HAL
open science

Verificarlo: checking floating point accuracy through Monte Carlo Arithmetic

Christophe Denis, Pablo de Oliveira Castro, Eric Petit

To cite this version:

Christophe Denis, Pablo de Oliveira Castro, Eric Petit. Verificarlo: checking floating point accuracy through Monte Carlo Arithmetic. 2015. hal-01192668v1

HAL Id: hal-01192668

https://hal.science/hal-01192668v1

Preprint submitted on 3 Sep 2015 (v1), last revised 7 Nov 2018 (v4)

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.

Verificarlo: checking floating point accuracy through Monte Carlo Arithmetic

Christophe Denis*, Pablo de Oliveira Castro†, Eric Petit†
*CMLA, ENS Cachan
Email: christophe.denis@cmla.ens-cachan.fr
†Université de Versailles Saint-Quentin-en-Yvelines
Email: {pablo.oliveira, eric.petit}@uvsq.fr
*†Université Paris-Saclay

***Abstract*—Numerical accuracy of floating point computation is a well studied topic, but which has not made its way to the end-user in scientific computing. With the recent requirements for code modernization to exploit new highly parallel hardware and perform higher resolution computation, this has become one of the critical issues to address. To democratize numerical accuracy analysis, it is important to propose tools and methodologies to study large use cases in a reliable and automatic way. In this paper, we propose verificarlo, an extension to the LLVM compiler to automatically use Monte Carlo Arithmetic in a transparent way for the end-user. It supports all the major languages including C, C++ and Fortran. We also illustrate the fact that unlike source-to-source approaches, our implementation captures the influence of compiler optimizations on the numerical accuracy. Finally, we illustrate on various use cases how Monte Carlo Arithmetic using the verificarlo tool outperforms the existing approaches and is a step toward automatic numerical analysis of large scientific applications.**

1. Introduction

This paper presents a new compiler tool to assess the uncertainties on a scientific code due to the floating point (FP) arithmetic. It builds upon the extensive work of Parker [Par97] and Frechtling [FL15] on Monte Carlo Arithmetic (MCA) for floating point accuracy verification. Floating point computations are a model of real number computation where a real number is rounded towards a floating point number, and some arithmetical properties, such as the associativity of the sum, are lost. Consequently, the computer numerical results are sensitive to the evaluation order of the floating point arithmetical operations, the floating point precision, and the rounding mode.

The quantification of the floating point uncertainties is important. In the next decade, exascale supercomputers will provide the computational power required to perform very large scale simulations. For certain applications the results of exascale simulations will be of such high resolution that experimental measurements will be insufficient for validation purposes. As floating point approximations of numeric expressions are neither associative nor distributive, the results of a numerical simulation can differ between executions. As reported by Duff [Duf11], "*Getting different results for different runs of the same computation can be disconcerting for users even if, in a sense, both results are correct*", there is a need to have an automatic and global approach giving a confidence interval on the results taking into account the floating point arithmetic effect.

Numerical verification is a procedure to estimate the effect of the floating point model on the accuracy of the computed results. It is the first step of a rigorous Verification and Validation (V&V) procedure. Several methods exist to perform a numerical verification on a numerical code. Kahan, the primary architect of the IEEE-754 standard for floating point computation, argues in [Kah06] that using extendable precision interval arithmetic is almost foolproof. The interval arithmetic is an arithmetic defined on sets of guaranteed intervals rather than on sets of IEEE-754 numbers. The numerical verification on a scientific code using IEEE-754 floating point numbers consists of comparing the results with those obtained on a shadow code using interval arithmetic. It requires that the results intervals are sufficiently small. If not, the computation needs to be performed again by extending the precision of the interval arithmetic. Unfortunately, even if the result is guaranteed, the interval arithmetic typically produces overly pessimistic bounds as it does not take into account the round-off error compensation when using the rounding mode to the nearest. Some numerical algorithms need to be also modified when using interval arithmetic. For example, the Newton-Raphson method needs to be modified in order to obtain convergence under interval arithmetic [Rev03]. Consequently, from an industrial point of view, it is only possible to use extendable precision interval arithmetic on specific numerical algorithms and not on a whole scientific code.

An alternative is to obtain confidence interval on the results by applying random perturbation on the numerical code. For example, the Discrete Stochastic Arithmetic implemented in the CADNA library perturbates the computing by randomly changing the rounding mode. Discrete Stochastic Arithmetic is based on CESTAC developed by Vignes in 1974. It is a powerful numerical debugger tool which has been used to solve real problems. Nevertheless, both for methodological and technical aspects, it could not been used to automatically perform numerical verification on a whole

scientific code unlike the Monte Carlo Arithmetic (MCA) as discussed in this paper.

We propose the following contributions:

- Verificarlo, a new LLVM compiler tool to automatically use the Monte Carlo arithmetic in place of the IEEE-754 FP.
- A set of experiments to validate the automatic MCA approach using verificarlo and compare it to the state-of-the-art MCA approach using CIL [FL15] and CESTAC approach using CADNA [Vig04].

This paper is organized as follows. Section 2 presents stochastic arithmetic for numerical verification of application. Section 3 introduces the verificarlo tool and discusses the pros and cons and compares to other approaches. Finally we propose in section 4 a set of experiments to validate our tool and demonstrate its capabilities.

2. Probabilistic methods to check the floating point accuracy

The aim of this section is to briefly present two probabilistic methods used to check the floating point accuracy: the Monte Carlo Arithmetic (MCA) and the Discrete Stochastic Arithmetic (DSA).

2.1. Monte Carlo Arithmetic (MCA)

Monte Carlo Arithmetic (MCA) tracks rounding and catastrophic cancellation errors at a given virtual precision t by applying randomization to input and output operands. MCA makes no assumption about the round-off error distribution and produces unbiased random round-off errors.

It forces the results of floating point operations to behave like random variables. This turns executions into trials of a Monte Carlo simulation allowing statistics on the effects of rounding error to be obtained over a number of executions. This section succintly summarizes MCA, for a full presentation see [Par97].

To model errors on a FP value x at virtual precision t, Parker proposes the following function:

$$inexact(x) = x + 2^{e_x - t}\xi, \tag{1}$$

where e_x is the exponent of the FP value x and ξ is a uniformly distributed random variable in the range $[-\frac{1}{2}, \frac{1}{2}]$.

Each floating point operation $x \circ y$ is transformed into a MCA FP operation using one of the following modes:

- RR: Random Rounding, which tracks rounding errors by introducing an error in the outbound value,

$$x \circ y \rightarrow round(inexact(x \circ y))$$

- PB: Precision Bounding, which tracks catastrophic cancellations by introducing errors in the inbound values,

$$x \circ y \rightarrow round(inexact(x) \circ inexact(y))$$

- Full MCA: Monte Carlo Arithmetic with inbound and outbound errors,

$$x \circ y \rightarrow round(inexact(inexact(x) \circ inexact(y)))$$

When the exact solution x of a problem is known, we can measure the number of significant digits s in base β by computing the magnitude of the relative error between the approximated value $\hat{x}$ and the exact value x using the following formula

$$s = -\log_\beta \left| \frac{\hat{x} - x}{x} \right| \tag{2}$$

Parker extends this definition to MCA and shows [Par97, p. 23] that the total significant digits for a set of MCA results at virtual precision t is given by the magnitude of the relative standard deviation

$$s' = -\log_\beta \frac{\sigma}{\mu} \tag{3}$$

In this formula, μ and σ are the mean and the standard deviation of the result distribution. Unfortunately, the exact distribution of results is unknown, but it can be empirically estimated by using a large number of Monte Carlo trials. Indeed for a large number of trials, $s' \approx -\log_\beta \frac{\hat{\sigma}}{\hat{\mu}}$, where $\hat{\mu}$ and $\hat{\sigma}$ are the sample mean and sample standard deviation.

The metrics given by equations 2 and 3 will be used in section 4 to evaluate our outputs and compare to other approaches.

2.2. Discrete Stochastic Arithmetic (DSA)

Discrete Stochastic Arithmetic (DSA) is based on the CESTAC method. The CESTAC method is a pioneer work in the domain of the random computer arithmetic [VLP74]. The ingenious idea is to randomly change the rounding mode of a floating point (FP) computation to estimate its accuracy. For debugging purposes, DSA has made the choice to carry out a single program in which each FP operation is performed N times with a rounding mode towards plus or minus infinity. For each sample, the rounding mode is randomly chosen. There is thus a probability $P_N = 2^{1-N}$ that all the N samples compute a FP operation with the same rounding mode. The number of significant digits is computed after each FP computation by using the Student distribution. DSA also redefines relational operators. A full review of DSA is provided in [Vig04].

The CADNA library is an implementation of Discrete Stochastic Arithmetic (DSA) with $N = 3$ samples. The first two samples compute each FP operation with a random rounding mode whereas the last one uses the rounding mode not used by the second sample.

The validation of CESTAC method and DSA is based on a probabilistic first order model. It has been established by considering that elementary round-off errors of the FP arithmetic operations are random independent, centered and uniformly distributed variables. Kahan has formulated strong objections to this assumption by proposing in [Kah96] the following case study:

$$Kh_x(dx) = cf(x + dx) - rp(x) \quad (4)$$

$$cf(x) = 4 - \frac{3(x-2.0) * ((x-5.0)^2 + 4)}{x + (x-2.0)^2((x-5.0)^2 + 3.0)} \quad (5)$$

$$rp(x) = \frac{622.0 - x(751.0 - x(324 - x(59.0 - 4.0x)))}{112 - x(151.0 - x(72.0 - x(14.0 - x)))} \quad (6)$$

with :

- $x = 1.60631924$
- $dx = i\epsilon$, i varying from 1 to 300 with step $1, \epsilon = 2^{-53}$

Figure 1 presents the evaluation of equation 4 by using IEEE-754 double precision floating point numbers. On one hand, the stripped patterns in figures 1 and 2 show that IEEE-754 and CADNA results are not uniformly distributed random variables. On the other hand, the Monte Carlo Arithmetic introduced in section 2.1 permits to obtain an independent and identically distributed (iid) centered uniform sample as shown in figure 3.

Chesneaux and Vignes argue in [CV88] that even if the independent, centered, and uniform assumption is not satisfied, the CESTAC method is able to correctly estimate the number of significant digits with a probability of 95%. Nevertheless, Chatelin [Cha88] indicates that CESTAC's *confidence cannot be greater than 5% under various conditions which are shown to be often met in practice* and Parker [Par97] explains how MCA can overcome these limitations.

It is important to notice that DSA and MCA also differ from a methodological point of view. DSA uses a synchronous approach where the user incrementally fixes the numerical instabilities reported by CADNA. DSA is based on a first order model: some operations such as unstable division or unstable multiplication may invalidate the model. The user must correct these unstable FP operations before CADNA can estimate accurately the number of significant digits. Therefore DSA is well suited to perform numerical debugging to correct numerical instabilities as done in [SJDC07].

3. Verificarlo: A software for automatic Monte Carlo Arithmetic analysis

As previously discussed in section 2.1, Monte Carlo Arithmetic is a powerful framework to understand the numerical stability of a function or program. To encourage its wide adoption by the community we have developed verificarlo, a tool for automatic MCA analysis of C, C++ and Fortran programs. Verificarlo builds upon the LLVM Compiler [LA04] project and the MCALIB [FL15]. It takes as input a source code project and compiles it with a special instrumentation pass that replaces all floating point operations by their MCA counterpart in MCALIB. The instrumentation can be applied to the whole program or only to a function of interest.

Two previous approaches for automatic MCA simulation have been proposed. Yeung et al. [YYL11] implement MCA at the hardware level through specialized FPGA coprocessors. While providing low overhead, this approach is impractical because it requires specialised hardware not available to the practitioner.

Frechtling et al. [FL15] leverage source-to-source rewriting of floating point operations through the CIL tool for program transformation [NMRW02]. The first drawback of using CIL is that analysis is limited to C programs. The second and main drawback with source-to-source rewriting is that the instrumentation happens before and may hinder the compiler optimizations. That means that the floating point operations in the MCA binary and in the original binary may be different. What is tested is not always what will be executed because CIL cannot capture the effect of compiler optimizations on numerical errors.

To tackle these issues, verificarlo instruments the floating point at the optimized Intermediate Representation level (IR). First, because the IR representation is independent of the source language used, verificarlo can operate on any source language supported by LLVM that includes C and C++ through clang and Fortran through dragonegg. Second, the instrumentation pass is done after all the other front-end and middle-end optimization passes (which include all the floating point optimizations such as `-ffast-math` or `-freciprocal-math`).

Verificarlo is available at http://www.github.com/verificarlo/verificarlo under an open source licence. Verificarlo computes Monte Carlo arithmetic using a modified version of MCALIB. One notable difference is that our version of MCALIB replaces the standard libc pseudo-random generator with Mersenne Twister [MN98b]. This provides two benefits: first for user programs using the libc `rand` function, having a separate generator avoids seeding collisions. Second, Mersenne Twister is a robust random number generation in the context of Monte Carlo simulations [MN98a].

One disadvantage of MCA is that it requires a large number of samples compared to DSA and is therefore more costly. Table 1 compares the cost of running a numerical analysis with CADNA, MCALIB and verificarlo. Verificarlo and MCALIB are significantly slower than CADNA. The first reason is that to be accurate they require a higher number of samples. The second reason is that both MCALIB and verificarlo use the MPFR [FHL+07] library to compute MCA samples. Performing high precision computations with MPFR is more costly than changing the rounding mode.

Fortunately, verificarlo supports massively parallel execution out of the box. The high overhead can be mitigated by concurrently measuring the MCA samples. Our tests show an ideal scalability thanks to the embarrassingly parallel nature of Monte Carlo.

4. Experimental results

This section presents four case studies to illustrate floating point accuracy verification using verificarlo and its ben-

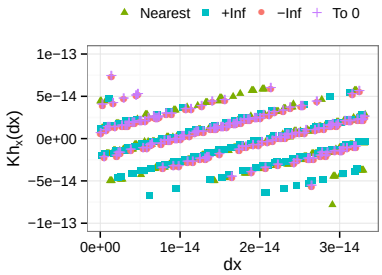

Figure 1: Eq. 4 using IEEE-754 DP

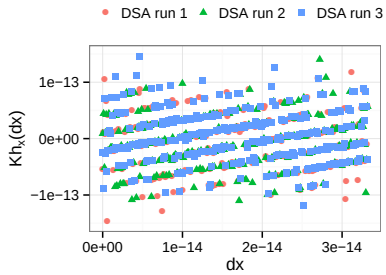

Figure 2: Eq. 4 using CADNA

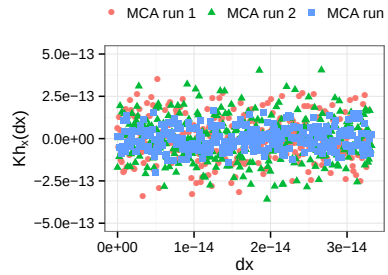

Figure 3: Eq. 4 using verificarlo

version	samples	total time (s)	time/sample (s)
original program	1	.056	.056
CADNA	3	2.93	.097
MCALIB	128	1184.02	9.25
verificarlo	128	834.57	6.52
verificarlo 16 threads	128	54.39	.42

TABLE 1: Performance in seconds for the numerical analysis of the compensated sum algorithm detailed in section 4.1 on an array of size 1000000. All the binaries were compiled using `-O0`. The experiment was performed on a 16-core 2-socket Xeon E5@2.70GHz with 20Mb L3 cache per socket and 64Gb of RAM.

efit compare to state-of-the-art approaches.

The first case study evaluates the numerical error in a compensated sum algorithm using CADNA, CIL+MCALIB, and verificarlo. Among the three tools, only verificarlo is able to capture the effect of compiler flags on numerical errors.

The second case study deals with the solving of a linear system $Ax = b$ proposed by Kahan in [Kah66]. The matrix A is ill-conditioned which can reduce the number of significant digits. This case study demonstrates how verificarlo using MCA can estimate the number of significant digits. The resolution is done by using the sophisticated LAPACK routines. It has not been possible to use CADNA as it requires to manually change the source code of the LAPACK library. Verificarlo gives an estimation of the number of significant digits close to the number of significant digits between the IEEE-754 computing and the exact solution.

The third case study deals with unstable branching. In this case, CADNA is too pessimistic as it estimates that the numerical result has no accurate digits whereas verificarlo finds a number of significant digits close to the number of accurate significant digits between the IEEE-754 computing and the exact value.

The fourth case study deals with the management of a counter. The comparison between the IEEE-754 double precision computing and the exact solution shows that the numerical result is a numerical noise having no significant digits. Unfortunately for this case, CADNA is too optimistic as it over-estimates the result significant digits whereas verificarlo succeeds to estimate that the IEEE-754 DP result

```
1  float sum = f[0];
2  float c = 0.0, y, t;
3
4  for (int i=1;i<n;i++) {
5      y = f[i] - c;
6      t = sum + y;
7      c = (t - sum) - y;
8      sum = t;
9  }
10
11 return sum;
```

Figure 4: *Kahan compensated summation:* with `-O3 -ffast-math` flags the compiler simplifies and removes the computation of the compensation term `c`.

is a numerical noise.

4.1. Case study 1: Compensated Summation

In the following, we demonstrate the importance of capturing compiler effects on a standard use case: Kahan's compensated summation algorithm [Hig02, p. 83] shown on figure 4. The C implementation is particularly sensible to compiler optimizations when floating point associativity rules are relaxed with `-ffast-math -O3`. The compiler uses simple common subexpression elimination and rewrites line 9 as `sum = sum + f[i]` which is the naive non-compensated summation.

Using verificarlo and CIL+MCALIB [FL15] we measured 1000 sample executions of the Kahan summation code compiled with `-O3 -ffastmath` and `-O0`. The input array contains 1000 random single precision floats in the interval $[0, 1]$ and therefore has a condition number of 1. Only Random Rounding MCA errors were considered in this study.

Figure 5 compares the results between verificarlo and CIL+MCALIB. On one hand, CIL+MCALIB is unable to detect any difference between the two versions. Table 2 shows the number of significant digits predicted by CADNA, CIL+MCALIB and verificarlo for an array of 100000 floats. Again, CADNA and CIL+MCALIB are blind to compiler optimizations because they operate at source

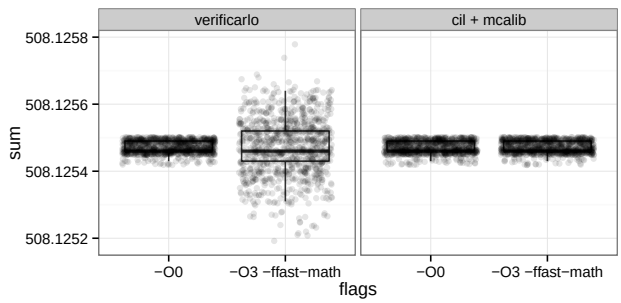

Figure 5: *One thousand MCA RR samples of Kahan summation:* CIL+MCALIB is unable to capture the compiler effect on Kahan's summation because it operates at the source level. On the other hand verificarlo operates after compiler optimizations and correctly shows that the `-O0` version is more precise than the `-O3 -ffast-math` version thanks to the compensation term `c`.

	-O0	-O3 -ffast-math
CADNA	7	7
CIL+MCALIB	7.3	7.3
verificarlo	7.3	**5.8**

TABLE 2: Number of significant digits estimated with size $n = 100000$. Verificarlo is the only tool that detects that `-O3 -ffast-math` introduces a loss of accuracy.

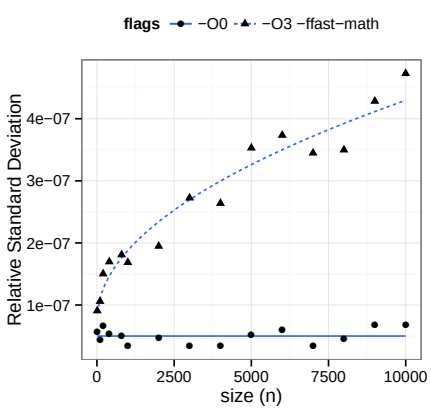

Figure 6: Relative Standard Deviation ($\frac{\sigma}{\mu}$) of Kahan's compensated sum computed on 1000 verificarlo samples. The compensated `-O0` version has a constant error while the `-O3 -fast-math` error increases as $O(\epsilon\sqrt{n})$.

level. On the other hand, verificarlo correctly shows the loss of accuracy in the `-O3 -ffast-math` version.

In figure 6 we plot the relative standard deviation of verificarlo's samples with different input sizes. Theoretical error analysis [Hig02, p. 85] shows that Kahan's compensated sum relative error is bounded by $2\epsilon + O(n\epsilon^2)$ where n is the input size and ϵ the computation's precision. So Kahan's sum relative error is constant for inputs satisfying $n\epsilon < 1$. However the relative error of a naive sum grows as $O(\epsilon\sqrt{n})$ when floating point errors are iid with zero mean. We see that verificarlo stochastic error analysis closely matches the theoretical error bounds.

This experiment demonstrates how the late instrumentation in verificarlo helps evaluating the impact of compiler optimizations on numerical stability.

4.2. Case study 2: Resolution of a linear system

Kahan [Kah66] proposes the following linear system with a large condition number:

$$\begin{pmatrix} 0.2161 & 0.1441 \\ 1.2969 & 0.8648 \end{pmatrix} x = \begin{pmatrix} 0.1440 \\ 0.8642 \end{pmatrix} \tag{7}$$

The exact solution of equation 7 is:

$$x = \begin{pmatrix} 2 \\ -2 \end{pmatrix} \tag{8}$$

In the context of this case study, we solve equation 7 using the LAPACK numerical library. LAPACK is written in Fortran 90 and provides sophisticated routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. Table 3 reports the results of the resolution using the IEEE-754 single precision and double precision arithmetic with a rounding mode to the nearest.

Unfortunately, it has not been possible to use CADNA as it requires to modify the source code which is difficult and costly to do in a whole numerical library such as LAPACK. For example, Montan [MDCL13] has developed a modified version of the LAPACK DGEMM routine (matrix multiplication) to efficiently use CADNA. In contrast, the use of verificarlo has permitted to implement automatically MCA on the whole LAPACK library. The number of samples used by MCA in this experiment is set to 1000. Table 4 reports the results of the resolution by using the MCA single and double precision floating point arithmetic.

In this example, the estimator given in section 2.1 accurately computes the number of significant digits. Moreover, this estimation does not require knowing beforehand the exact solution of the system.

4.3. Case study 3: Unstable branching

This section presents the differences between MCA and DSA when dealing with branches testing a FP value. The following C program is used in this case study:

FP arithmetic	Mean value	Standard deviation	s'
MCA single precision	$\bar{x}(1) = 1.02463705$	$\sigma(x(1)) = 6.46717332$	0.0
	$\bar{x}(2) = 6.46717332$	$\sigma(x(2)) = 9.69851698$	0.0
MCA double precision	$\bar{x}(1) = 1.9999999992$	$\sigma(x(1)) = 8.4541287415 \times 10^{-9}$	8.3
	$\bar{x}(2) = -1.9999999988$	$\sigma(x(2)) = 1.26782603316 \times 10^{-8}$	8.2

TABLE 4: Resolution of equation 7 by using the MCA single and double precision floating point arithmetic

FP arithmetic	Result	s
IEEE-754 single precision	$x(1) = 1.33317912$	0
(default rounding)	$x(2) = -1.00000000$	0
IEEE-754 double precision	$x(1) = 2.00000000240030218$	9
(default rounding)	$x(2) = 2.00000000359962060$	9

TABLE 3: Resolution of equation 7 by using the IEEE-754 single and double precision floating point arithmetic and comparison to the exact solution

```
1  double a,b,c,d;
2  a=(2.0*sqrt(3.0))/3.0;
3  b=4.0/3.0;
4  c=a*a-b*b;
5  if (c>=0)
6    d=sqrt(c)+10.0;
7  else
8    d=sqrt(-c)+10.0;
9  return d;
```

The test on c at line 5 is to prevent the square root computation of a negative number. In the IEEE-754 standard, the square root of a negative number returns NaN (Not A Number).

Table 5 compares the exact value of d, its numerical evaluation by using IEEE-754 double precision with the rounding mode to the nearest and the numerical verification done both by CADNA and verificarlo.

The estimation of the number of significant digits (4.66 decimal digits) given by verificarlo is close to the number of accurate significant digits in IEEE-754 (4 decimal digits). On the other hand, the numerical verification performed by CADNA indicates that the result d is a numerical noise having no significant digit. CADNA, unlike verificarlo, works in a synchronous mode. Each floating point operation is computed three times with a random rounding mode towards plus or minus infinity, then the number of significant digits is computed using the Student law under a normality assumption. In the case of an if test, CADNA follows the branch which satisfies at least two out of the three samples. For the test at line 5, the three samples of c are $c_1 = 6.661339 \times 10^{-16}$, $c_2 = -2.220446 \times 10^{-16}$ and $c_3 = -2.220446 \times 10^{-16}$. Therefore CADNA decides that the test is false as two samples of c are negative. Unfortunately, the square root evaluation on the third sample produces an invalid NaN value. This case study shows that CADNA can produce invalid results on branch programs. Using CADNA on large code bases requires the help of an expert to detect invalid results due to branching. In contrast, MCA uses independent computing on these samples so no invalid computation is performed during this case study.

FP arithmetic	Result	s
Exact solution	$c = -50.0$	-
IEEE-754 double precision		
rounded to the nearest	$c = -0.02460...$	0
rounded towards $-\infty$	$c = -2073773.08...$	0
rounded towards $+\infty$	$c = -0.008202...$	0
rounded towards 0	$c = -0.008202...$	0

TABLE 6: Comparison between the exact value of c and its numerical evaluation by using IEEE-754 double precision with the rounding mode to the nearest, towards $-\infty$ and towards $+\infty$

4.4. Case study 4: Alternating counter

In the C code below, a counter c is initialized to 5×10^{13}. The counter is updated iteratively N times, with $N = 10^8$. For each update, c is incremented or decremented by 10^{-6} depending on the parity of the iteration number.

```
1  unsigned int i;
2  double c=-5e13;
3  for (i=0;i<100000000;i++) {
4    if (i%2==0)
5      c=c+1.e6;
6    else
7      c=c-1.e-6;
8  }
9  return c;
```

Assuming infinite precision, at the end of this process the expected exact value of c should be:

$$c = -5 \times 10^{13} + \frac{1}{2}10^8 10^6 - \frac{1}{2}10^8 10^{-6} = -50 \quad (9)$$

Table 6 compares the exact value of c to its numerical evaluation when using IEEE-754 double precision with the rounding mode to the nearest, towards $-\infty$ and towards $+\infty$.

The IEEE-754 arithmetic provides results having no significant digits whatever the rounding mode used: there is then a strong numerical problem. Table 7 reports the results of the numerical verification performed by CADNA and verificarlo and figure 7 the evolution of the number of significant bits in the result estimated by verficarlo and CADNA.

CADNA in this case study overestimates the number of significant digits of the result. Indeed, the result is a numerical noise with no significant digit. Furthermore,

FP arithmetic	Result	s	s'
Exact solution	$d = 10.$	-	-
IEEE-754 double precision	$d = 10.00035...$	-	5
CADNA double precision	$d = @.0$	0	NA: CADNA returns NaN
verificarlo 100 samples	$\bar{d} = 10.000521$ $\sigma(d) = 0.0002170856$	4.66	4

TABLE 5: Comparison between the exact value of d, its numerical evaluation by using IEEE-754 double precision with the rounding mode to the nearest and the numerical verification done both by CADNA and verificarlo. @.0 means that CADNA found that the result has no significant digits.

FP arithmetic	Result	s'	s
CADNA DP	$c = -0.103 \times 10^6$	3	0
	CADNA detects no instabilities, only estimated significant digits are printed		
verificarlo DP (1000 samples)	$\bar{c} = -45.101042$ $\sigma(c) = 44.33$	0	0

TABLE 7: Numerical verification of the computation of c performed both with CADNA and MCA

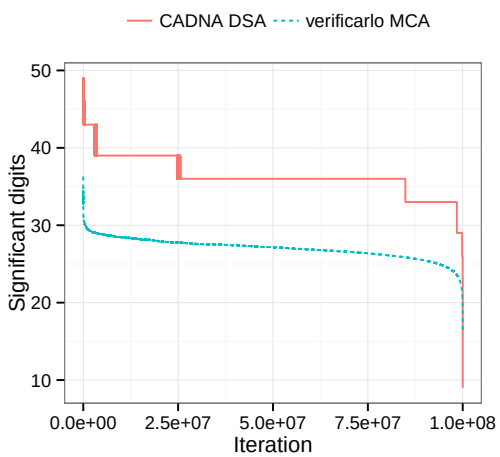

Figure 7: Evolution of the estimated significant number of bits with the iteration number

CADNA overestimates the number of significant for all the program iterations. The overestimation is not due to the small number of samples (N=3) used by CADNA. The issue here is that each CADNA sample performs the arithmetical operations with a rounding mode towards $+\infty$ or $-\infty$ with a probability equal to $\frac{1}{2}$. As this case study is a linear problem, the expected CADNA result should be the mean value of the c values computed with IEEE-754 rounding to $+\infty$ or $-\infty$ that is to say:

$$\begin{aligned} E(c_{cadna}) &\simeq \frac{-2073773.08...-0.008202...}{2} \\ &\simeq -103686 \simeq -0.103 \times 10^6 \end{aligned}$$

In contrast, the numerical verification performed with verificarlo shows that standard deviation is greater than the mean value of the MCA samples. It correctly indicates that the result computed using IEEE-754 floating point numbers is a numerical noise.

5. Limitations and future work

Verificarlo is a fully automatic tool to instrument an application for numerical precision analysis. The current version is stable and has been successfully used to analyse small and large code bases, yet it is limited in some respects.

As shown in section 3, the verificarlo runtime overhead is high. This is due to MCA inexact computations being performed with MPFR. When the desired virtual precision is low and known in advance, the overhead can be reduced by performing computations using a fixed precision implementation (e.g. double, quads) and avoiding the MPFR abstraction. This improvement is scheduled for the next version of verificarlo.

Unlike CADNA, verificarlo does not support numerical debugging out of the box. In the future we would like to include a mode that allows pinpointing the exact operation or routine that is to blame for a precision loss. We would also like to include a statistical post-treatment toolbox to go beyond the standard deviation analysis. This toolbox could help non-experts understand and interpret the output of the MCA analysis.

Finally, it is important to test the robustness of the MCA approach on different classes of numerical algorithms such as linear algebra or compensated algorithms and also full-scale real-life applications.

6. Conclusion

The control of the numerical accuracy of scientific codes becomes crucial in particular when using HPC ressources. It is also necessary to control the floating point computation when porting a scientific code on another programing

language or on different computing ressources. These tasks raise the need for a tool that automatically estimates, without the assistance of an expert, the interval of confidence of computed results. Verificarlo is the first step toward a fully automatic tool. It is based on the Monte-Carlo Arithmetic and uses a compiler approach to easily instrument the code that is executed.

Our case studies illustrate the advantages of using verificarlo for numerical analysis on scientific codes. They show that verificarlo overcomes some methodological and technical limitations of the CADNA library to estimate the numerical accuracy. Verificarlo is the first tool to implement MCA arithmetic at the intermediate representation level. Unlike CADNA or MCALIB+CIL, this allows to capture the effect of compiler optimizations on numerical accuracy.

Verificarlo is available at http://www.github.com/verificarlo/verificarlo under an open source licence.

References

[Cha88] Franoise Chatelin. On the general reliability of the cestac method. *C. R. Acad.Sci. Paris*, 1:851–854, 1988.

[CV88] Jean-Marie Chesneaux and Jean Vignes. On the robustness of the cestac method. *C. R. Acad.Sci. Paris*, 1:855–860, 1988.

[Duf11] Iain S. Duff. European exascale software initiative: Numerical libraries, solvers and algorithms. Technical Report TR/PA/11/118, CERFACS, 2011.

[FHL+07] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):13, 2007.

[FL15] Michael Frechtling and Philip H.W. Leong. Mcalib: Measuring sensitivity to rounding error with monte carlo programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 37(2):5, 2015.

[Hig02] Nicholas J. Higham. *Accuracy and stability of numerical algorithms*. Siam, 2002.

[Kah66] William Kahan. Numerical linear algebra. In *Canadian Mathematical Bulletin*, pages 756–801, 1966.

[Kah96] William Kahan. The improbability of probabilistic error analyses for numerical computations. In *UCB Statistics Colloquium, evans hall edition*, page 20, 1996.

[Kah06] William Kahan. How futile are mindless assessments of roundoff in floating-point computation? Technical report, 2006.

[LA04] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*, pages 75–86. IEEE, 2004.

[MDCL13] Sethy Montan, Christophe Denis, Jean-Marie Chesneaux, and Jean-Luc Lamotte. Efficient matrix multiplication based on discrete stochastic arithmetic. *Reliable Computing*, 19(4):398–416, 2013.

[MN98a] Makoto Matsumoto and Takuji Nishimura. Dynamic creation of pseudorandom number generators. *Monte Carlo and Quasi-Monte Carlo Methods*, 2000:56–69, 1998.

[MN98b] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[NMRW02] George C Necula, Scott McPeak, Shree P Rahul, and Westley Weimer. Cil: Intermediate language and tools for analysis and transformation of c programs. In *Compiler Construction*, pages 213–228. Springer, 2002.

[Par97] Scott Parker. Monte carlo arithmetic: exploiting randomness in floating-point arithmetic. Technical Report CSD-970002, UCLA Computer Science Dept., 1997.

[Rev03] Nathalie Revol. Interval newton iteration in multiple precision for the univariate case. *Numerical Algorithms*, 34(2-4):417–426, 2003.

[SJDC07] Norman Stanley Scott, Fabienne Jézéquel, Christophe Denis, and Jean-Marie Chesneaux. Numerical 'health check' for scientific codes: the CADNA approach. *Computer Physics Communications*, 176(8):507–521, 2007.

[Vig04] Jean Vignes. Discrete stochastic arithmetic for validating results of numerical software. *Numerical Algorithms*, 37(1-4):377–390, 2004.

[VLP74] Jean Vignes and Michel La Porte. Error analysis in computing. In *Proceedings of IFP 1974*, pages 610–614. IFP, 1974.

[YYL11] Jackson H.C. Yeung, Evangeline F.Y. Young, and Philip H.W. Leong. A monte-carlo floating-point unit for self-validating arithmetic. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 199–208. ACM, 2011.