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-01192668v4

HAL Id: hal-01192668
https://hal.archives-ouvertes.fr/hal-01192668v4
Preprint submitted on 7 Nov 2018

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 - Centre de Mathématiques et de Leurs Applications, ENS Cachan
Université Paris-Saclay et CNRS UMR 8536, 94235 Cachan France
Email: christophe.denis@cmla.ens-cachan.fr
† Université de Versailles Saint-Quentin-en-Yvelines
Université Paris-Saclay
Email: {pablo.oliveira, eric.petit}@uvsq.fr

Abstract—Numerical accuracy of floating point computation is a well studied topic which has not made its way to the end-user in scientific computing. Yet, it has become a critical issue with the recent requirements for code modernization to harness new highly parallel hardware and perform higher resolution computation. 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. Unlike source-to-source approaches, our implementation captures the influence of compiler optimizations on the numerical accuracy. We illustrate how Monte Carlo Arithmetic using the verificarlo tool outperforms the existing approaches on various use cases and is a step toward automatic numerical analysis.

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 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 it guarantees the result, 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 adapted 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 a practical 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 compute stochastic confidence intervals on the results by applying random perturbations on the numerical operations. For example, the Discrete Stochastic Arithmetic implemented in the CADNA library perturbates computations by randomly changing the rounding mode. Discrete Stochastic Arithmetic is based on CESTAC developed by Vignes in 1974. CADNA is a powerful numerical debugger tool which has been used to solve real problems. Nevertheless, CADNA has some limitations. First, Chatelin
and Parker [Cha88], [Par97] show that the CESTAC assumptions are not always verified when performing numerical analysis, which can introduce errors in CADNA estimations. Second, using CADNA requires manually modifying the original program sources to use special CADNA types. For large code bases, this process can be costly.

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. Verificarlo operation is transparent for the user and does not require manually modifying the source code.
- 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. Section 3 introduces the verificarlo tool, its advantages and limitations, and compares it to other approaches. Finally, section 4 proposes a set of experiments to validate verificarlo 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 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.

#### 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 $P_N = 2^{t-N}$ that all the $N$ samples compute a FP operation with the same rounding mode. The number of significant digits is computed 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...
rional stability of a function or program. To encourage

Carlo Arithmetic analysis

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 co-processors. 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 exe-
between the IEEE-754 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 estimates that the IEEE-754 DP result is a numerical noise.

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 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

#### 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.

<table>
<thead>
<tr>
<th>version</th>
<th>samples</th>
<th>total time (s)</th>
<th>time/sample (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>original program</td>
<td>1</td>
<td>.056</td>
<td>.056</td>
</tr>
<tr>
<td>CADNA</td>
<td>3</td>
<td>2.93</td>
<td>.97</td>
</tr>
<tr>
<td>MCALIB</td>
<td>128</td>
<td>1184.02</td>
<td>9.25</td>
</tr>
<tr>
<td>verificarlo</td>
<td>128</td>
<td>834.57</td>
<td>6.52</td>
</tr>
<tr>
<td>verificarlo 16 threads</td>
<td>128</td>
<td>54.39</td>
<td>.42</td>
</tr>
</tbody>
</table>

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

```c
float sum = f[0];
float c = 0.0, y, t;
for (int i=1;i<n;i++) {
  y = f[i] - c;
  t = sum + y;
  c = (t - sum) - y;
  sum = t;
}
return sum;
```
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 $c$.

<table>
<thead>
<tr>
<th>flags</th>
<th>sum relative error</th>
<th>-O0</th>
<th>-O3 -ffast-math</th>
</tr>
</thead>
<tbody>
<tr>
<td>CADNA</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>CIL+MCALIB</td>
<td>7.3</td>
<td>7.3</td>
<td></td>
</tr>
<tr>
<td>verificarlo</td>
<td>7.3</td>
<td>5.8</td>
<td></td>
</tr>
</tbody>
</table>

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 ($\sigma$) of Kahan’s compensated sum computed on 1000 verificarlo samples. The compensated -O0 version has a constant error while the -O3 -ffast-math error increases as $O(\epsilon \sqrt{n})$.

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

$$
\begin{pmatrix}
0.2161 & 0.1441 \\
1.2909 & 0.8648
\end{pmatrix}
x =
\begin{pmatrix}
0.1440 \\
0.8642
\end{pmatrix}
$$

(7)

The exact solution of equation 7 is:

$$
\begin{pmatrix}
x \\
-2
\end{pmatrix}
$$

(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:

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

The test on `b` at line 4 prevents 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 (9.13 decimal digits) is reasonable given that the rounded IEEE-754 computation is exact. On the other hand, the numerical verification performed by CADNA indicates that the result `c` is a numerical noise having no significant digit.

In verificarlo, each sample execution can follow a different branch in the code; executions are independent. CADNA, unlike verificarlo, works in a synchronous mode. Each floating point operation is computed three times with different rounding mode towards plus or minus infinity. Then a reconciliation process is used to select a single branch outcome for the three CESTAC traces. For the test at line 4, the three samples of `b` are `b1 = -0`, `b2 = 2.22045 \times 10^{-16}` and `b3 = -2.22044 \times 10^{-16}`. In this case, CADNA reconciliation uses the mean of the three traces, which is positive, and concludes that the test is true. Unfortunately, the square root evaluation on the third negative 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.

4.4. Case study 4: Alternating counter

In the C code below, a counter `c` is initialized to `5 \times 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.

```c
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.e6;
8 }
9 return c;
```

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

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

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 verificarlo 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, 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

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

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Significant digits</th>
<th>CADNA</th>
<th>DSA</th>
<th>verificarlo</th>
<th>MCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0e+00</td>
<td>2.5e+07</td>
<td>7.5e+07</td>
<td>1.0e+08</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.33317912</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-1.00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Comparison of CADNA, DSA, verificarlo, and MCA in the case study.

- **CADNA DSA** indicates the result is valid.
- **verificarlo MCA** indicates the result is invalid.

- **IEEE-754 single precision** (default rounding)
- **IEEE-754 double precision** (default rounding)

The IEEE-754 arithmetic provides results having no significant digits whatever the rounding mode used: there is then a strong numerical problem.
15 and 17.

small and large code bases, yet it is limited in some respects.

version is stable and has been successfully used to analyse
application for numerical precision analysis. The current

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.

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.

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

<table>
<thead>
<tr>
<th>FP arithmetic</th>
<th>Mean value</th>
<th>Standard deviation</th>
<th>s'</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCA single precision</td>
<td>( \bar{x}(1) = 1.02463705 )</td>
<td>( \sigma(x(1)) = 6.46717332 )</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>( \bar{x}(2) = 6.46717332 )</td>
<td>( \sigma(x(2)) = 9.69851698 )</td>
<td>0.0</td>
</tr>
<tr>
<td>MCA double precision</td>
<td>( \bar{x}(1) = 1.9999999992 )</td>
<td>( \sigma(x(1)) = 8.4541287415 \times 10^{-9} )</td>
<td>8.3</td>
</tr>
<tr>
<td></td>
<td>( \bar{x}(2) = -1.9999999988 )</td>
<td>( \sigma(x(2)) = 1.26782603316 \times 10^{-8} )</td>
<td>8.2</td>
</tr>
</tbody>
</table>

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. \( \odot.0 \) means that CADNA found that the result has no significant digits.

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 resources. It is also necessary to control the floating point computation when porting a scientific code on another programming language or on different computing resources. 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**


<table>
<thead>
<tr>
<th>FP arithmetic</th>
<th>Result</th>
<th>Significant digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>CADNA DP</td>
<td>( c = -0.103 \times 10^6 )</td>
<td>3</td>
</tr>
<tr>
<td>verificarlo DP</td>
<td>( \bar{c} = -4.5101042 )</td>
<td>( \sigma(c) = 44.33 )</td>
</tr>
</tbody>
</table>

**TABLE 7:** Numerical verification of the computation of \( c \) performed both with CADNA and MCA.


