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

Fabienne Jézéquel, Stef Graillat, Daichi Mukunoki, Toshiyuki Imamura, Roman Iakymchuk

▶ **To cite this version:**

Fabienne Jézéquel, Stef Graillat, Daichi Mukunoki, Toshiyuki Imamura, Roman Iakymchuk. Can we avoid rounding-error estimation in HPC codes and still get trustworthy results?. NSV'20, 13th International Workshop on Numerical Software Verification, Jul 2020, Los Angeles, CA, United States. hal-02925976

HAL Id: hal-02925976
https://hal.science/hal-02925976v1

Submitted on 31 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Fabienne Jézéquel[1,2], Stef Graillat[1], Daichi Mukunoki[3], Toshiyuki Imamura[3], Roman Iakymchuk[1,4]

[1] Sorbonne Université, CNRS, LIP6, Paris, France
{fabienne.jezequel,stef.graillat,roman.iakymchuk}@sorbonne-universite.fr
[2] Université Panthéon-Assas, Paris, France
[3] RIKEN Center for Computational Science, Kobe, Japan
{daichi.mukunoki,imamura.toshiyuki}@riken.jp
[4] Fraunhofer ITWM, Kaiserslautern, Germany

Abstract. Numerical validation enables one to ensure the reliability of numerical computations that rely on floating-point operations. Discrete Stochastic Arithmetic (DSA) makes it possible to validate the accuracy of floating-point computations using random rounding. However, it may bring a large performance overhead compared with the standard floating-point operations. In this article, we show that with perturbed data it is possible to use standard floating-point arithmetic instead of DSA for the purpose of numerical validation. For instance, for codes including matrix multiplications, we can directly utilize the matrix multiplication routine (GEMM) of level-3 BLAS that is performed with standard floating-point arithmetic. Consequently, we can achieve a significant performance improvement by avoiding the performance overhead of DSA operations as well as by exploiting the speed of highly-optimized BLAS implementations. Finally, we demonstrate the performance gain using Intel MKL routines compared against the DSA version of BLAS routines.

Keywords: BLAS, Discrete Stochastic Arithmetic (DSA), floating-point arithmetic, numerical validation, rounding errors.

1 Introduction

Numerical simulations rely on finite precision arithmetic. This means that each elementary operation (like addition or multiplication) is potentially subject to rounding errors. The existence of rounding errors may cause a catastrophic consequence when meaningless results are computed owing to their accumulation, in particular in large-scale computation on supercomputers. In addition, there is a reproducibility issue: as floating-point computations are non-associative, different results may be computed even with the same code and the same input if the order of the computation is changed. As a consequence, it makes it difficult for us to distinguish bugs from rounding errors in software development. Therefore, we can see a strong motivation for analyzing the numerical quality of computed results. Moreover, the recent advances in mixed-precision techniques with precision reduction [3,12], which intend to achieve better speed as well as

energy efficiency, assist the importance of understanding the effect of rounding errors.

Several approaches exist for addressing issues caused by rounding errors. Backward error analysis [13,21] provides error bounds on the computed solutions of linear problems. This approach is used for instance to verify the numerical quality of solutions of linear systems in linear algebra libraries and in particular in the LAPACK library [2]. Interval arithmetic [1,16], which briefly consists in performing floating-point operations on intervals instead of scalars, provides guaranteed error bounds on computed results. However, a naive application of interval arithmetic in a code may result in a gross overestimation of the errors. Therefore, interval arithmetic is usually used together with a special algorithm for each numerical method.

Numerical validation can also be performed with a probabilistic approach based on several executions of the program to control. While interval arithmetic ensures the number of correct digits, probabilistic methods estimate it. One of the advantages of the probabilistic approach is that it is applicable for any floating-point computations without any changes in the algorithms. This study focuses on a probabilistic method for numerical validation, discrete stochastic arithmetic (DSA), which estimates rounding errors using random rounding. DSA is implemented in the CADNA library [6,15], which can control the numerical quality of codes using half, single, double and/or quadruple precision, as well as in the SAM library [11], which can be used in arbitrary precision codes. Other tools rely on a probabilistic approach to control rounding errors: MCALIB [9], Verificarlo [5], and Verrou [10]. While DSA is a synchronous method and requires three executions of each operation, the other tools rely on an asynchronous approach which requires more executions (at least 50 for MCALIB). Another advantage of DSA is its ability to detect numerical instabilities during the execution.

In this article, we consider the common situation when the input data are affected by rounding and/or measurement errors. We address, at the same time, the problem of numerical quality of computed results, the performance overhead of existing numerical validation methods, as well as the development cost induced by their application to HPC codes. Thus, we respond to these three challenging issues with the following contributions.

- We study numerical validation in case of perturbed data in HPC codes on examples of key Basic Linear Algebra Subprograms (BLAS) routines. These routines often appear in HPC codes consuming significant part of their execution time.
- In case of perturbed data, we propose an alternative approach that uses the standard floating-point arithmetic, instead of DSA, to perform efficient computations and still obtain trustworthy results.
- This novel approach shows outstanding performance as well as simplifying performance optimization as we directly rely on existing user-implemented codes or highly-optimized vendor-provided implementations. Hence, this approach is suitable for high-performance computations.

The remaining part of this paper is organized as follows. Section 2 introduces the principles of DSA and describes its implementations. Section 3 presents the

error induced by perturbed data. Section 4 proposes the utilization of the standard floating-point arithmetic in numerical validation. Then, we demonstrate our approach on matrix-vector multiplication (GEMV) and matrix-matrix multiplication (GEMM) using Intel MKL compared against DSA with the reference BLAS code: Section 5 is devoted to the accuracy comparison, while Section 6 presents the performance evaluation. Section 7 discusses the pros and cons of the proposed approach. Finally, conclusions are given in Section 8.

2 Discrete Stochastic Arithmetic (DSA)

2.1 DSA in a Nutshell

The CESTAC method [19] enables one to estimate the rounding error propagation which occurs with floating-point arithmetic. This probabilistic method uses a random rounding mode: at each elementary operation, the result is rounded up (towards $+\infty$) or down (towards $-\infty$) with the probability of 50 %. Hence, with this random rounding mode, the same program run several times provides different results. Therefore, the computer's deterministic arithmetic is replaced by a stochastic arithmetic, where each arithmetic operation is performed N times before the next one is executed. The CESTAC method supplies us with N samples $R_1, \ldots, R_N$ of the computed result R. The value of the computed result $\overline{R}$ is then chosen as the mean value of $\{R_i\}$ and, if no overflow occurs, its number of correct digits (*i.e.* its number of digits not affected by rounding errors) can be estimated [4,19] as

$$C_R = \log_{10}\left(\frac{\sqrt{N}\,|\overline{R}|}{\sigma\tau_\beta}\right) \text{ with } \overline{R} = \frac{1}{N}\sum_{i=1}^{N} R_i,\ \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(R_i - \overline{R}\right)^2. \quad (1)$$

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a confidence level $1-\beta$. In practice, $\beta = 5\%$ and $N = 3$. Therefore the number of correct digits is estimated within a $1-\beta = 95\%$ confidence interval. It has been shown [4,19] that $N = 3$ is in some reasonable sense the optimal sample size. The estimation with $N = 3$ is more reliable than with $N = 2$ and increasing N does not significantly improve the quality of the estimation.

If both operands in a multiplication or the divisor in a division have no correct digits, the validity of C_R is compromised [4]. Therefore, the CESTAC method requires, during the execution of the user code, a dynamical control of multiplications and divisions, which is a so-called *self-validation* of the method. In order to estimate the accuracy of such operands, the method is implemented in a synchronous way: the N values R_i are computed simultaneously. This self-validation has also led to the concept of *computational zero* [18]. A computed result is a computational zero, denoted by @.0, if $\forall i, R_i = 0$ or $C_R \leq 0$. This means that a computational zero is either a mathematical zero or a number without any significance, *i.e.* numerical noise. Relational operators that take into account rounding errors have been defined as follows and called *discrete stochastic relations* [4]. Let $X = (X_1, ..., X_N)$ and $Y = (Y_1, ..., Y_N)$ be two results

computed using the CESTAC method,

$$X = Y \text{ if and only if } X - Y = @.0;$$
$$X > Y \text{ if and only if } \overline{X} > \overline{Y} \text{ and } X - Y \neq @.0;$$
$$X \geq Y \text{ if and only if } \overline{X} \geq \overline{Y} \text{ or } X - Y = @.0.$$

Discrete Stochastic Arithmetic (DSA) [19,20] is the joint use of the CESTAC method, the concept of computational zero, and the discrete stochastic relations.

2.2 The CADNA library

CADNA[5] [6,15] is a library which implements DSA in C, C++, or Fortran codes. Thus, classic floating-point variables are replaced by the corresponding stochastic variables, which are composed of three floating-point values and an integer to store the accuracy. The library contains the definition of all arithmetic operations and order relations for the stochastic types. The rounding error that affects any stochastic variable can be estimated with the probability of 95%. Only the correct digits of a stochastic variable are printed, or "@.0" for numerical noise. Because all operators are redefined for stochastic variables, CADNA requires only a few modifications in a program: essentially in the declarations of variables and in input/output statements.

During the execution of the user code, CADNA can detect numerical instabilities. Such instabilities can occur for instance if a mathematical function or a branching statement involves numerical noise. A numerical instability is also reported in the case of a cancellation, *i.e.* the subtraction of two very close values which generates a sudden loss of accuracy. When numerical instabilities are detected, dedicated CADNA counters are incremented. At the end of the run, the value of these counters together with appropriate warning messages are printed. Because of the composition of stochastic variables, CADNA requires 4x memory storage compared to the original code. Its cost in execution time depends on the code to control and on the instability detection level chosen by the user. If self-validation of DSA is activated, arithmetic compute- and memory-bound benchmarks described in [6] run about 10 times slower with CADNA.

3 Error induced by perturbed data

In the sequel, we assume to work with a binary floating-point arithmetic adhering to IEEE 754 floating-point standard [14]. The relative rounding error unit is denoted by $\mathbf{u}$. For the IEEE 754 *binary64* format (hereafter referred to as "double-precision"), we have $\mathbf{u} = 2^{-53}$ and for the *binary32* format (hereafter referred to as "single-precision") $\mathbf{u} = 2^{-24}$.

Let us consider a function f and let us denote by x its input data. Let $y = f(x)$. Let us denote by $\widehat{f}$ the numerical evaluation of f on a computer. Usually, $\widehat{f} \neq f$ because of the finite precision of the computer arithmetic. Therefore, the computed result is often not y, but $\widehat{y} = \widehat{f}(x)$. The forward error is the difference between the exact solution y and the computed solution $\widehat{y}$.

[5] http://cadna.lip6.fr

The backward analysis tries to seek for Δx such that $\widehat{y} = f(x + \Delta x)$. The quantity Δx is said to be the backward error associated with $\widehat{y}$. It measures the distance between the problem that is solved and the initial problem.

Let us denote by C the condition number of the problem. It is defined as

$$C := \lim_{\varepsilon \to 0^+} \sup_{|\Delta x| \leq \varepsilon} \left[\frac{|f(x + \Delta x) - f(x)|}{|f(x)|} / \frac{|\Delta x|}{|x|} \right]. \tag{2}$$

If the algorithm is backward-stable (*i.e.* the backward error is of the order of the rounding unit $\mathbf{u}$) then one has the following *rule of thumb* [13],

$$|f(x) - \widehat{f}(x)|/|f(x)| \lesssim C\mathbf{u}. \tag{3}$$

If the input data is perturbed, *i.e.* the input data is not x but $\widehat{x} = x(1 + \delta)$, then, by definition of the condition number (2), one computes $\widehat{f}(\widehat{x})$ with

$$|f(x) - \widehat{f}(\widehat{x})|/|f(x)| \lesssim C(\mathbf{u} + |\delta|). \tag{4}$$

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

4 Combining DSA and standard floating-point arithmetic

In this section we show how numerical validation can be performed if standard floating-point arithmetic is used instead of DSA operations. For the sake of simplicity, in the sequel, we consider a BLAS routine that is executed in a code controlled using DSA. However, the approach described here is the same if several computation routines are used continuously. We assume that the BLAS routine is executed with input data affected by rounding errors and/or by measurement errors. We compare the results provided by the CADNA routine and its classic floating-point version.

Let us denote by D the input data of the BLAS routine. Note that BLAS is divided into three levels: BLAS-1 for scalar-vector and vector-vector operations; BLAS-2 for matrix-vector computations; BLAS-3 for matrix operations. Thus, D can correspond to a few arrays: a few vectors for BLAS-1 routines; a matrix and a few vectors for BLAS-2; a few matrices for BLAS-3. Because the code uses the CADNA library, D is composed of stochastic variables: each array element of D is a triplet. Let us assume that the result of the BLAS routine is an array: a matrix in the case of a matrix multiplication; a vector in the case of a matrix-vector multiplication or scalar-/ and vector-vector operations. We describe, first, the case when a CADNA routine is used, then our approach which consists in replacing a call to the CADNA routine by three calls to a classic BLAS routine:

– On the one hand, the CADNA version of the BLAS routine is executed: every arithmetic operation is performed three times with the random rounding mode. The result R is a stochastic array. The associated execution flow is represented in Figure 1 in the general case when one CADNA routine is used or several CADNA routines are executed continuously.

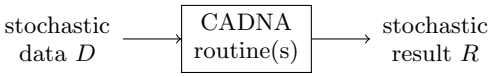


Fig. 1: Execution flow with a call to CADNA routines.

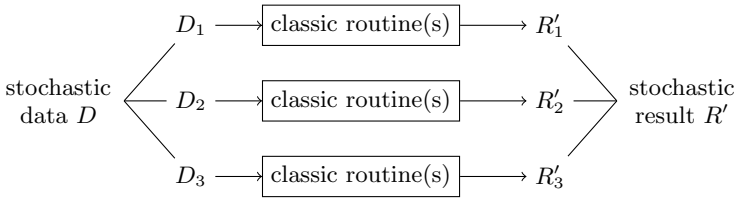


Fig. 2: Execution flow with three calls to classic BLAS routines.

– On the other hand, three input data D_1, D_2, D_3 are created from the triplets of the stochastic data D. Indeed each array element D^i is a triplet (D_1^i, D_2^i, D_3^i). This is followed by three executions of a classic BLAS routine (a user or a vendor implementation) providing three results R'_1, R'_2, R'_3, each of them being a classic floating-point array. A stochastic array R' is created from these three arrays R'_1, R'_2, R'_3: each array element R'^i is a triplet (R'^i_1, R'^i_2, R'^i_3). This array R' can be used in the CADNA routines in the next parts of the code. Figure 2 illustrates the associated execution flow. In this approach, we can clearly benefit from highly-optimized vendor routines to speed up the entire DSA process.

Section 5 compares the accuracy of R and R', while Section 6 presents the performance comparison between the CADNA and pure BLAS routines. The C++ codes developed for our experiments are available[6].

5 Accuracy comparison

5.1 Experimental setup

In this section, we compare the accuracy of results provided by the CADNA version and the classic version of BLAS routines for matrix (xGEMM) and matrix-vector (xGEMV) multiplications. The numerical experiments have been carried out with stochastic data randomly generated between -10^E and $+10^E$, where the choice of E depends on the test case. In the experiments described in Sections 5.2 and 5.3, E is set to 20 (resp. 3) if computations are carried out in double-precision (resp. single-precision). Each stochastic value $x = \{x_i\}$ $(i = 1, 2, 3)$ is initialized as $x_1 = x_2 = x_3 = \alpha 10^e$, where α is a random variable uniformly distributed between -1 and 1 and e is an integer randomly generated in $\{0, ..., E\}$. This initialization ensures the generation of data with different orders of magnitude.

We assume that the input data is affected by measurement errors and/or rounding errors due to previous computation. To simulate such errors, we use

[6] http://www.lip6.fr/Fabienne.Jezequel/ARTICLES/CODES_NSV2020.tar.gz

Table 1: Accuracy comparison of matrix multiplication.

(a) Double-precision

δ	accuracy of R		accuracy difference between R and R'	
	mean	min-max	mean	max
1.e-14	13.9	9-15	2.5e-2	2
1.e-13	12.8	8-15	5.8e-3	1
1.e-12	11.9	7-14	4.2e-4	1
1.e-11	10.9	6-13	2.4e-5	1

(b) Single-precision

δ	accuracy of R		accuracy difference between R and R'	
	mean	min-max	mean	max
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

the *data_st* CADNA function that enables one to perturb a stochastic value with a relative or an absolute error set by the user. In this paper, according to the chosen error δ, each stochastic value $x = \{x_i\}$ $(i = 1, 2, 3)$ is perturbed following this equation:

$$\widehat{x_i} = x_i(1 + \beta_i \delta) \text{ for } i = 1, 2 \quad \text{and} \quad \widehat{x_3} = x_3, \tag{5}$$

where β_i is a random variable uniformly distributed between -1 and 1.

5.2 Matrix multiplication

We present here results obtained with the multiplication of square matrices of size $n = 500$. In accordance with Section 4, we denote by R the stochastic matrix computed with the CADNA routine and by R' the stochastic matrix built from the three matrices computed with the classic floating-point routine. In both cases, the accuracy of each element of the resulting matrix is estimated by CADNA according to Eq. 1. For $i = 1, \ldots, n^2$, CADNA computes the number of correct digits C_{R^i} (resp. $C_{R'^i}$) in the element R^i (resp. R'^i) of the array R (resp. R'). As a remark, superscripts are used here to avoid confusion with the triplet composing a stochastic variable. Tables 1 reports:

- the relative data perturbation δ;
- the mean value, the minimum and the maximum of the accuracy of the n^2 results that are computed by the CADNA routine;
- the mean value and the maximum of the difference between the accuracy of the results obtained with the CADNA routine and the accuracy of those computed with the classic routine. For $i = 1, \ldots, n^2$, this difference Δ^i is evaluated as $\Delta^i = |C_{R^i} - C_{R'^i}|$.

It has been observed that the minimum and the maximum accuracy are the same with both approaches. Table 1 (a) presents results computed in double-precision with data randomly generated between -10^{20} and 10^{20}, while Table 1 (b) shows results computed in single-precision with data randomly generated between -10^3 and 10^3. We recall that the best possible accuracy is 15 digits in double-precision and 7 digits in single-precision.

As the order of magnitude of the perturbation δ increases, the mean accuracy decreases by 1 digit. A low difference can be observed between the accuracy of

Table 2: Accuracy comparison of matrix-vector multiplication.

(a) Double-precision

δ	accuracy of R		accuracy difference between R and R'	
	mean	min-max	mean	max
1.e-14	13.9	12-15	4.6e-2	1
1.e-13	12.7	11-14	7.0e-3	1
1.e-12	11.8	10-13	0	0
1.e-11	10.9	9-12	0	0

(b) Single-precision

δ	accuracy of R		accuracy difference between R and R'	
	mean	min-max	mean	max
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

the results obtained with the CADNA routine and the accuracy of those computed with the classic routine. The order of magnitude of the mean value of this difference decreases as the order of magnitude of δ increases. If the perturbation is sufficiently high (10^{-13} in double-precision, 10^{-4} in single-precision), maximum accuracy difference is 1 digit. As expected, a high perturbation in single-precision induces a low accuracy on the results: if $\delta = 10^{-3}$ the mean accuracy is less than 3 digits.

5.3 Matrix-vector multiplication

The same accuracy comparison as in Section 5.2 is performed for the multiplication of a square matrix of size 1000 with a vector. On the one hand, the matrix-vector multiplication is performed with a CADNA routine. On the other hand, it is performed three times with a classic routine. Like in Section 5.2, the minimum and the maximum accuracy are the same with both approaches. Table 2 (a) presents results computed in double-precision with data randomly generated between -10^{20} and 10^{20}, while Table 2 (b) represents results computed in single-precision with data randomly generated between -10^{3} and 10^{3}.

Like in Section 5.2, as the order of magnitude of δ increases, the mean accuracy decreases by 1 digit. The mean value of the accuracy difference remains low and it decreases as the pertubation δ increases: in double-precision, all the results have the same accuracy if δ is greater than or equal to 10^{-12}. Like in Section 5.2, in single precision a high pertubation results in a poor accuracy.

6 Performance comparison

6.1 Experimental setup

The run times of the execution flows described in Figures 1 and 2 are compared for different matrix sizes. Section 6.2 presents the performance obtained for matrix multiplication and Section 6.3 for matrix-vector multiplication. In both sections, we compare the execution time of a code using CADNA with that of various implementations of our proposed approach: they perform three matrix or matrix-vector multiplications and array copies. We analyze the performance of:

– "CADNA": a sequential code that performs with CADNA a naive matrix or matrix-vector multiplication;

– "naive seq": a sequential code that implements our approach (described in Figure 2) using the same naive algorithm with classic floating-point arithmetic;
– "naive OMP": an implementation of our proposed approach that relies on a naive algorithm with classic floating-point arithmetic parallelized using OpenMP;
– "MKL seq": an implementation of our proposed approach using a sequential MKL BLAS routine;
– "MKL OMP": an implementation of our proposed approach that relies on a parallel MKL BLAS routine, which underneath uses OpenMP.

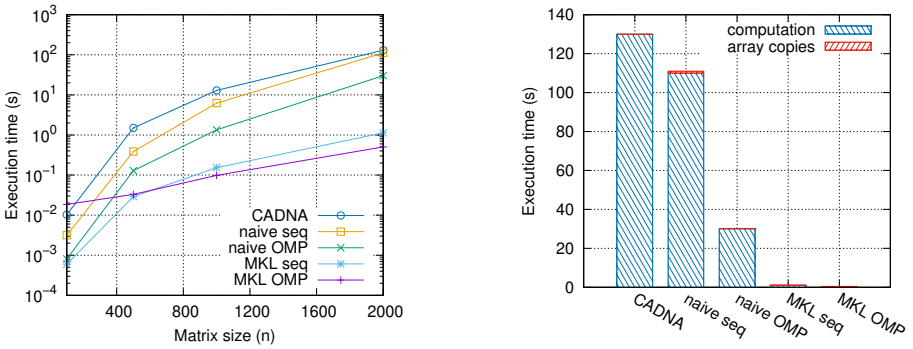
By "naive algorithm", we mean a non-optimized algorithm based on nested loops. In naive OMP and MKL OMP, the array copies are also parallelized using OpenMP, but they are not parallelized in the other cases. In this section, we present the execution times of double-precision codes. As a remark, the same trends are observed in single-precision. However, as mentioned in Section 5, single-precision may not be suitable for computation with perturbed data because they induce low accuracy results.

Except with the CADNA routine, array copies are required to split the stochastic data into three classic floating-point data, and then to merge the three classic floating-point resulting arrays into a stochastic array, as shown in Figure 2. In our evaluations, to point out the array copy cost w.r.t. the BLAS routine execution time, those array copies are performed before and after the BLAS routine execution. This is the worst case that maximizes the array copy cost in the total execution time. However, if standard floating-point operations (or classic BLAS routines) are continuously used, those array copies are required only before and after them. As the stochastic type in double-precision is a structure consisting of three 64-bit floating-point values and two 32-bit integer values (one for storing the accuracy and one for memory alignment), it has 4 times memory footprint than the standard 64-bit floating-point operation. The array copy corresponds to the conversion between array-of-structures (the stochastic type) and structure-of-arrays (three standard floating-point values).

The evaluation environment is as follows. The platform for performance measurements is an Intel Core i7-8650U processor clocked at 1.9 GHz with 4 cores, 8 MB cache. The operating system is Linux Ubuntu 18.04.4. The codes are compiled with gcc version 8.3.0 and optimized with the `-O3` flag. We use the `-frounding-math` option to disable transformations and optimizations that assume default floating-point rounding behavior. The MKL version is 2019.5.281. The CADNA version is cadnac-3.1.5 and for performance evaluation, instability detection is deactivated. In parallel codes that rely on OpenMP, the number of threads is set to 4.

6.2 Matrix multiplication (compute-bound)

Figure 3 presents the execution time of the different implementations as previously described. A log scale is used for the y-axis to improve the readability of the obtained results. Figure 3 also shows the time spent in matrix multiplications and in array copies if the matrix size is 2000. We can observe that despite the memory copies the codes using three classic matrix multiplications perform



(a) W.r.t. problem size

(b) Histogram at n=2000

Fig. 3: Execution time including matrix multiplications and array copies.

better than the CADNA routine. Most of their total execution time is spent in matrix multiplications.

The performance ratio between the CADNA routine and the sequential code that performs three naive matrix multiplications ("naive seq") decreases from 4 to 1.2 as the matrix size increases from 100 to 2000. The performance ratio can be explained by the following two differences. First, the CADNA routine performs the random rounding at each operation. Second, the CADNA routine accesses the matrix values from a structure, that is indirect memory access.

As expected, the OpenMP naive matrix multiplication provides better performance than the sequential one. The MKL sequential matrix multiplication performs even better. From a certain matrix size, the MKL parallel implementation using OpenMP outperforms all the other codes. In particular, if the matrix size is 2000, we can notice a performance ratio of 294 between the CADNA routine and the MKL OMP code that performs three matrix multiplications using OpenMP and array copies. Table 3 summarizes the speedup against CADNA with our proposed method with MKL OMP. Herein, we also show another result with n=5000 on 24 cores with dual-socket Intel Xeon Gold 6126 (2.6 GHz with 12 cores) with the MKL library 2019.1.144 (the codes were compiled with Intel icpc 19.0.1.144). This table shows that the performance gain increases on large scale and that the array copy cost becomes visible against the computation cost, in particular on many-cores that can efficiently perform the computation.

6.3 Matrix-vector multiplication (memory-bound)

Figure 4 shows the execution time of the different implementations as previously described. It can be observed that the matrix-vector multiplication with CADNA performs better than the sequential codes which execute three floating-point matrix-vector multiplications and array copies. In the sequential codes that use classic floating-point arithmetic the main part of the execution time is spent in array copies. If we consider computation time only, the performance ratio between the CADNA routine and the code that performs three sequential matrix-vector multiplications decreases from 4 to 3 as the matrix size increases from

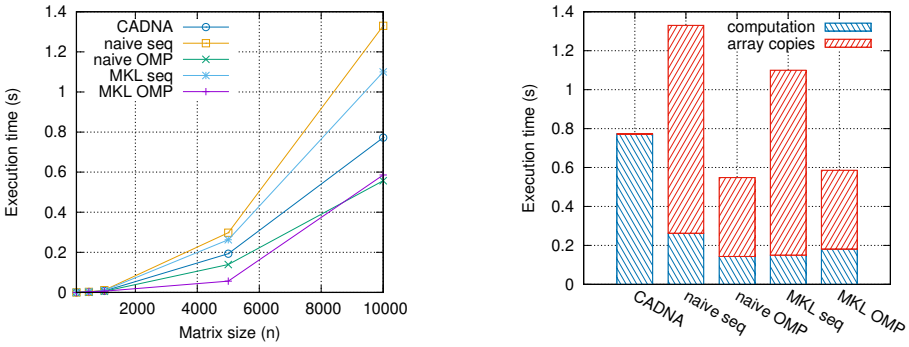
Table 3: Execution time (in seconds) of CADNA and the proposed method with MKL OMP on a matrix multiplication.

(a) 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

(b) 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



(a) W.r.t. problem size (b) Histogram at n=10000

Fig. 4: Execution time including matrix-vector multiplications and array copies.

100 to 10000. Like for matrix multiplication, accessing values from a structure is more costful, especially for small problems. From a certain matrix size, thanks to the parallel array copies, the OpenMP codes that use classic floating-point arithmetic perform better than the CADNA code.

As a remark, we note that the array copy cost shown in this example is the worst case, which is the situation when classic GEMV routines are used only once and the ratio between the memory reference in one array copy and that in the routine is approximately 1. In this case, three matrix copies are performed before and after the classic GEMV executions. If standard floating-point operations (or classic BLAS routines) are continuously used, the array copy cost arises only before and after them.

7 Discussion: pros and cons of the proposed approach

7.1 Pros: performance and applicability

The proposed approach includes two performance advantages and one regarding the applicability as follows.

– The performance gain by avoiding DSA operations. On compute-bound operations, the overhead caused by random rounding may be eliminated. On memory-bound operations, in the cases we can ignore the array copy cost

(*i.e.* when we use classic routines continuously), the overhead of CADNA is reduced by a factor of 3.
- The possible performance gain by directly relying on existing optimized codes. The demonstration on matrix-multiplication in Section 6.2 is a typical example that gains high-performance from a vendor optimized library.
- Avoiding the translation of an existing code to the CADNA version for numerical validation. Previously, when the target code included some portions where CADNA could not be applied (for example, relying on external libraries or including some intrinsic instructions for optimization), we needed to prepare an alternative code.

Those advantages can be more pronounced on many-core processors. While CADNA supports OpenMP, the performance overhead can be larger than single-thread due to the existence of some private sections [8]. CADNA for CUDA is also available, but it is observed that the overhead (especially on compute-bound operations) becomes higher than that on CPUs [7].

7.2 Cons: instability detection and accuracy

Instability detection With perturbed data, if a CADNA routine is replaced by three calls to a classic routine, the result accuracy can still be correctly estimated by CADNA. In particular, the user can be informed that the result is numerical noise (*i.e.* no correct digit). However, numerical instabilities are not detected if a classic floating-point routine is used instead of a CADNA routine.

As an example, we consider the multiplication in double-precision of two matrices A and B of size 10. The first row of A is $[1, 1, 1, 1, 1, -1, -1, -1, -1, -1]$: its first half is set to 1, while the second half is -1. The other lines of A are randomly generated between -10^{20} and 10^{20}, as described in Section 5.1. All the elements of the matrix B are set to 1. Then A and B are perturbed with a relative error $\delta = 10^{-12}$. We denote by C the matrix product obtained using CADNA and by C' the matrix product computed by three calls to a classic routine according to the execution flow described in Figure 2. In both resulting matrices, each element of the first line is numerical noise and can be displayed by CADNA as "@.0". Let us focus on the first element of C and C'. The associated triplets are:

$$C_{1,1} = \{\text{-1.1590728377086634e-13}, \text{+4.9227288911879442e-13}, \text{+0.0000000000000000e+00}\},$$
$$C'_{1,1} = \{\text{-1.1790568521589528e-13}, \text{+4.9327208984079348e-13}, \text{+0.0000000000000000e+00}\}.$$

Each triplet is composed of values that have no common digit. The differences between $C_{1,1}$ and $C'_{1,1}$ are due to the random rounding mode of CADNA. One advantage of the CADNA routine is the instability detection. Here, the user is informed that 10 catastrophic cancellations occurred (*i.e.* subtractions of close values affected by rounding errors).

Accuracy improvement CADNA may improve the accuracy of results thanks to the detection of numerical noise. Such improvement is not possible if a CADNA

routine is replaced by three calls to a classic routine. As an example, let us consider the linear system $Ax = b$ with

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

Its exact solution is $x_{exact} = \left(1, 1, 10^{-8}, 1\right)^T$.

A and b are perturbed with a relative error $\delta = 10^{-6}$. Then the linear system is solved using Gaussian elimination with partial pivoting. The result x obtained using CADNA and the result x' computed by three calls to a classic routine are displayed by CADNA as:

$$x = \begin{pmatrix} 0.100\text{E+}001 \\ 0.999\text{E+}000 \\ 0.999999\text{E-}008 \\ 0.999999\text{E+}000 \end{pmatrix} \text{ and } x' = \begin{pmatrix} @.0 \\ @.0 \\ @.0 \\ 0.999999\text{E+}000 \end{pmatrix}.$$

The numerical quality of the first three elements of x is rather satisfactory, whereas these elements are numerical noise in x'. Pivoting implies to choose at several steps of Gaussian elimination a suitable value for the pivot that will be used in subsequent computation. Among several matrix elements, the one having the greatest absolute value is selected with a test such as: `if` $(|A_{i,j}| > p_{max})$. During the Gaussian elimination, a matrix element is numerical noise: it has no correct digits. With CADNA this non-significant element in A is not chosen as a pivot. Without CADNA, this element cannot be detected as non-significant. As its absolute value is high, it is chosen as a pivot and numerical noise is propagated in subsequent computation. With CADNA, the following list of three numerical instabilities is provided:

```
1 UNSTABLE BRANCHING(S)
1 UNSTABLE INTRINSIC FUNCTION(S)
1 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S)
```

A catastrophic cancellation occurred and generated a non-significant element used as an argument in the absolute value function. This numerical noise also caused an unstable test for the pivot selection.

8 Conclusion

This paper proposed an alternative approach for numerical validation on perturbed data, which can rely on the standard floating-point arithmetic instead of DSA operations. If the input data includes a perturbation with an order of magnitude greater than the relative rounding error unit, we can replace the DSA operations, which execute each floating-point operation three times with random rounding, by three executions with the standard floating-point arithmetic. It brings almost no accuracy difference in the results. This proposed approach contributes a significant performance improvement, in particular on compute-bound operations on many-core processors as it can directly rely on existing user-implemented codes or even highly-optimized vendor libraries. On

the other hand, we lose the instability detection and the possibility of accuracy improvement, which were available on the DSA implementation CADNA. The same conclusions would be valid with a parallel code using MPI for communication. CADNA enables one to control the numerical quality of HPC codes that rely on MPI [17]. If in such a code computation-intensive routines are executed with perturbed data, the CADNA-MPI routines can be replaced by optimized floating-point MPI routines. Because the experiments presented in this article use synthetic input data, as a perspective we plan to apply our approach to real-life examples with realistic data sets.

Acknowledgements

This research was partially supported by the European Union's Horizon 2020 research, innovation programme under the Marie Skłodowska-Curie grant agreement via the Robust project No. 842528 and the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant No. 19K20286.

References

1. Alefeld, G., Herzberger, J.: Introduction to interval analysis. Academic Press (1983)
2. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide. SIAM, Philadelphia, PA, third edn. (1999)
3. Carson, E., Higham, N.J.: Accelerating the solution of linear systems by iterative refinement in three precisions. SIAM J. Sci. Comput. **40**(2), A817–A847 (2018)
4. Chesneaux, J.M., Vignes, J.: Les fondements de l'arithmétique stochastique. Comptes Rendus de l'Académie des Sciences - Series I - Mathematics **315**, 1435–1440 (1992)
5. Denis, C., de Oliveira Castro, P., Petit, E.: Verificarlo: checking floating point accuracy through Monte Carlo Arithmetic. In: ARITH'23, Silicon Valley, USA (Jul 2016)
6. Eberhart, P., Brajard, J., Fortin, P., Jézéquel, F.: High performance numerical validation using stochastic arithmetic. Reliable Computing **21**, 35–52 (2015)
7. Eberhart, P., Landreau, B., Brajard, J., Fortin, P., Jézéquel, F.: Improving CADNA Performance on GPUs. In: IPDPSW. pp. 1016–1025. Vancouver, Canada (2018)
8. Eberhart, P., Brajard, J., Fortin, P., Jézéquel, F.: Estimation of Round-off Errors in OpenMP Codes. In: IWOMP 2016. LNCS, vol. 9903, pp. 3–16. Springer (2016)
9. Frechtling, M., Leong, P.H.W.: MCALIB: Measuring Sensitivity to Rounding Error with Monte Carlo Programming. ACM TOPLAS **37**(2), 1–25 (2015)
10. Févotte, F., Lathuilière, B.: Debugging and optimization of HPC programs in mixed precision with the Verrou tool. In: CRE at SC18, Dallas, USA (2018)
11. Graillat, S., Jézéquel, F., Wang, S., Zhu, Y.: Stochastic arithmetic in multiprecision. Mathematics in Computer Science **5**(4), 359–375 (2011)
12. Haidar, A., Abdelfattah, A., Zounon, M., Wu, P., Pranesh, S., Tomov, S., Dongarra, J.: The design of fast and energy-efficient linear solvers: On the potential of half-precision arithmetic and iterative refinement techniques. In: International Conference on Computational Science (ICCS 2018). vol. 10860, p. 586–600. Springer, Wuxi, China (Jun 2018)
13. Higham, N.: Accuracy and stability of numerical algorithms, 2nd ed. SIAM (2002)
14. IEEE Computer Society: IEEE Standard for Floating-Point Arithmetic. IEEE Standard 754-2008 (Aug 2008)

15. Jézéquel, F., Chesneaux, J.M.: CADNA: a library for estimating round-off error propagation. Elsevier CPC **178**(12), 933–955 (2008)
16. Kulisch, U.: Advanced Arithmetic for the Digital Computer. Springer, Wien (2002)
17. Montan, S., Denis, C.: Numerical verification of industrial numerical codes. In: ESAIM: Proc. vol. 35, pp. 107–113 (Mar 2012)
18. Vignes, J.: Zéro mathématique et zéro informatique. Comptes Rendus de l'Académie des Sciences - Series I - Mathematics **303**, 997–1000 (1986)
19. Vignes, J.: A stochastic arithmetic for reliable scientific computation. Mathematics and Computers in Simulation **35**(3), 233–261 (1993)
20. Vignes, J.: Discrete Stochastic Arithmetic for validating results of numerical software. Numerical Algorithms **37**(1–4), 377–390 (Dec 2004)
21. Wilkinson, J.H.: Rounding errors in algebraic processes, vol. 32. HMSO (1963)