Floating Point Computing Verification on new architecture and large scale systems: Monte Carlo Arithmetic and Verificarlo

Eric Petit*, Pablo Oliveira, Yohan Chatelain, Damien Thenot, David Defour

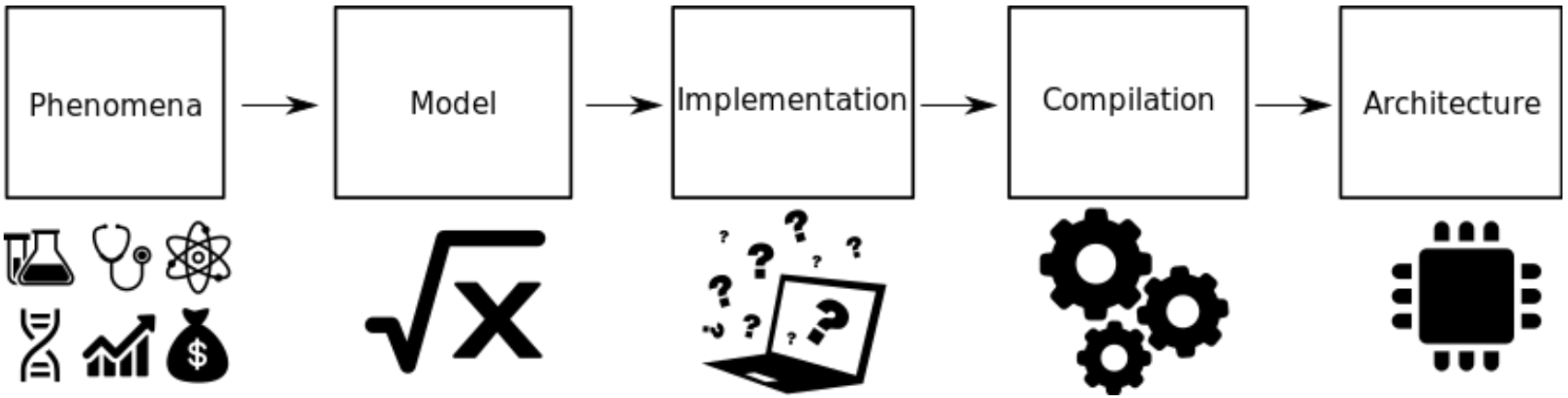
*Intel DCG E&G; UVSQ; Exascale Computing Research

2018-09 IXPUG tutorial

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY

Context

Building fast and robust applications for HPC is a complex task !



- At each step, numerical bugs can be introduced

Objective: Track and analyze numerical bugs

Architectural evolution cycles impact simulation results

- HPC systems and modern CPU are generally NOT deterministic!
  - runtimes (e.g. scheduling decisions)
  - parallel algorithm, asynchronousity
  - memory alignment
  - weak normalization
  - compiler optimization, vector size, FMA
- Some architectural improvement requires to 'break the rules', explicitly or not, of sequential program semantic
- This makes the numerical precision a recurrent hot topic every-time a major change is happening on the hardware side and propagating in the full system.
  - vector instruction set, multi-core, manycore, FMA, accelerators...
- The need for precision/reproducibility must be carefully analyze and optimize.

Motivation

Changing algorithm, architecture, parallelisation, heterogeneity, compiler, optimizations level and language can result in different numerical results.

Does different results means wrong results?

Outline IXPUG session

- Verificarlo is a tool to model IEEE-754 floating point model error
  - Estimate significant digits of a computation
  - Estimate the impact of precision change
  - rely on Monte Carlo Arithmetic
- VeriTracer is a trace tool for verificarlo
  - Trace precision and range of computations in a program across time
- VeriTracer-GUI is a graphical interface to veritracer
  - Interactive plot of Veritracer output

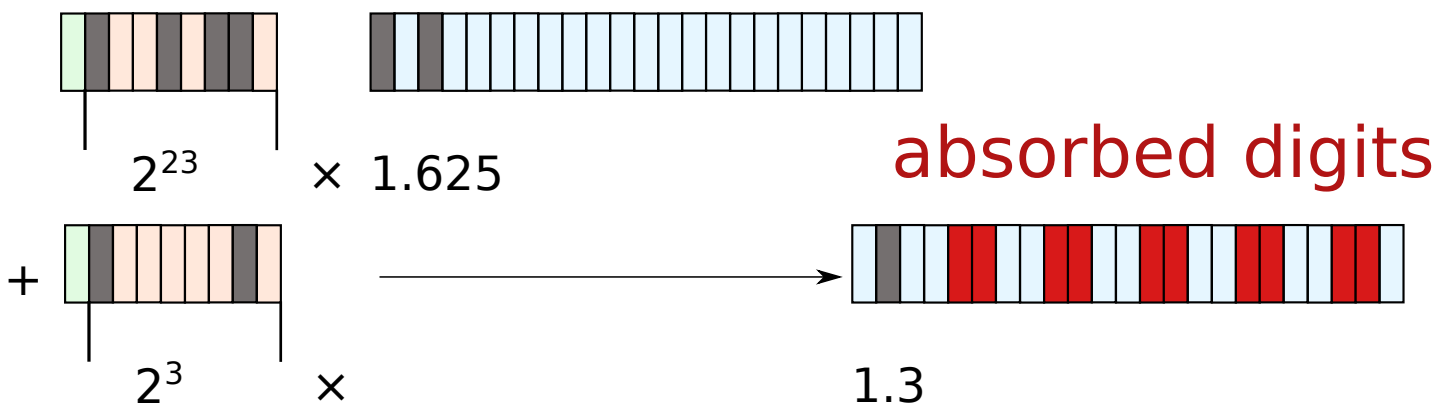
Outline

Floating point computation: the IEEE-754 standard

- What Every Computer Scientist Should Know About Floating-Point Arithmetic, *David Goldberg*, 1991 ACM issue of Computing Surveys
- Floating point (FP) numbers approximate real numbers with a finite precision
  - Discrete and finite set of values
  - In base 2
- Different representation and encoding in memory defined in IEEE 754
- Trade-off between range and precision
  - Single ($1 + 8 + 23 = 32$ bits), Double ($1 + 11 + 52 = 64$ bits)...
- And four rounding modes :
  - nearest, toward $+\infty$, toward $-\infty$, toward zero

Floating point computation: some adverse effects

- Representation errors
  - 3.14159265359300
- A floating point computation approximates the exact computation
- Loss of arithmetical properties (for example the floating point summation is not associative)
  - Absorption, a part of the significant digits cannot be represented in the result format
  - Cancellation, relative error when subtracting variables with very close values



The most common sources of FP arithmetic bugs, indeterminism or imprecision in HPC

- Large summation: dot product, integral computation, global values reduction (global energy...)
- Gradient computation of near values: small variations in large quantities, gradient with neighbor (e.g. stencil, CFD), residual
- Small contributions overtime: explicit methods, last iterations of a linear solver
- Duplication of mathematically equivalent computation on parallel actors
- Or a combination of the above: L2 norm of a residual, standard-deviation...

Outline

Monte Carlo Arithmetic [Stott Parker, 1999]

- Each FP operation may introduce a δ error

$$z = fl[x + y] = (x + y)(1 + \delta)$$

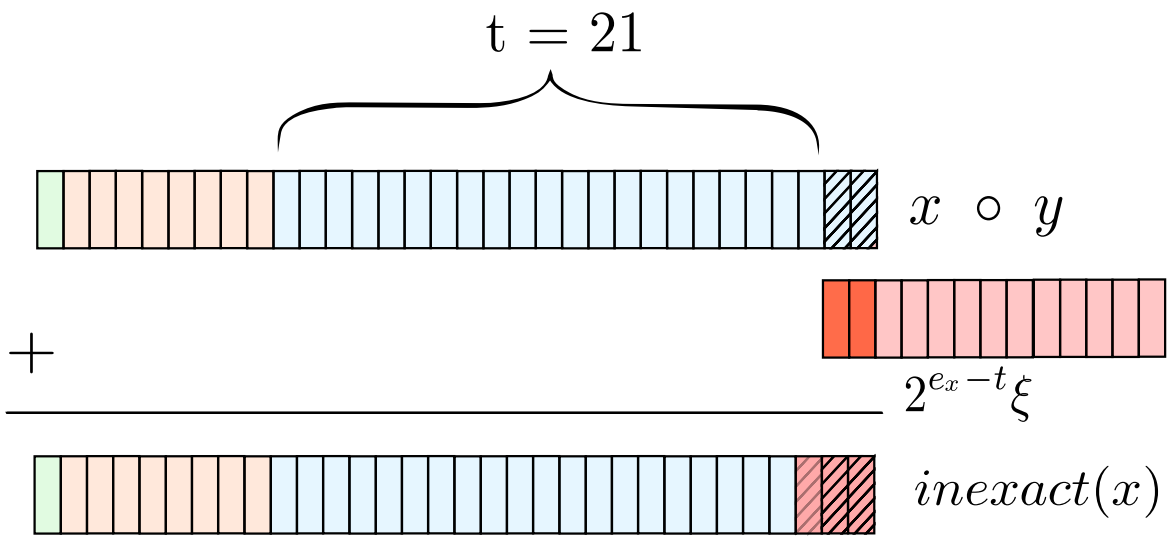
- When chaining multiple operations, errors can accumulate and snowball
- Monte Carlo Arithmetic key principle
  - Make δ a random variable
  - Use a Monte Carlo simulation to empirically estimate the FP error distribution

Monte Carlo Arithmetic: Random Rounding

- MCA simulates error with

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

  - $e_x = \lfloor \log_2 |x| \rfloor + 1$ is the order of magnitude of x;
  - ξ is an uniform random variable in $]-\frac{1}{2}, \frac{1}{2}[$;
  - t is the virtual precision, selects the magnitude of the simulated error.



FP operations $\circ$ are replaced by:

$$mca(x \circ y) = round(inexact(x \circ y))$$

absorption and rounding errors

Absorption example

- Remember our absorption example ?

```
float a = 13631488.0f; // 1.625 * 2^23
float b = 10.4f; // 1.3 * 2^3
printf("%0.7f\n", a+b);
```

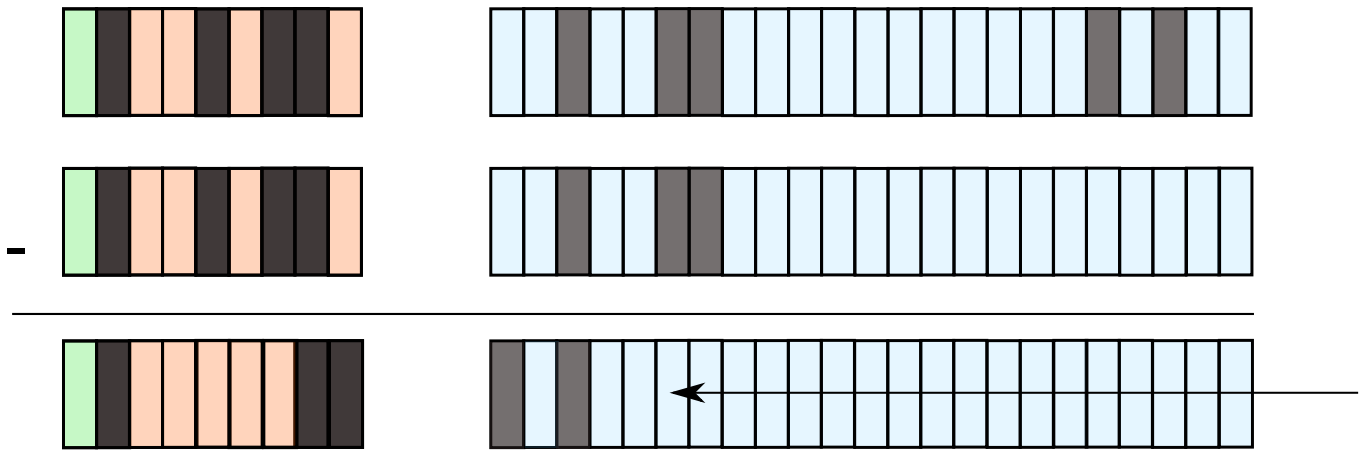
- Exact result = 13631498.4
- IEEE-754 result = 13631498 (always)
- MCA Random Rounding noise with $t = 24$

Sample	Result
1	13631498
2	13631499
3	13631498

- Exact digits are common across MCA samples
- Error digits change across MCA samples

Catastrophic Cancellation

- A cancellation happens when we subtract two close values:
  - $9633812.0 - 9633792.0 = 20.000000$



- Random Rounding models this operation as exact: the right-digits are always the same.

Sample	MCA RR t=53
1	20.0000000
2	20.0000000
3	20.0000000

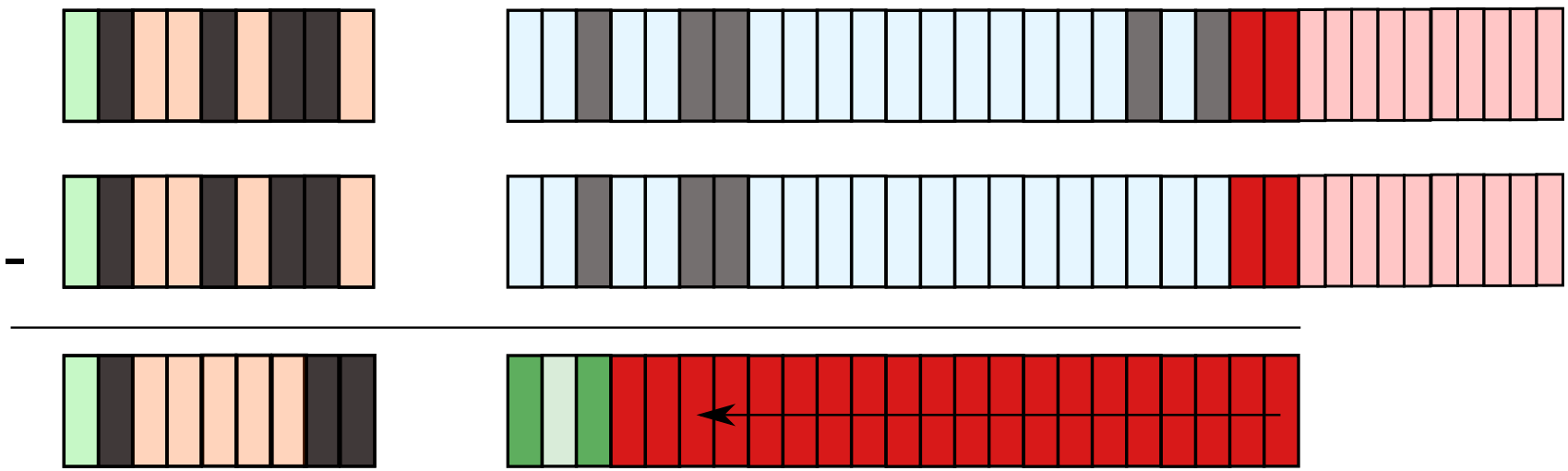
- How to model the fact that the right-digits came out of thin-air and information may have been lost ?

Monte Carlo Arithmetic: Precision Bound Mode

$$mca(x \circ y) = round(inexact(x) \circ inexact(y))$$

cancellation

mca noise



Across multiple MCA executions: error digits will change
significant digits will stay stable

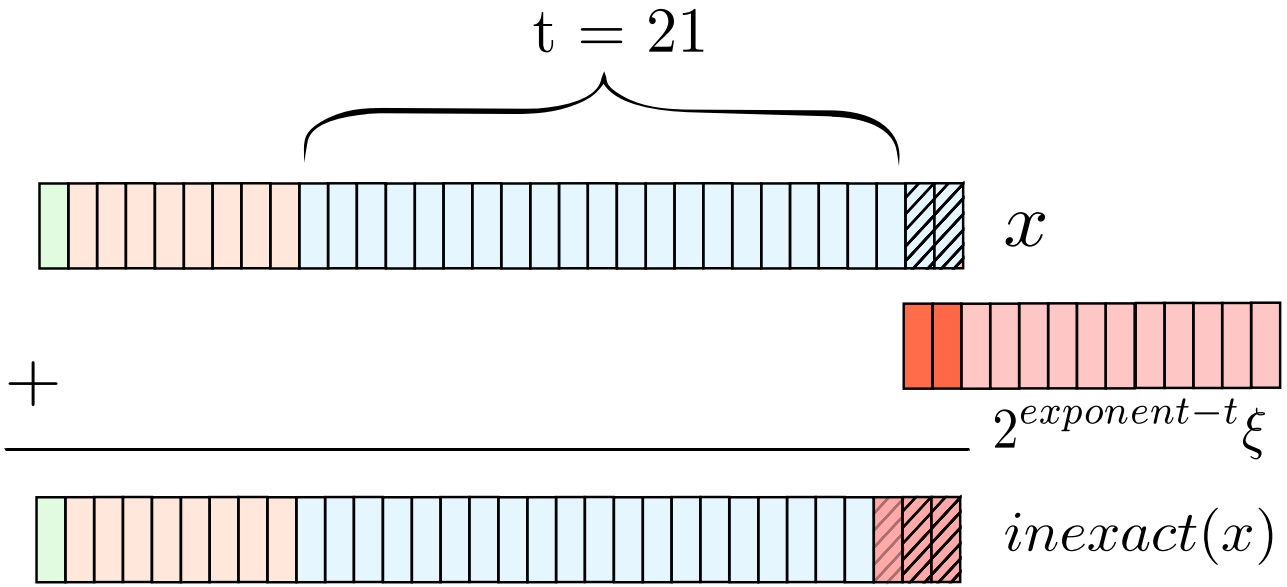
Precision Bound Mode Example

```
float a = 9.633812E6 ;
float b = 9.633792E6  ;
printf("%0.7f\n", a - b);
```

Sample	MCA PB t=24	MCA RR t=24
1	19.9846210	20.0000000
2	20.0592098	20.0000000
3	19.8788948	20.0000000

Monte Carlo Arithmetic: Full MCA Mode

- Full MCA Mode combines both Random Rounding and Precision Bound.



FP operations $\circ$ are replaced by:

$$mca(x \circ y) = round(inexact(inexact(x) \circ inexact(y)))$$

absorption

cancellation

Monte Carlo Arithmetic: Estimating output error

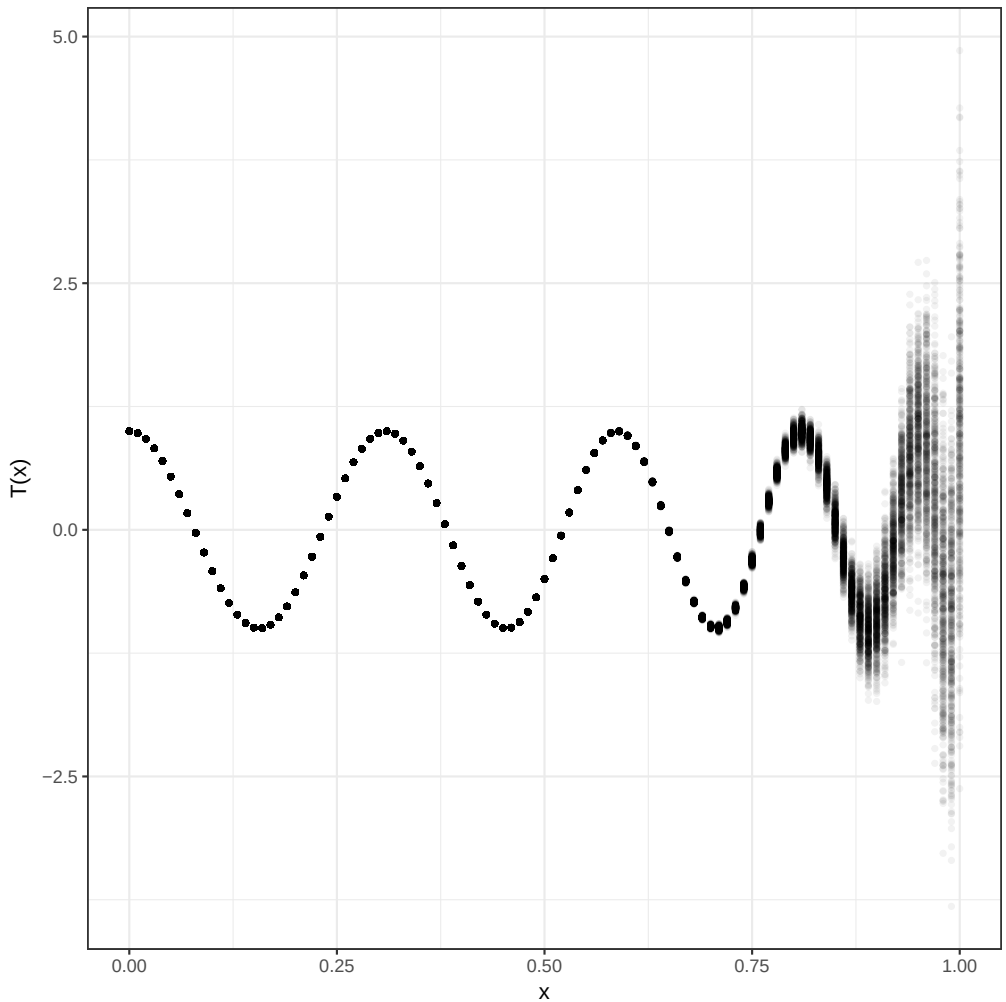


Figure: Tchebychev Polynomial $T_{20}(x) = cos(20 * cos^{-1}(x))$, catastrophic cancellation near 1 [Wilkinson, 1994]

How to measure the significance of a result ?

- MCA simulates stochastically the effect of IEEE-754 arithmetic errors.
- $X_1, X_2, \ldots, X_n$ are the values returned by n different MCA runs of a program.
- Stott D. Parker defines the number of significant digits as $\hat{s} = -\log \frac{\hat{\sigma}}{|\hat{\mu}|}$
  - $\hat{\sigma}$ is the empirical standard deviation of X
  - $\hat{\mu}$ is the empirical average of X
- This term is the magnitude of the signal to noise ratio.
- We also propose a unifying in [13], and rigorous probabilistic definition of the number of significant digits.
  - Stott Parker formula has 68% chances of being significant with confidence 95% for a centered normal distribution.

Ill conditioned System by W. Kahan

$$\begin{pmatrix} 0.2161 & 0.1441 \\ 1.2969 & 0.8648 \end{pmatrix} x = \begin{pmatrix} 0.1440 \\ 0.8642 \end{pmatrix}, x_{exact} = \begin{pmatrix} 2 \\ -2 \end{pmatrix}$$

- Results obtained using the LAPACK routines (condition number 2.497e8)

$$x_{single} = \begin{pmatrix} 1.33317912 \\ -1.00000000 \end{pmatrix} x_{double} = \begin{pmatrix} 2.0000000240030218 \\ -2.0000000359962060 \end{pmatrix}$$

- How to automatically estimate s, the number of significant digits ?
  - Without computing the exact theoretical answer
  - On a whole full scale scientific code
  - Without modifying the application code
  - Taking into account compiler optimizations

Ill conditioned System by W. Kahan

- Computations using IEEE-754 FP numbers (condition number 2.497e8)

Precision	**Result**	s
SP	$x_1 = 1.33317912$	0
	$x_2 = -1.00000000$	0
DP	$x_1 = 2.0000000240030218$	9
	$x_2 = 2.0000000359962060$	9

- Computation performed with MCA ($N = 1000$ samples)

Precision	$\hat{\mu}$	$\hat{\sigma}$	$\hat{s}$
MCA SP	$\hat{\mu}_1 = 1.02463705$	$\hat{\sigma}_1 = 6.4...$	0.0
	$\hat{\mu}_2 = 6.46717332$	$\hat{\sigma}_2 = 9.6...$	0.0
MCA DP	$\hat{\mu}_1 = 1.9999999992$	$\hat{\sigma}_1 = 8.4... \times 10^{-9}$	8.3
	$\hat{\mu}_2 = -1.9999999988$	$\hat{\sigma}_2 = 1.2... \times 10^{-8}$	8.2

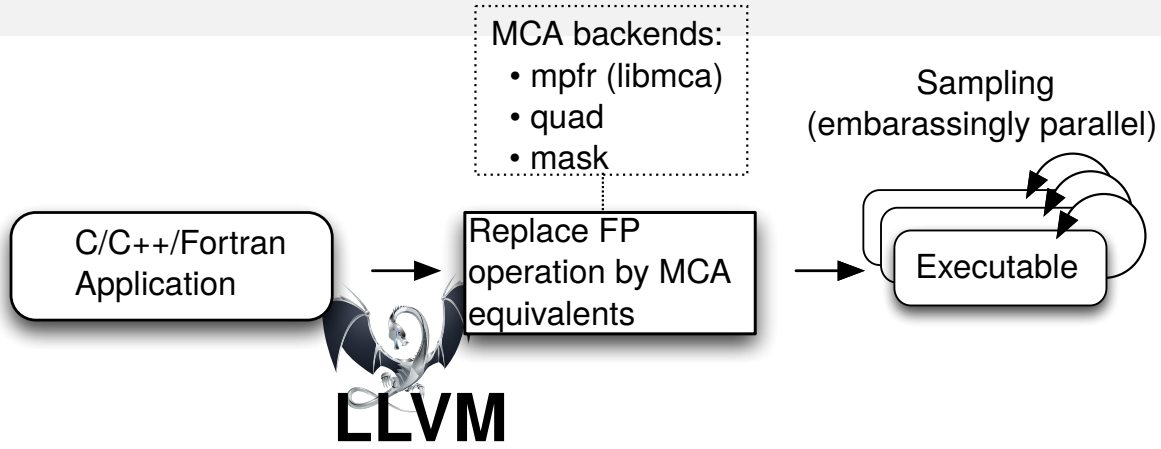
- For this example, Verificarlo automatically instrumented LAPACK and BLAS libraries without any modification of their source code

Outline

Introducing Verificarlo

- Verificarlo is a numerical debugger for the IEEE-754 floating point model
  - Estimate significant digits of a computation
  - Compromise between performance, precision and reproducibility
  - Open Source GPLv3 at `github.com/verificarlo/verificarlo`

Verificarlo



- Transparently transforms code to Monte Carlo Arithmetic
- Operates on optimized code: evaluates floating point errors introduced by compiler optimizations

```
-   %37 = fadd fast <4 x float> %wide.load.2, %31
+   %37 = call <4 x float> @_4x_mca_floatadd(<4 x float> %wide.load.2,
                                             <4 x float> %31)
```

Verificarlo: an Automatic LLVM Tool for FP Accuracy Checking using MCA

- Using LLVM brings advantages:
  - Supports multiple languages and multiple ISA
  - Powerful dependency and source information analysis of the LLVM compiler
    - e.g. per function/loop analysis, access to debug info, ...
  - Fast instrumentation: no emulation; can limit instrumentation to critical functions
  - Most of the compiler optimizations are captured
- But also some constraints:
  - Tied to LLVM compiler
  - Cannot handle pre-compiled libraries and some backend optimizations

Compiler optimizations are instrumented

- Instrumentation occurs **just before code generation**
- Enables analyzing precision loss due to (most) compiler optimizations

```
for (int i=1;i<n;i++) {
  y = f[i] - c;
  t = sum + y;
  c = (t - sum) - y;
  sum = t;
 }
return sum;
```

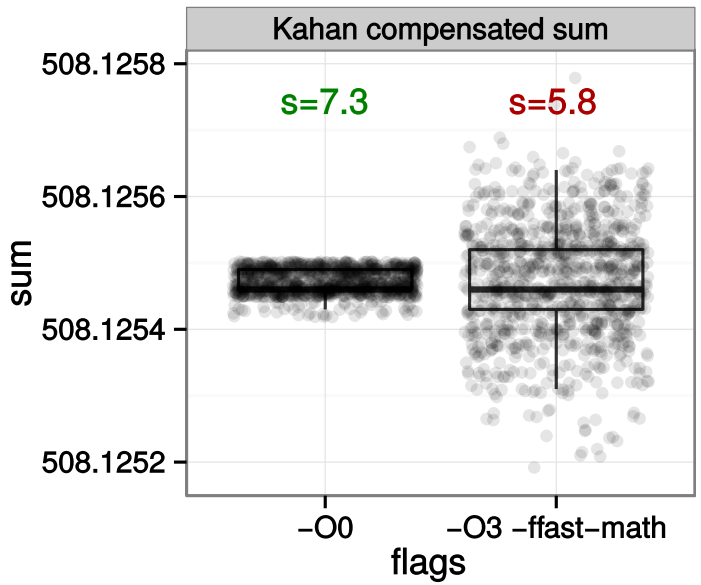


Figure: Analysis of the effect of compiler flags on a Kahan compensated sum algorithm (RR mode only)

Kahan Sum

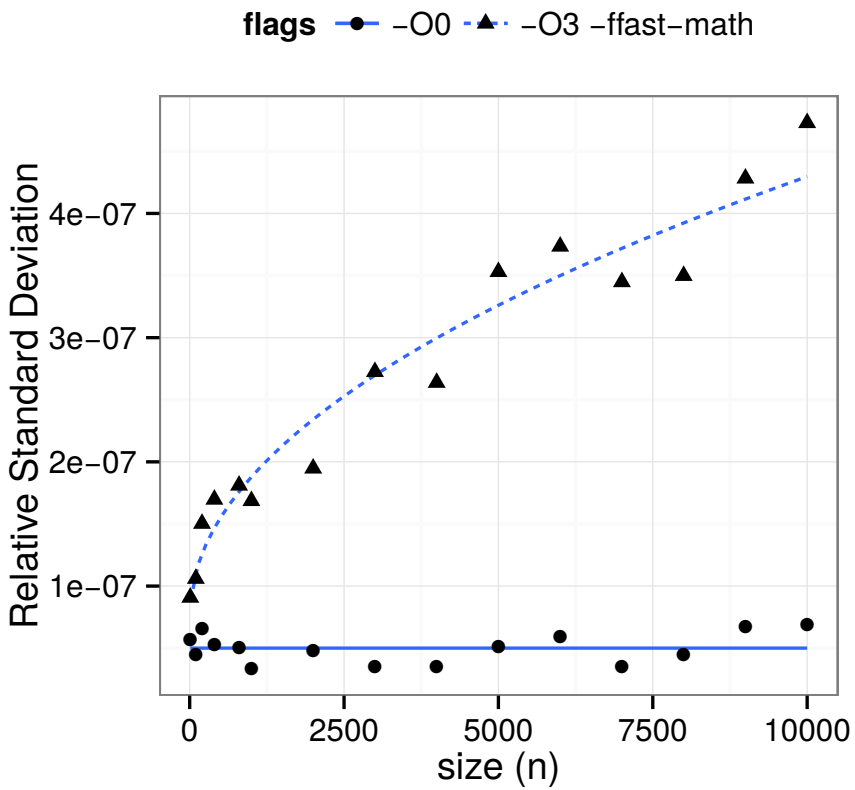


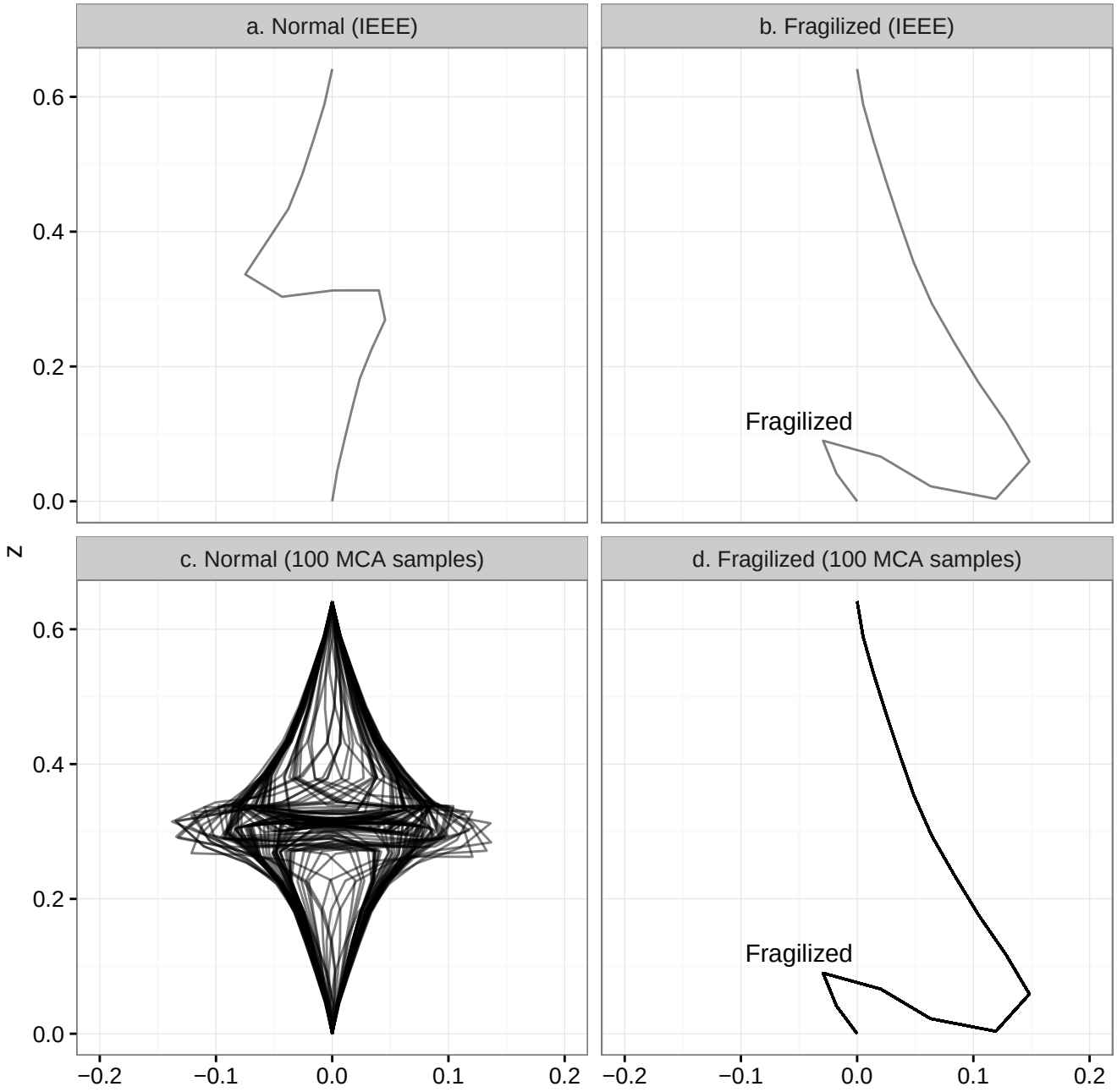
Figure: Relative Standard Deviation $\frac{\sigma}{\mu}$ of Kahan's compensated sum computed on 1000 verificarlo samples. The compensated `-O0` version has a constant error while the `-O3 -fast-math` error increases as $O(\epsilon\sqrt{n})$.

Study on Europlexus (EPX)

`www-epx.cea.fr` collaboration with O. Jamond (CEA)

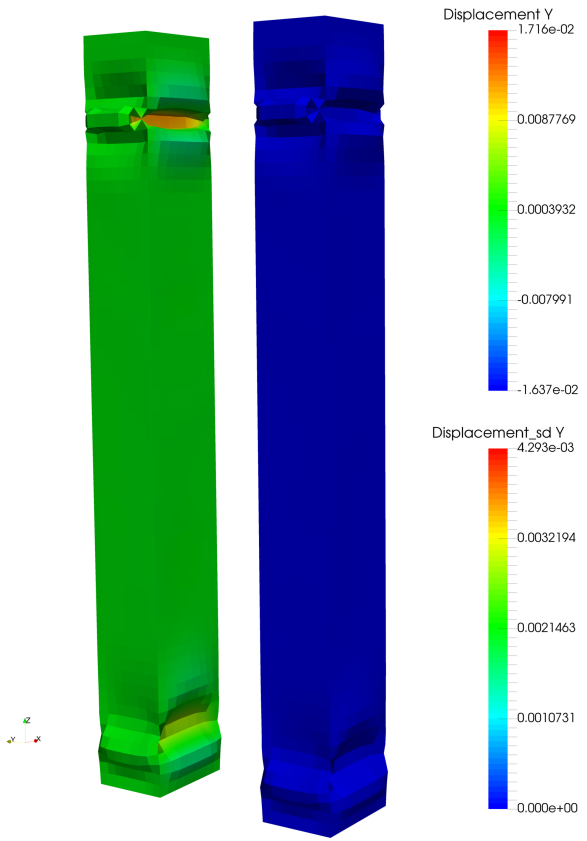
- Fast transient dynamic simulation
  - soundness of mechanical confinement barriers in nuclear reactors
  - soundness of public structures in case of explosions
- 1 million lines of Fortran 77 and Fortran 90
- Instrumented out of the box with Verificarlo without any change to the source code
- Checking FP reproducibility
  - Port to new architectures with more vectorization or parallelism
  - Select reproducible regression checks for code modernization

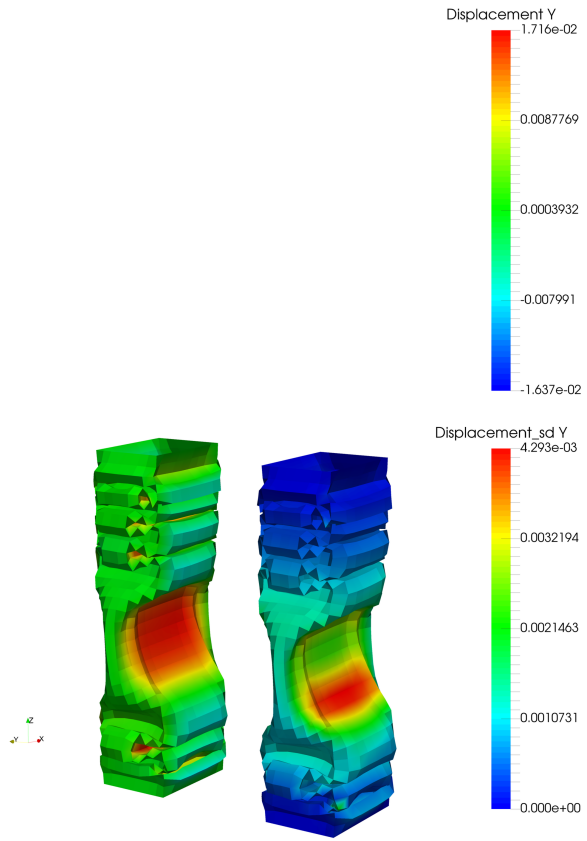
1D beam example



VTK plugin for Europlexus

- VTK plugin includes multiple views allowing a refinement of the analysis
- Using stdev and mean allows to better localize the loss of precision on the beam





Outline

Muller's sequence[2] overview

$$u_n = \begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{111}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{cases} \quad \lim_{n\to\infty} u_n =$$

exact solution

6

100

finite precision

Verificarlo

u_{30}
mean: 100
significant digits: 16

Muller's sequence[2] overview

$$u_n = \begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{111}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{cases}$$

$\lim_{n\to\infty} u_n =$

exact solution

6

100

finite precision

Verificarlo

u_{30}
mean: 100
significant digits: 16

Without an exact solution

- Only checking the final result may leads to wrong conclusions !
- The result is precisely wrong :-) And reproducible...

Current challenges

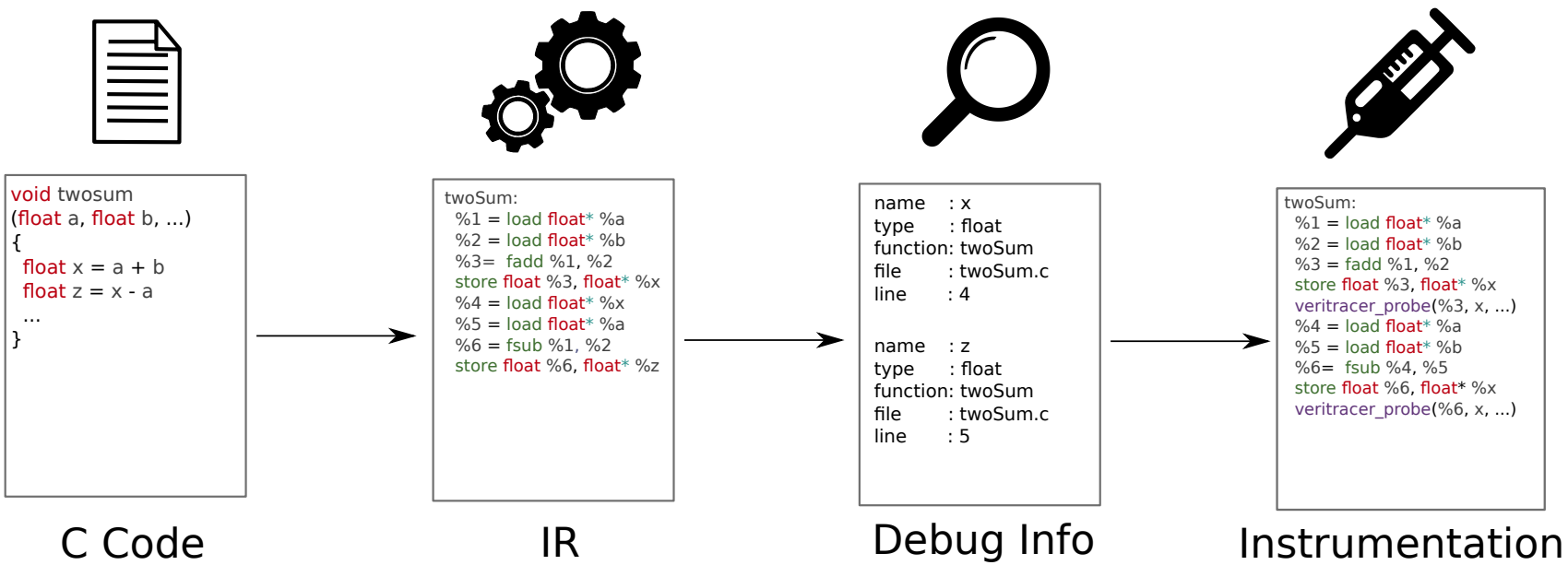
Existing tools explore the spatial dimension of numerical computations:

- which variable or operation is imprecise
- which function can be switched into lower precision
- but programs have different numerical requirements over time

$\Rightarrow$ **Need to explore the temporal dimension**

Veritracer LLVM pass

- Accepts any source code accepted by the LLVM frontend
- Transforms into an LLVM Intermediate Representation
- Searches debug information through LLVM IR
- Instruments the IR code with veritracer probes



Implementation

Extends verificarlo to

- Trace the precision of FP variables over time
- Provide contextual information on traced variables
  - Current version implements Context Analysis
  - Inserts call to `backtrace()` (GNU C library)
  - Analyzes backtraces to detect flow divergences
- All in an automatic way

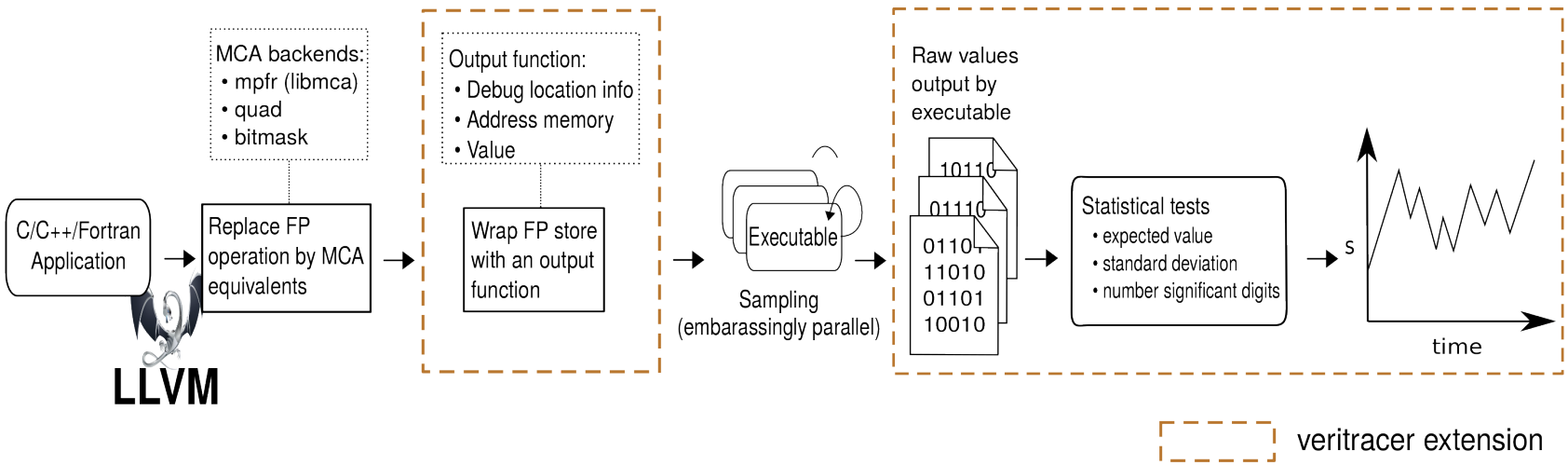


Figure: Veritracer's workflow

Muller's sequence with veritracer

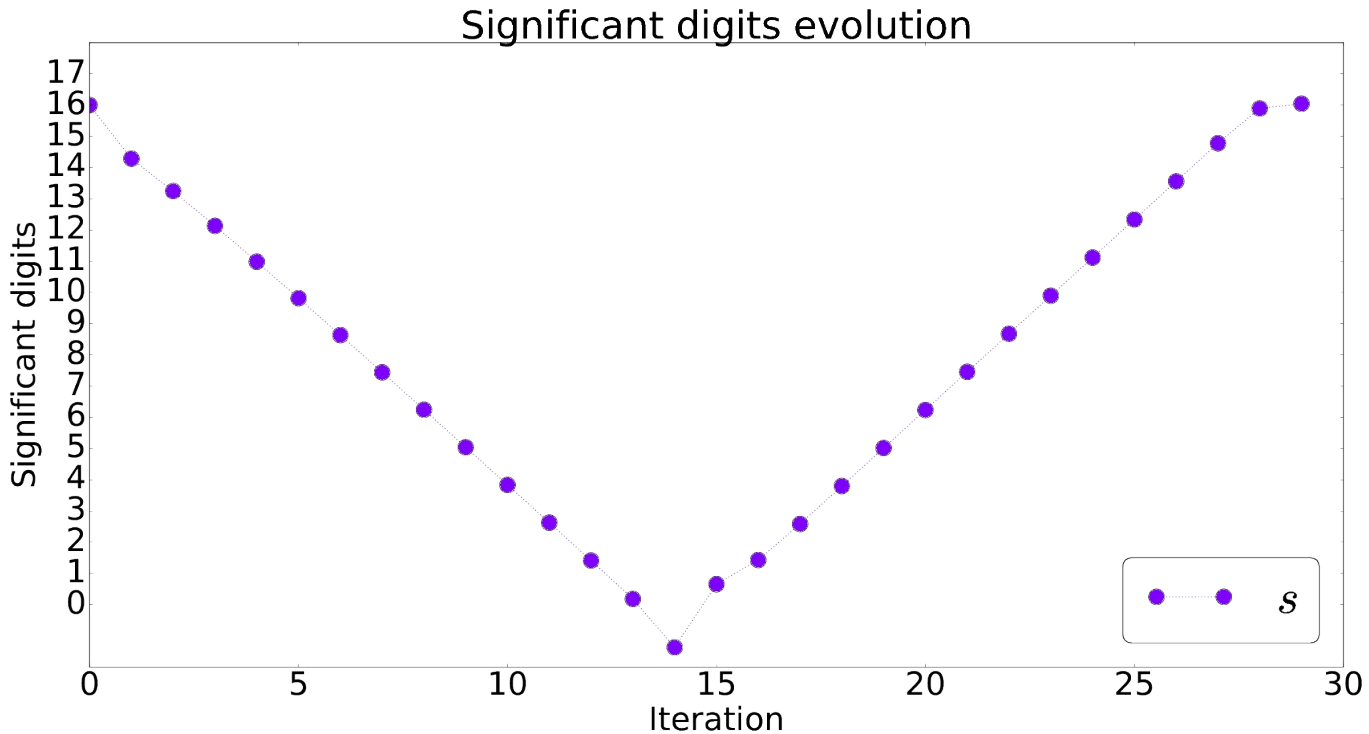


Figure: Muller's sequence evaluation over 500 samples with verificarlo. Evolution of the number of significant digits of u_n over time. Compiled with double format and executed with $t = 53$ in RR mode

- Automatically plots the number of significant digits over time
- At $n = 14$ there are no more significant digits correct

Heuristic to pinpoint relevant function to trace

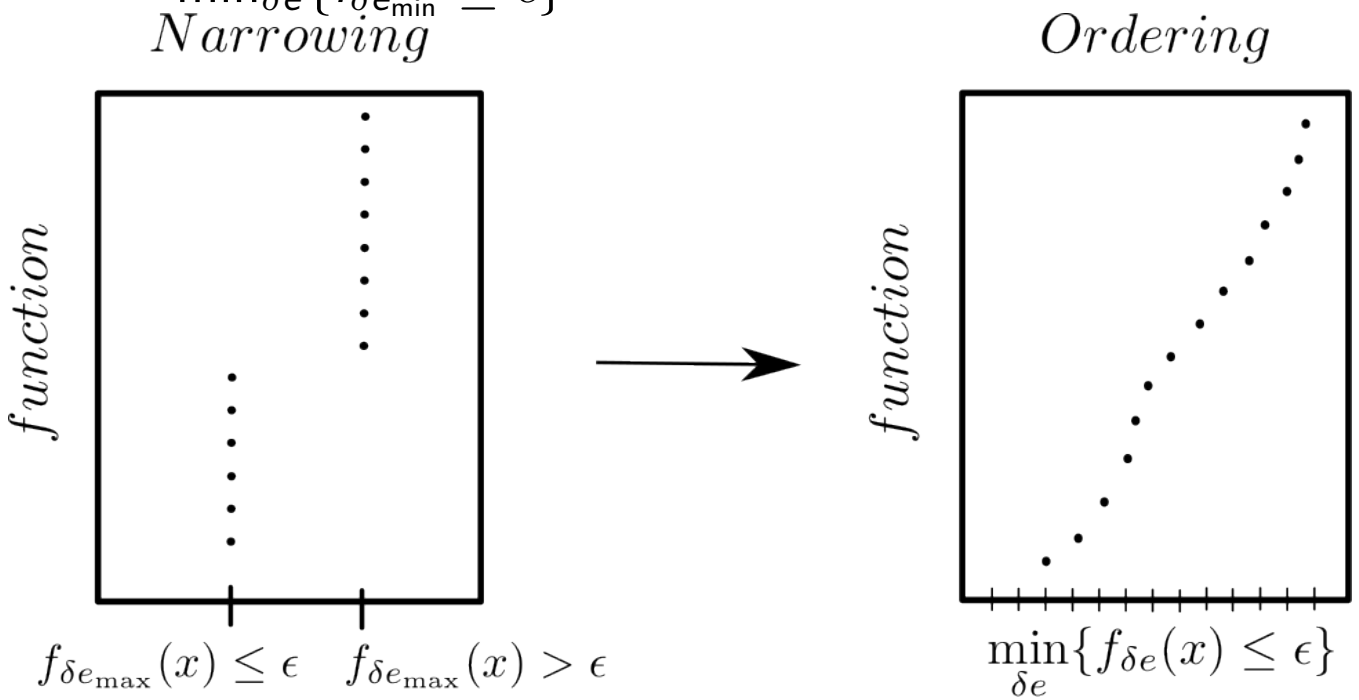
Narrowing the search space

To narrow the search space, we:

Scan functions with FP values

Narrow the set by removing non-contributing functions $f_{\delta e_{\max}}(x) \leq \epsilon$,

Order functions by *numerical stress resistance* $\min_{\delta e}\{f_{\delta e_{\min}} \leq \epsilon\}$



ABINIT [6]

- Collaboration with CEA (M. Torrent and J. Bieder)
- Calculates observable properties of materials (optical, mechanical, vibrational)
- Works on any chemical composition (molecules, nanostructures, solids)

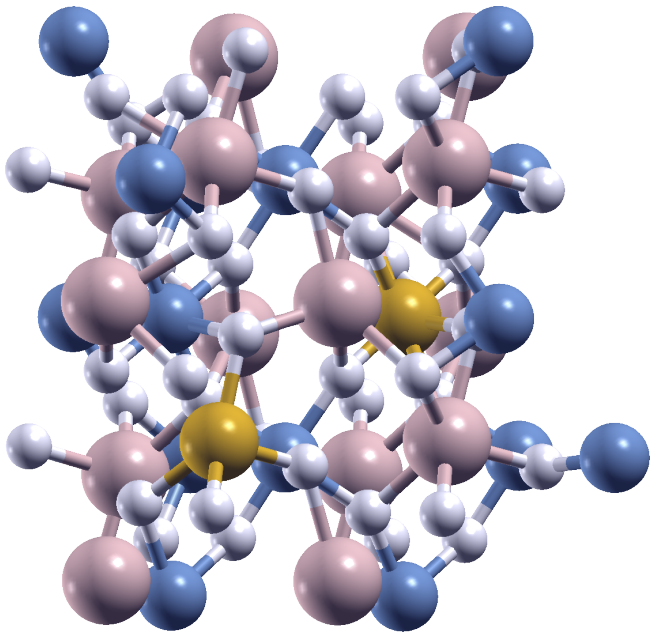
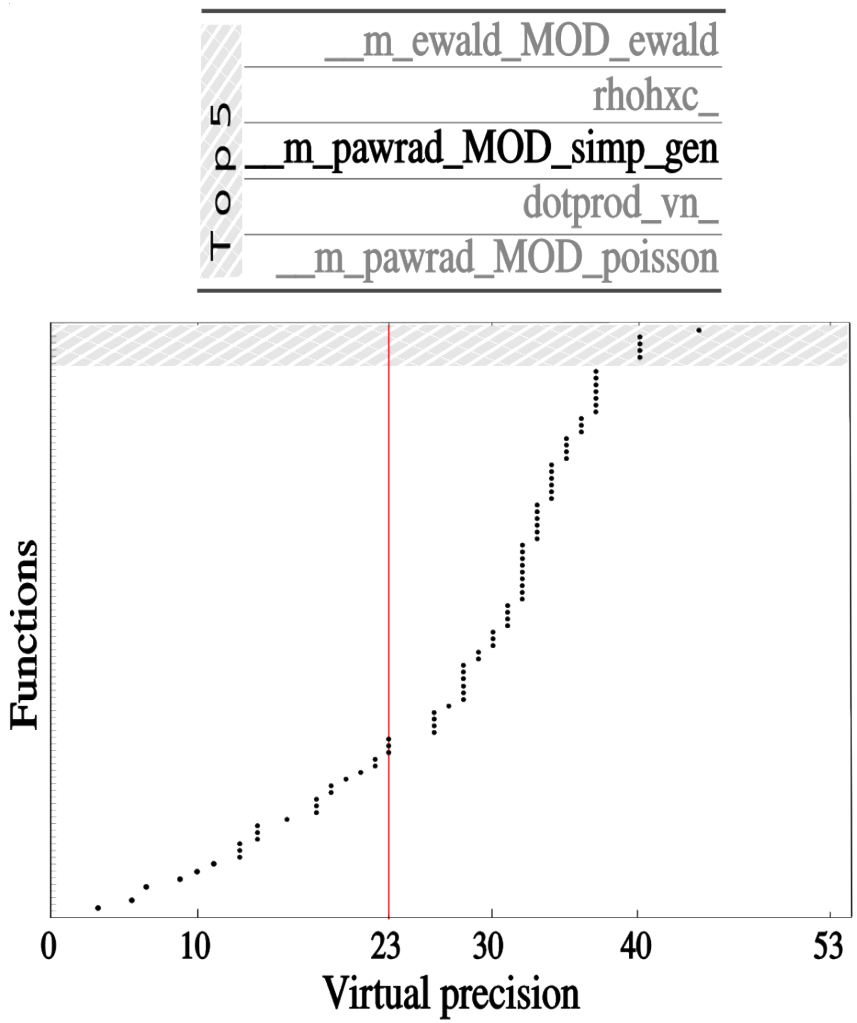


Figure: Sound velocity calculation in an earth mantle component ($MgSiO_3$ perovskyte with Al impurities) [1]

Ordering the search space



Functions below 23bits

- are more resistant
- can potentially be changed to single precision

Functions above 23bits

- are less resistant
- require further analyses

Interesting function among the top 5

`__m_pawrad_MOD_simp_gen`

Simp_gen: Evaluation with Veritracer

Computes an integral by Simpsons' rule over a generalized 1D-grid

- Can be seen as a dot product

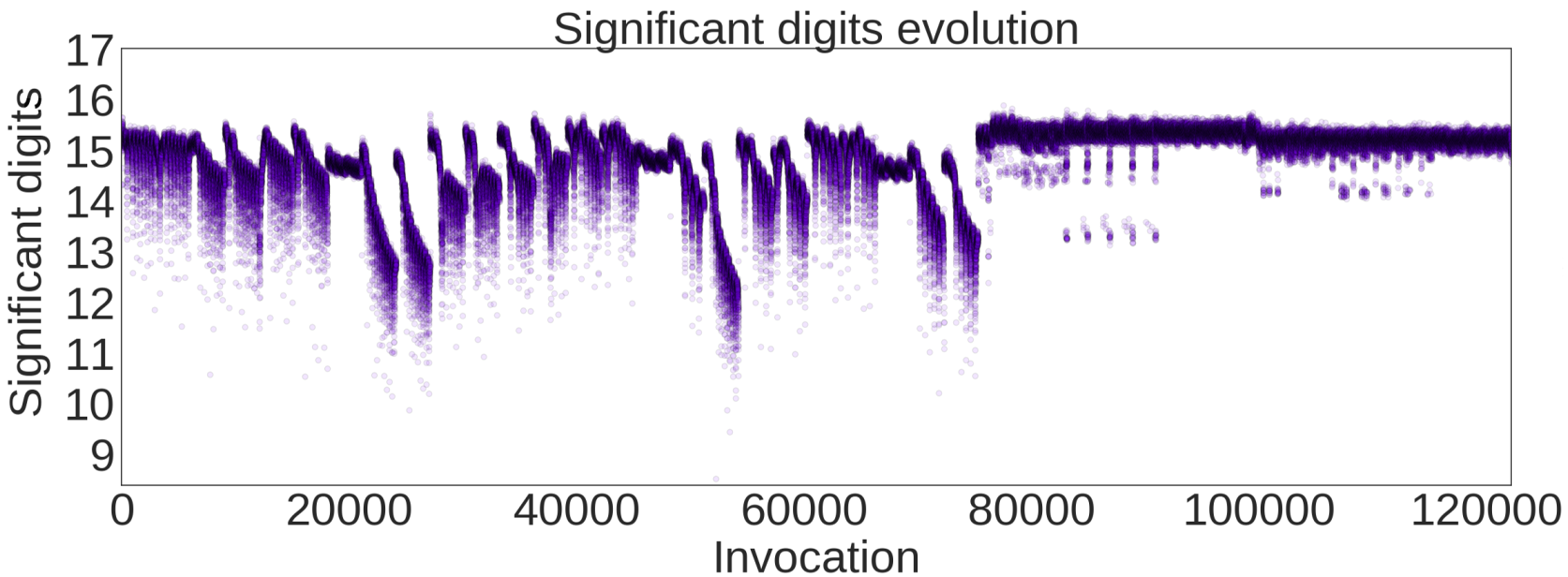
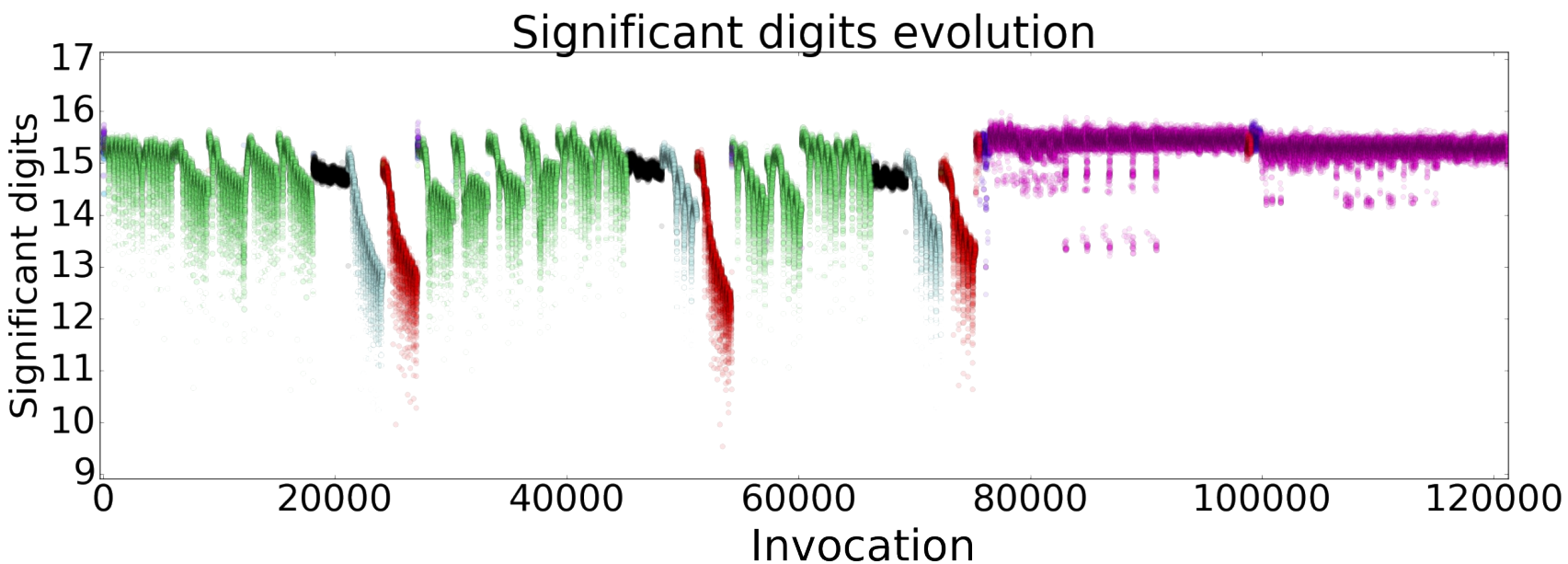


Figure: Evolution of the value returned by simp_gen with $t = 53$ in Random Rounding mode with 24 samples in parallel on the **CINES Occigen** cluster (2106 nodes x2 Intel 12-Cores (E5-2690V3@2.6 GHz))

Simp_gen: the importance of the context analysis

- Colors represent the different CSPs
- Red arrow seems to point an enhancement of the quality ...
- ... but red points come from another context
- Hence the importance of separating contexts



VeriTracer overhead

Instrumentation	Time (s)	$\frac{overhead}{sample}$
Original	201	1
Probes	208	1.03
MCA	236	1.17
VeriTracer (Probes + MCA)	238	1.18

Table: Instrumentation overhead on the Occigen cluster when producing the traces. VeriTracer combines two instrumentations: tracing *Probes*, and replacing FP operations by *MCA* operations.

Verificarlo is able to only instrument a critical function while the rest of the code runs at the full original speed.

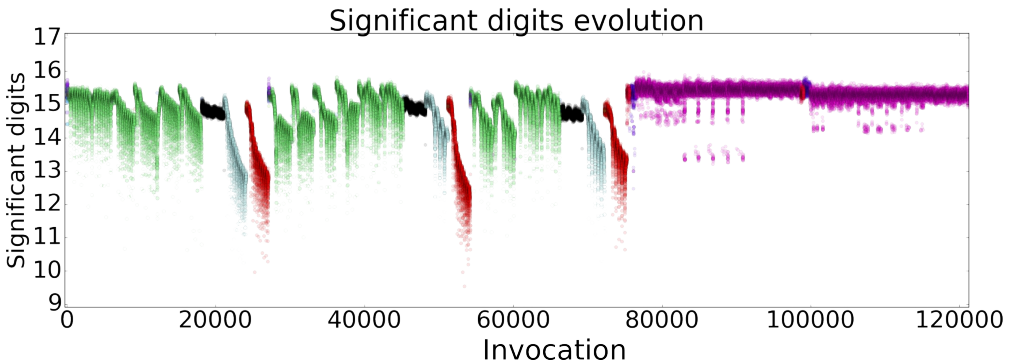
simp_gen code

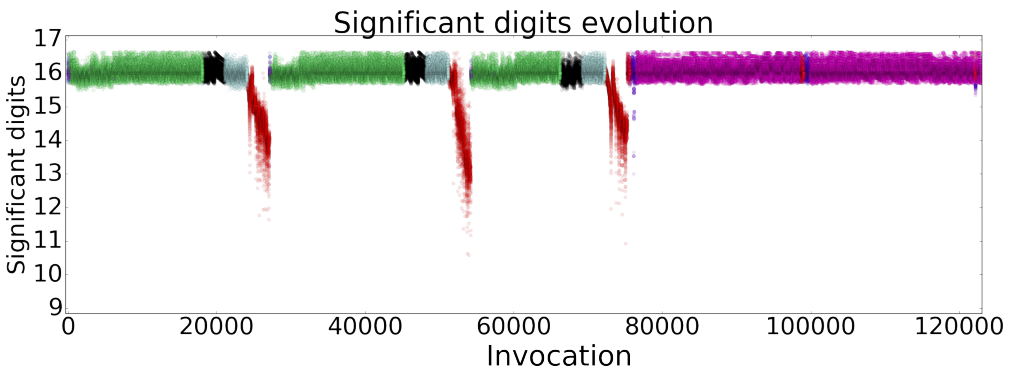
- The main loop can be seen as a dot product
- Can we improve the accuracy ?
- Compensated algorithm: *Dot2*
- Dot product in twice the working precision from Ogita, Rump and Oishi [ogita2005accurate]
- Implemented with `libeft`
  `github.com/ffevotte/libeft`

```
subroutine
simp_gen(intg,func,radmesh)
...
nn=radmesh%int_meshsz
radsim=radmesh%simfact
simp=zero
do i=1,nn
 simp=simp+func(i)*radsim(i)
end do
...
intg=simp+resid
end subroutine
```

```
subroutine
simp_gen(intg,func,radmesh)
...
nn=radmesh%int_meshsz
radsim=radmesh%simfact
simp=zero
call Dot2(simp,func,radsim)
...
intg=simp+resid
end subroutine
```

Simp_gen: A compensated version





Simp_gen: Altered version

- The precision of 30/31 CSPs is improved
- 1 CSP has a low precision due to reentrance of the error

Conclusion

Veritracer is a numerical debugger/optimizer which

- Automatically instruments codes through the verificarlo framework
- Automatically collect trace the numerical quality of variables
- Provides contextual information

Veritracer can be used on real world HPC applications to

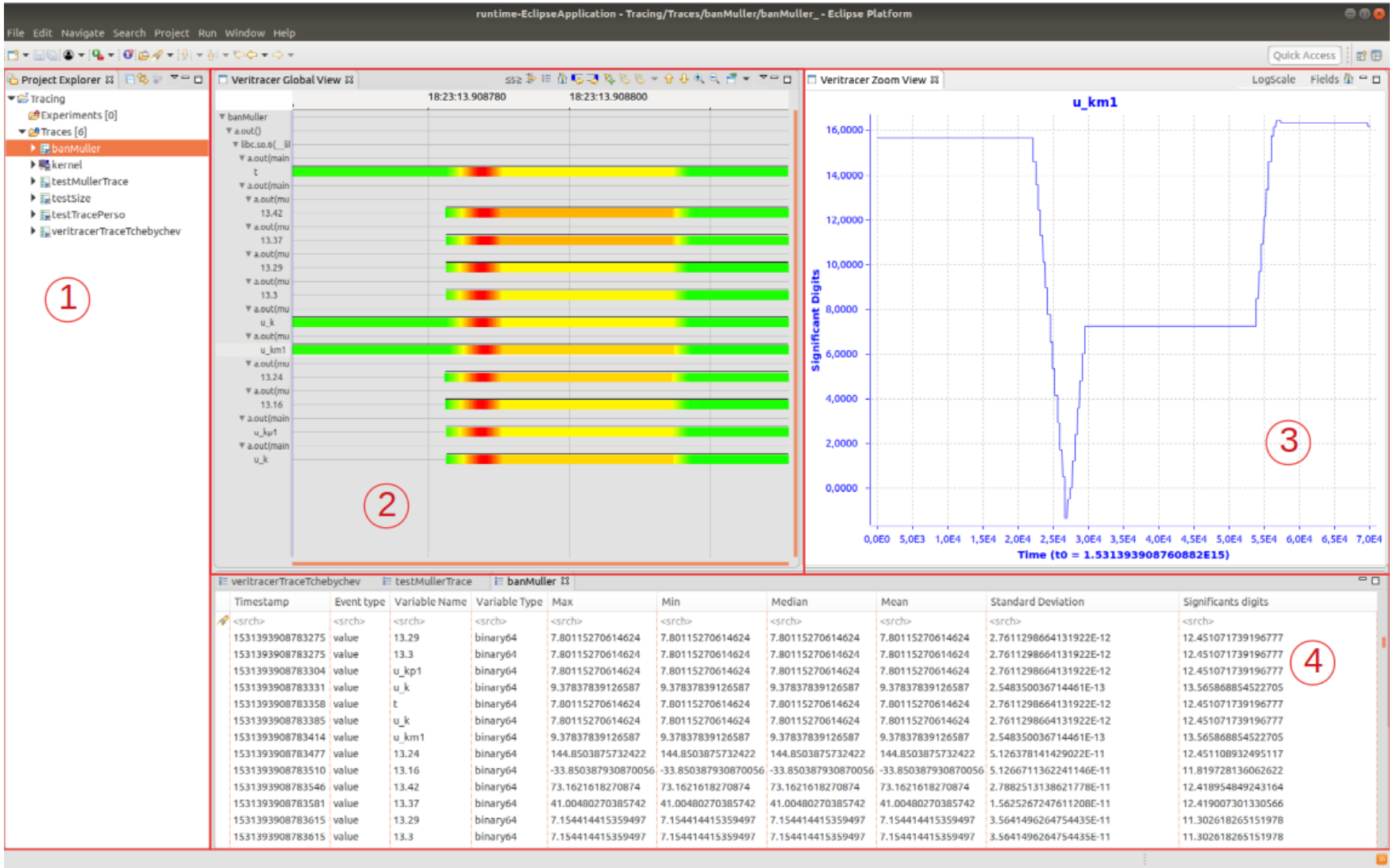
- Detect the numerical sensitivity of functions
- Classify functions calls according to their numerical sensitivity
- Observe the numerical behavior of functions over time

Outline

Veritracer trace visualization

- Veritracer-GUI is a visualization tool for Veritracer traces
- it takes the form of a plugin for the Trace Compass visualization tool used by the LTTng tracing framework.
- The tool comes with a python program using the Babeltrace's python bindings to easily obtain binary traces from Veritracer native traces in the Common Trace Format.

GUI snapshot



Timestamp	Event type	Variable Name	Variable Type	Max	Min	Median	Mean	Standard Deviation	Significants digits
<srch>	<srch>	<srch>	<srch>	<srch>	<srch>	<srch>	<srch>	<srch>	<srch>
1531393908783275	value	13.29	binary64	7.80115270614624	7.80115270614624	7.80115270614624	7.80115270614624	2.7611298664131922E-12	12.451071739196777
1531393908783275	value	13.3	binary64	7.80115270614624	7.80115270614624	7.80115270614624	7.80115270614624	2.7611298664131922E-12	12.451071739196777
1531393908783304	value	u_kp1	binary64	7.80115270614624	7.80115270614624	7.80115270614624	7.80115270614624	2.7611298664131922E-12	12.451071739196777
1531393908783331	value	u_k	binary64	9.37837839126587	9.37837839126587	9.37837839126587	9.37837839126587	2.548350036714461E-13	13.565868854522705
1531393908783358	value	t	binary64	7.80115270614624	7.80115270614624	7.80115270614624	7.80115270614624	2.7611298664131922E-12	12.451071739196777
1531393908783385	value	u_k	binary64	7.80115270614624	7.80115270614624	7.80115270614624	7.80115270614624	2.7611298664131922E-12	12.451071739196777
1531393908783414	value	u_km1	binary64	9.37837839126587	9.37837839126587	9.37837839126587	9.37837839126587	2.548350036714461E-13	13.565868854522705
1531393908783477	value	13.24	binary64	144.8503875732422	144.8503875732422	144.8503875732422	144.8503875732422	5.126378141429022E-11	12.451108932495117
1531393908783510	value	13.16	binary64	-33.850387930870056	-33.850387930870056	-33.850387930870056	-33.850387930870056	5.1266711362241146E-11	11.819728136062622
1531393908783546	value	13.42	binary64	73.1621618270874	73.1621618270874	73.1621618270874	73.1621618270874	2.7882513138621778E-11	12.418954849243164
1531393908783581	value	13.37	binary64	41.00480270385742	41.00480270385742	41.00480270385742	41.00480270385742	1.5625267247611208E-11	12.419007301330566
1531393908783615	value	13.29	binary64	7.154414415359497	7.154414415359497	7.154414415359497	7.154414415359497	3.5641496264754435E-11	11.302618265151978
1531393908783615	value	13.3	binary64	7.154414415359497	7.154414415359497	7.154414415359497	7.154414415359497	3.5641496264754435E-11	11.302618265151978

Veritracer and GUI Demo

...

Outline

Normalization effort and community building

Interflop

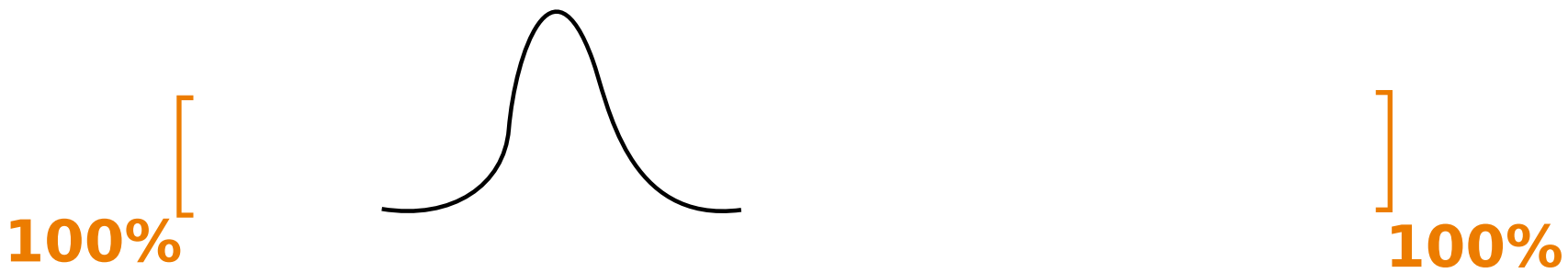
- Open framework for numerical analysis tools
- Initiated by EDF, Intel, and UVSQ
- followed by Lip6 (cadna), CEA (fluctuat), ANEO, Numalis...
- Today, the protoype enables composing Verrou and Verificarlo back-ends (MCA/Random Rounding/...) and front-ends (Valgrind / LLVM instrumentation)
- Unite efforts on processing and analysis tools

`github.com/verificarlo/verificarlo`
`github.com/edf-hpc/verrou`
`github.com/interflop/interflop`

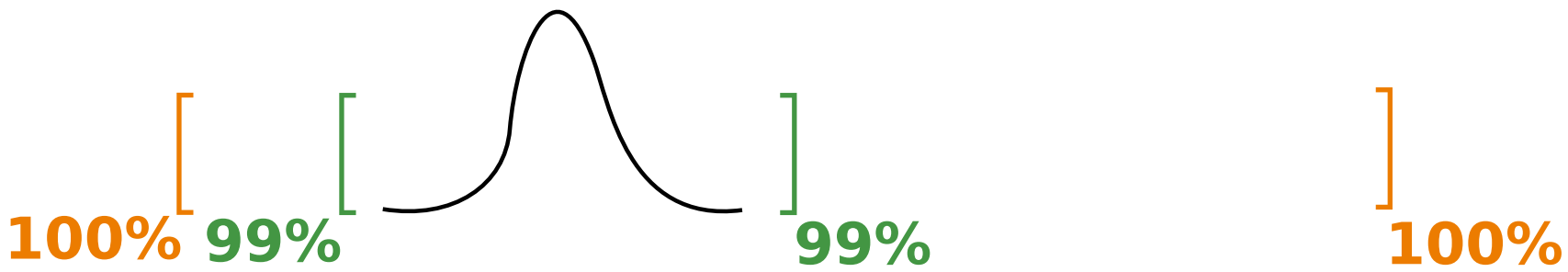
Thank you for your attention

Available on github:
`github.com/verificarlo/verificarlo`
`github.com/verificarlo/verificarlo/tree/veritracer`

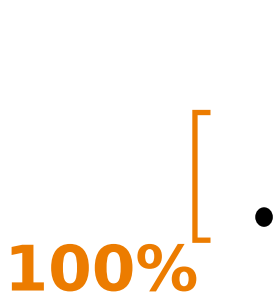
Formal interval vs. Stochastic

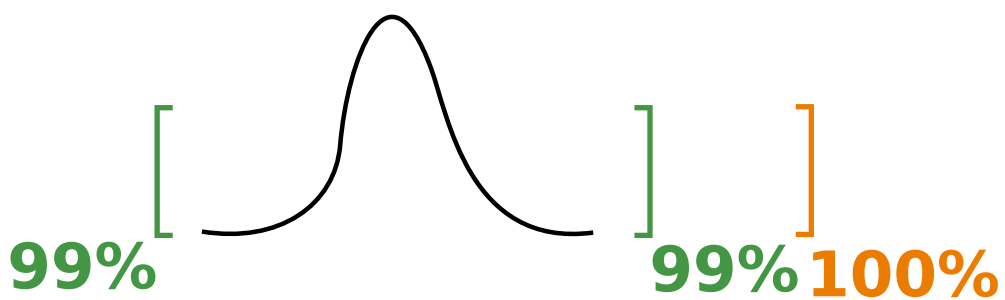


Formal interval vs. Stochastic

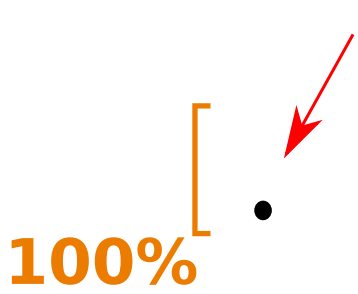


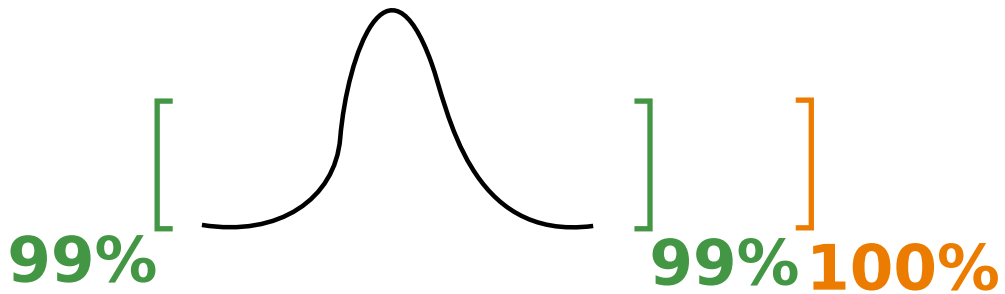
Formal interval vs. Stochastic



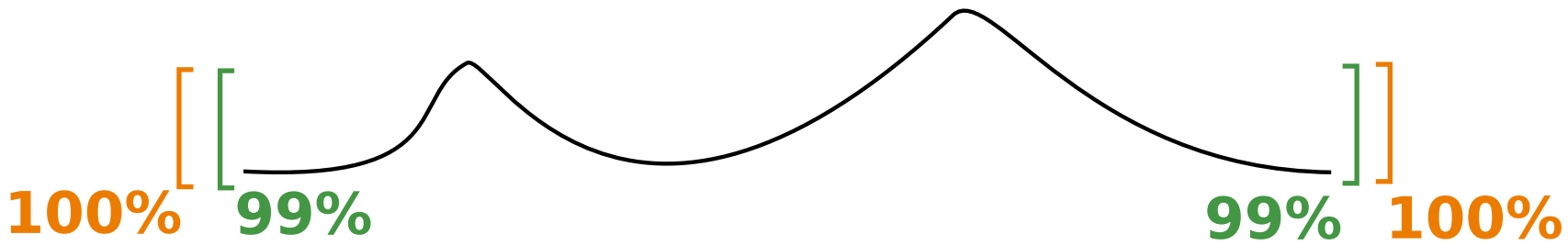


Formal interval vs. Stochastic





Formal interval vs. Stochastic



Code validation for FP errors?

- Two main tools family:
  - Formal
    - (very) limited on coverage, expression, loop, but not a full HPC simulation for all possible inputs... but strictly 100% sure
    - Static analysis, abstract interpretation, interval arithmetic, affine arithmetic...
    - Fluctuat, Daisy, Numalis suite
  - Empirical
    - Can scale to a full application and full-scale use-case but strictly less than 100% sure
    - CESTAC, MCA, extended precision...
    - Verificarlo, Verrou, Cadna, Precimonious, FPdebug, Herbgring, HPC Craft...
- FP arithmetic statiscal verification tools verify the implementation of your model on a given set of experiment with a given confidency

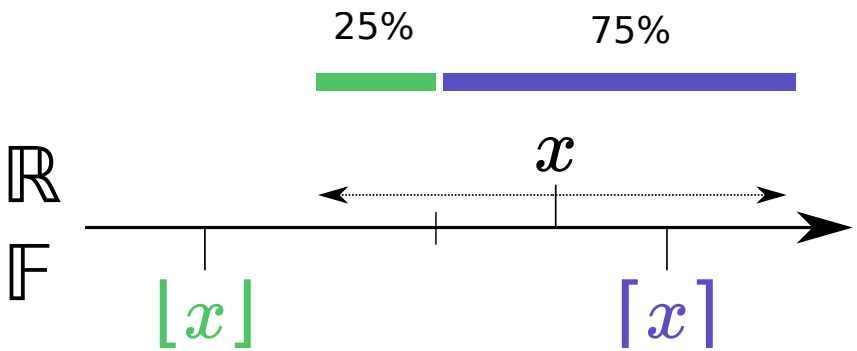
Existing tools

	Method	Name	Implementation
Debugging	Stochastic Arithmetic	CADNA [8] VERROU [5] Verificarlo [4]	CESTAC/DSA (library) CESTAC (Valgrind) MCA (LLVM)
	Extended Precision (EP)	HPC Craft [9] FpDebug [3] Herbgrind [12]	Exponent comparison (DynInst) MPFR (Valgrind) MPFR (Valgrind)

	Name	Method	Mixed-Precision	Any-Precision
Optimization	Verificarlo [4]	MCA (User specific)	✓	✓
	HPC Craft [9]	Bitmask (ref value)	✓	✓
	Promise [7]	CESTAC/DSA ($\Delta - \textit{debug}$)	✓	
	Precimonious [11]	EP ($\Delta - \textit{debug}$)	✓	
	Herbie [10]	EP (Rewriting)		

MCA Random Rounding at the ulp (1/2)

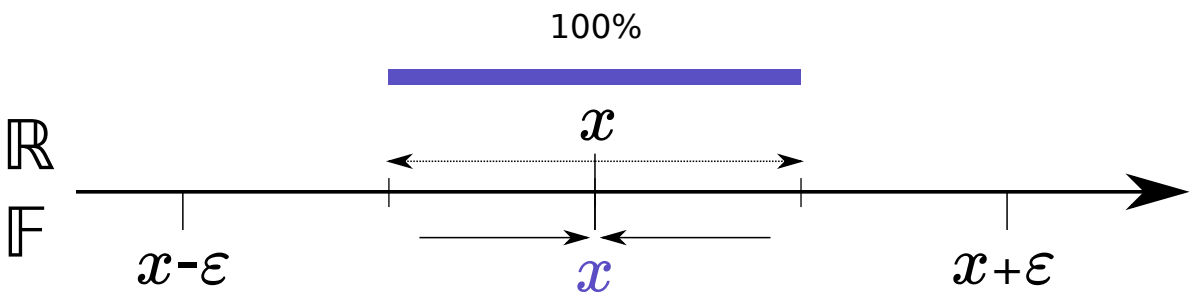
- t=24 for `float` and t=53 for `double` is a special case: the virtual precision corresponds to a one ulp ϵ error.
- The random error introduced is in $]-\frac{\epsilon}{2}, \frac{\epsilon}{2}[$.
- MCA result is either the downwards or upwards roundoff, with a probability proportional to $\frac{\lceil x \rceil - x}{\epsilon}$.



- This mode is equivalent to Verrou's *average* random rounding.

MCA Random Rounding at the ulp (2/2)

- t=24 for `float` and t=53 for `double` is a special case: the virtual precision corresponds to a one ulp ϵ error.
- The random error introduced is in $]-\frac{\epsilon}{2}, \frac{\epsilon}{2}[$.
- For exact values no error is introduced in this mode.



Legal Notices and Disclaimers

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Configurations: see each performance slide notes for configurations. For more information go to http://www.intel.com/performance/datacenter.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Relative performance is calculated by assigning a baseline value of 1.0 to one benchmark result, and then dividing the actual benchmark result for the baseline platform into each of the specific benchmark results of each of the other platforms, and assigning them a relative performance number that correlates with the performance improvements reported. SPEC and SPEC MPI* are trademarks of the Standard Performance Evaluation Corporation. See http://www.spec.org for more information.

Intel processors of the same SKU may vary in frequency or power as a result of natural variability in the production process.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at Intel.com

Intel products may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

References I

[1] ABINIT.
Silicon carbide, 2016.
[Online; accessed May 15, 2018].

[2] J.-C. Bajard, D. Michelucci, J.-M. Moreau, and J.-M. Muller.
Introduction to the Special Issue "Real Numbers and Computers".
In *The Journal of Universal Computer Science*, pages 436–438.
Springer, 1996.

[3] F. Benz, A. Hildebrandt, and S. Hack.
A dynamic program analysis to find floating-point accuracy problems.
ACM SIGPLAN, 47(6):453–462, 2012.

References II

[4] C. Denis, P. de Oliveira Castro, and E. Petit.
Verificarlo: checking floating point accuracy through monte carlo arithmetic.
In *Computer Arithmetic (ARITH), 23nd Symposium on*, pages 55–62. IEEE, 2016.

[5] F. Févotte and B. Lathuilière.
VERROU: a CESTAC evaluation without recompilation.
SCAN 2016, page 47, 2016.

[6] X. Gonze, F. Jollet, et al.
Recent developments in the ABINIT software package.
Computer Physics Communications, 205:106–131, 2016.

References III

[7] S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière.
Auto-tuning for floating-point precision with Discrete Stochastic Arithmetic.
working paper or preprint, June 2016.

[8] F. Jézéquel and J.-M. Chesneaux.
CADNA: a library for estimating round-off error propagation.
Computer Physics Communications, 178(12), 2008.

[9] M. O. Lam, J. K. Hollingsworth, and G. Stewart.
Dynamic floating-point cancellation detection.
Parallel Computing, 39(3):146–155, 2013.

References IV

[10] P. Panchekha, A. Sanchez-Stern, J. R. Wilcox, and Z. Tatlock.
Automatically improving accuracy for floating point expressions.
In *ACM SIGPLAN Notices*, volume 50, pages 1–11. ACM, 2015.

[11] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough.
Precimonious: Tuning assistant for floating-point precision.
In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 27. ACM, 2013.

[12] A. Sanchez-Stern, P. Panchekha, S. Lerner, and Z. Tatlock.
Finding root causes of floating point error with herbgrind.
arXiv preprint arXiv:1705.10416, 2017.

References V

[13] D. Sohier, P. De Oliveira Castro, F. Févotte, B. Lathuilière, E. Petit, and O. Jamond.
Confidence Intervals for Stochastic Arithmetic.
working paper or preprint, July 2018.