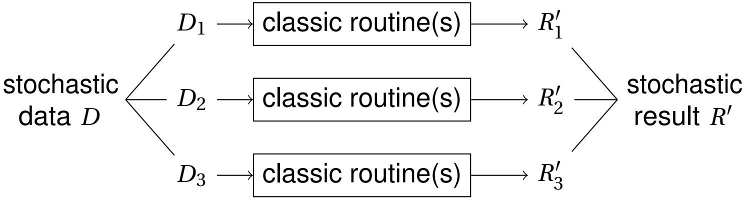
- with a call to CADNA routines
- with 3 calls to classic routines.

Computation with a call to CADNA routines

stochastic data D $\longrightarrow$ $\boxed{\text{CADNA routine(s)}}$ $\longrightarrow$ stochastic result R

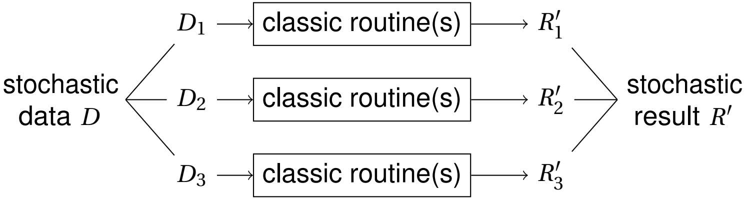
- D and R consist in stochastic arrays (each element is a triplet).
- Every arithmetic operation is performed 3 times with the random rounding mode.

Our approach: 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.

Our approach: 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'

Outline

1. Discrete Stochastic Arithmetic (DSA) and the CADNA library
2. Error induced by perturbed data
3. Our approach: combining DSA and standard floating-point arithmetic
4. Numerical experiments
5. Pros and cons of our approach

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 for matrix multiplication

Multiplication of square random matrices of size 500:

δ	accuracy of R		accuracy difference between R & R'	
	mean	min-max	mean	max
		double precision		
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
		single precision		
1.e-6	5.6	1-7	2.3e-1	2
1.e-5	4.8	0-7	1.9e-2	2
1.e-4	3.7	0-6	2.8e-3	1
1.e-3	2.8	0-5	2.8e-4	1

- As the order of magnitude of $\delta \nearrow$ the mean accuracy $\searrow$ by 1 digit
- High perturbation in single precision $\Rightarrow$ low accuracy on the results
- Low difference between the accuracy of R & R'

Accuracy comparison for matrix-vector multiplication

Multiplication 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
double precision				
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
single precision				
1.e-6	5.5	3-7	3.2e-1	2
1.e-5	4.8	2-6	2.4e-2	1
1.e-4	3.7	1-5	7.0e-3	1
1.e-3	2.8	0-4	1.0e-3	1

- As the order of magnitude of $\delta \nearrow$ the mean accuracy $\searrow$ by 1 digit
- High perturbation in single precision $\Rightarrow$ low accuracy on the results
- The accuracy difference between R & R' remains low
  (in double precision, all the results have the same accuracy if $\delta \geq 10^{-12}$)

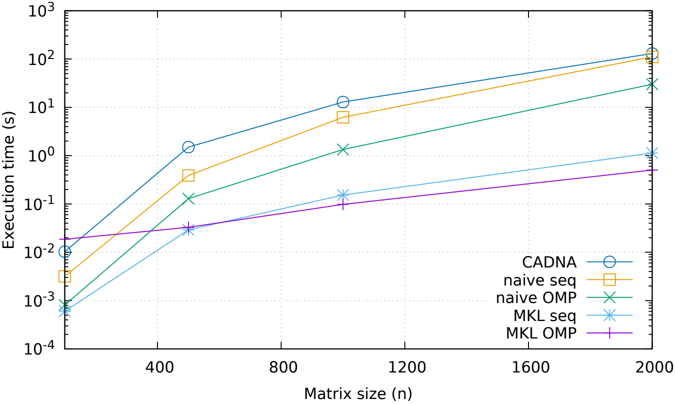
Performance comparison

- We compare the performance of the CADNA routine with codes using:
  - a naive floating-point algorithm
  - the Intel MKL implementation.

  In both cases: sequential and OpenMP 4 cores
- Array copies except with CADNA
- Both computation and array copies parallelized in the OpenMP codes

Performance for matrix multiplication

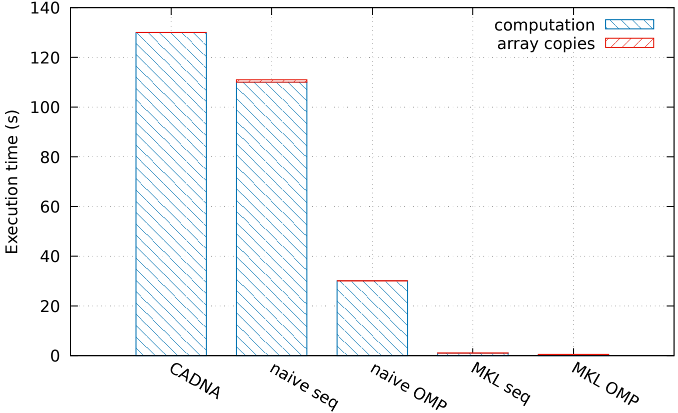
Execution time including matrix multiplications and array copies:



- 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 294.

Performance for matrix multiplication

Execution time for matrices of size 2000:



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

Performance for matrix multiplication

CADNA vs our approach with MKL OMP

Core i7-8650U (1.9 GHz, 4 cores), n=2000:

	CADNA	Proposed w/ MKL OMP	Speedup
Comp	130	0.393	331x
Copy	–	0.0495	–
Total	130	0.4425	294x

Dual-socket Xeon Gold 6126 (2.6 GHz, 12 cores×2), n=5000:

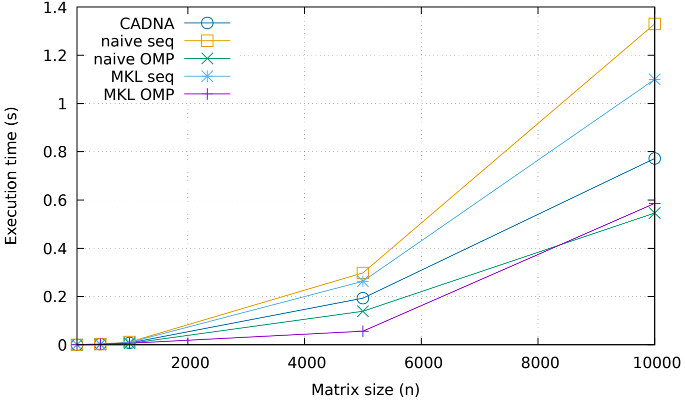
	CADNA	Proposed w/ MKL OMP	Speedup
Comp	2520	0.563	4476x
Copy	–	0.0889	–
Total	2520	0.652	3865x

On large scale:

- the performance gain increases
- the array copy cost becomes visible

Performance for matrix-vector multiplication

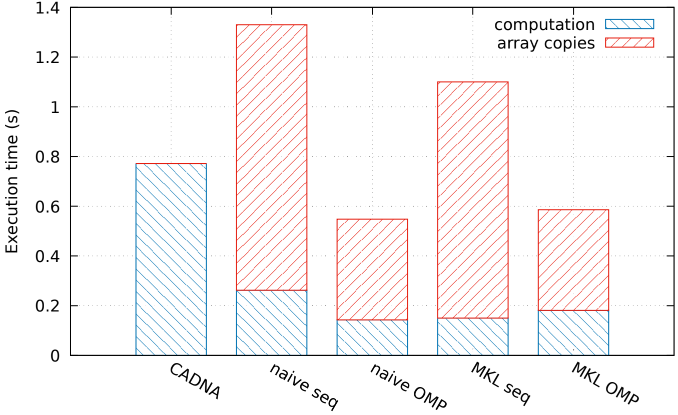
Execution time including matrix-vector multiplications and array copies:



- The CADNA routine performs better than the other sequential codes.
- From a certain matrix size, the OpenMP codes that use classic floating-point arithmetic perform better than the CADNA code.

Performance for matrix-vector multiplication

Execution time for matrices of size 10000:



- In the codes that use classic floating-point arithmetic the main part of the execution time is spent in array copies.
- Worst case here: if several BLAS routines continuously used, array copy cost w.r.t. total execution time ↘

Outline

1 Discrete Stochastic Arithmetic (DSA) and the CADNA library

2 Error induced by perturbed data

3 Our approach: combining DSA and standard floating-point arithmetic

4 Numerical experiments

5 Pros and cons of our approach

Pros and cons

Pros

- performance gain:
  - DSA operations are avoided
  - use of vendor optimized libraries
- applicability:
  - no code translation to a CADNA version

Cons

we lose CADNA features:

- instability detection
- accuracy improvement:
  in linear system solving, a non-significant element is not chosen as a pivot.

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.

Conclusions/Perspectives

- In a code controlled using CADNA, if computation-intensive routines are run with perturbed data
  - CADNA routines ⇒ classic BLAS routines
    with almost no accuracy difference on the results
  - high performance gain 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.
  In the same conditions (computation-intensive routines & perturbed data)
  CADNA-MPI routines ⇒ optimized floating-point MPI routines.
- Application of our approach to real-life examples with realistic data sets.

F. Jézéquel, S. Graillat, D. Mukunoki, T. Imamura, R. Iakymchuk, Can we avoid rounding-error estimation in HPC codes and still get trustworthy results?, NSV'20

`https://hal.archives-ouvertes.fr/hal-02925976v1`

Thanks for your attention!