Numerical Computing Validation on New Architecture and Large Scale Systems

Eric Petit; Pablo Oliveira; Yohan Chatelain;
Bruno Lathuliere; Francois Fevotte

Intel-DCG-IPAG; UVSQ; ECR Lab.; EDF Lab.

2018-01-23 – ETSN 2018 Lecture on Verrou and Verificarlo – Cargese

Foreword about ECR Lab. at Teratec

CEA DAM, Intel, UVSQ

Since 2010, around 10 people working on MAQAO, MPC, ASK, Cere, DClib, Titus, Verificarlo, MALT...

And optimizing many workloads from CEA and other institute and industrials: QMC=Chem, Abinit, Polaris, EPX, Yales2, AVBP, Polaris, Total RTM,...

Objective: How to build applications for the next generations of Intel systems?

- Application-architecture co-design
- Scalable and parallel efficient algorithms
- Scalable performance tools
- Other challenges arising from code modernization for highly parallel systems: numerical precision, programmability, scalable runtime, benchmarking...

Table of contents

Section 1

The Evolution of Modern Architecture and its Challenges for the Software Stack

Intel innovations for HPC systems

Anyone can build a fast CPU. The trick is to build a fast system.
Seymour Cray

And I would add: which efficiently compute the expected results!

Architectural evolution cycles

- Multiple reasons in the architecture environment can be challenging for the precision/reproducibility/determinim of your computation
- A system is not just a computing unit, and it is NOT generally deterministic!
  - runtimes
  - memory alignement
  - parallel algorithm
  - implicit/explicit hw level parallelism
  - compiler optimization
  - asynchronousity...
- This makes the numerical precision a recurrent hot topic everytime a major change is happening on the hw side and propagating in the full system.
  - vector instruction set, multi-core, manycore, FMA...

Intel architecture drivers and the interweaved HPC and AI systems evolution

- For a long time performance has been by far the main driver,
- And HPC was ahead, paving the way of future consumer technology
- Support market were and still are commodity datacenters, PC, and videogames.
- However new concerns and new market are arising
  - Memory and I/O often are the main bottleneck (not new)
  - $\alpha.Flop/(\beta.W.\gamma.\$)$ is the main metric to tune
  - Cloud infrastructure and ML/DL workloads (embedded or not) are becoming the most influential for processor, memory and network (wire and wireless) technologies

  $\rightarrow$ New trade-off are arising for processor/memory organization, productivity, and FLOP operators and their accuracy

BTW, 2 common shortcuts in HPC about AI and its influence on compute ressources

- DL framework relies on large FP16 throughput?
  - WRONG
  - Training and inference are very different steps:
    - while training some operation can work on lower precision or range (flexpoint, fixed point), others welcome DP. Througput-oriented specialized design (ASIC) are welcome (Nervana, TPU) with high-throughput large-memory hierarchy (Intel NVME, HBM)
    - inference is low precision (a few bits) and less compute intensive. Latency matters as much as throughput and power consumption: FPGA, ASIC, SoC (nervana, movidius, mobileye, Altera)
    - Intel has strong assets in all aspect of ML that already percolate in both direction
- We will have to compute on AI plateform?
  - WRONG
  - For Intel hardware, there is two separate roadmaps, and the one for HPC is today closer to the cloud computing one. But there will be probably mutual fertilization.
  - For software, I cannot see any convergence in a near future, but there is great idea on both side that can benefit us all.

Issue with implementing FP codes on a moving base

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

- How to assess correctness of a result? or validate an implementation?
- Ensuring the numerical reproducibility is not always a requirement!
  - Despite most the (HPC) users want to be conservative...
- Does different results means wrong results?

Section 2

About numerical stability from the floating point evaluation perspective

Defining the correctness of a result?

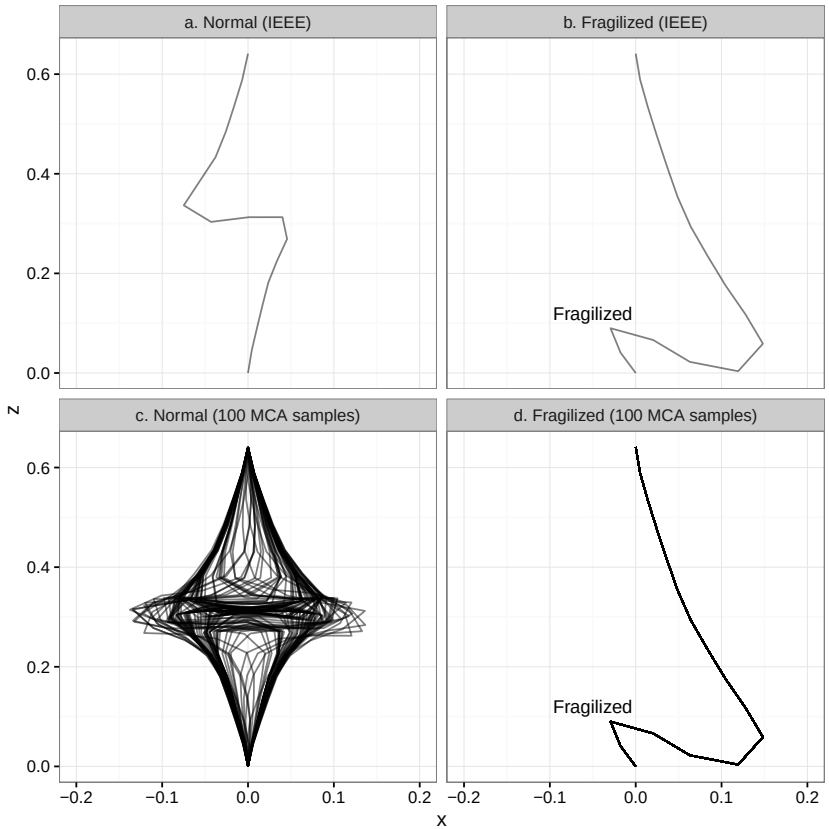


Figure 1: Buckling simulation of a 1D beam with EPX

From the physics to the compute units

1. Physical phenomena

→ Model error

2. Mathematical model

→ Discretization error

3. Numerical Algorithm

→ Computation error

4. Implementation
5. Compilation
6. Execution

- Verification: the computation match the model
- Validation: the computation match the physics

Code validation for FP errors?

- It does not validate the model, but its implementation!
- 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, interval arithmetic, extended precision
    - Verificarlo, Verrou, Cadna, Precimonious, FPdebug...
- Is statistical FP implementation debugging and validation enough for you?
  - Unitary test coverage?
  - On my use case, I am 95% sure that:
    - (The model predict that) My nuclear reactor will not melt.
    - My search engine is giving accurate results.
    - Is this enough? ;)
- FP arithmetic statiscal validation tools validate the implementation of your model on a given set of experiment with a given confidency

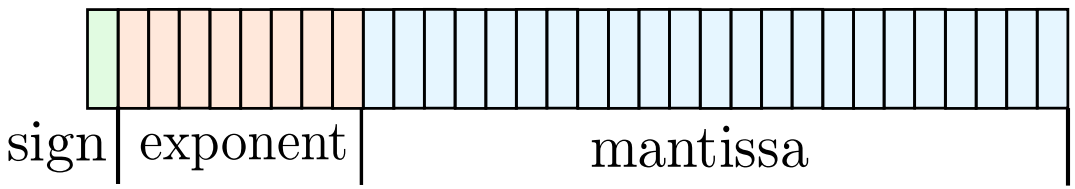
Floating point computation: the IEEE-754 standard

- 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 (32 bits), Double (64 bits)...
- And four rounding modes :
  - nearest, toward $+\infty$, toward $-\infty$, toward zero

Floating-point representation

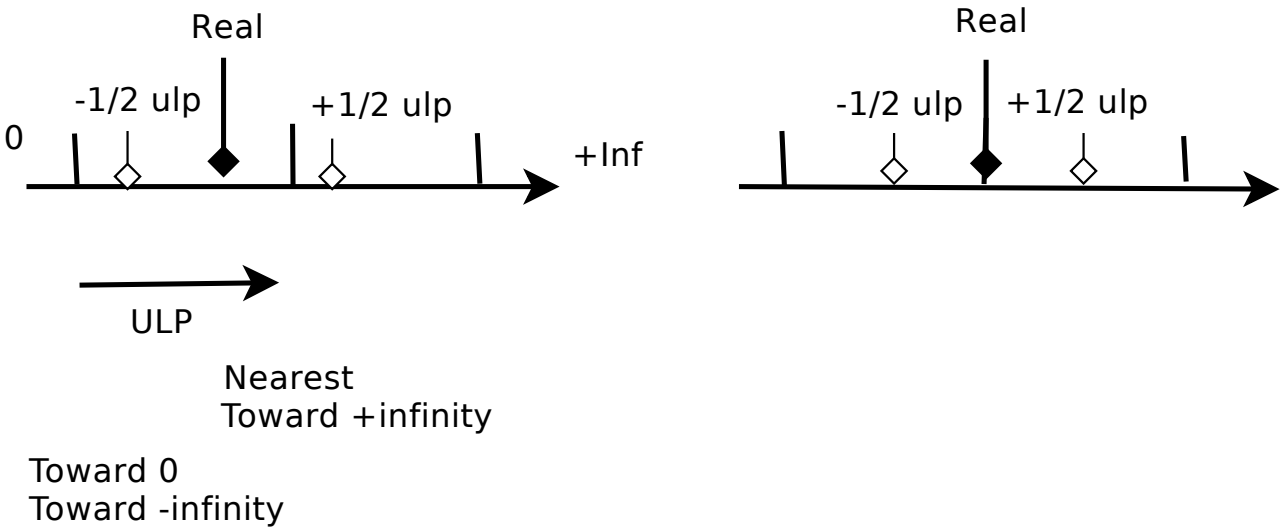
IEEE-754 Single Precision 32-bit

$$f = (-1)^{sign} \times 2^{exponent-127} \times 1.M \, , M = \sum_{i=1}^{23} 2^{-i}$$



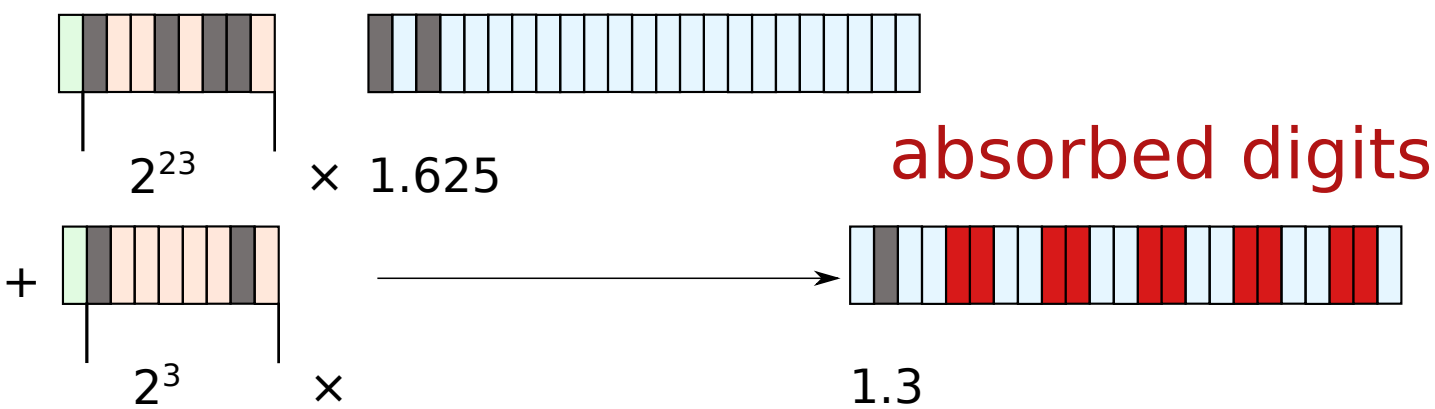
Precision	sign	exponent	mantissa	Total bits
Single	1	8	23	32
Double	1	11	52	64

Rounding modes



Floating point computation: some adverse effects

- A floating point computation approximates the real computation
  - Representation errors
    - 3.14159265359300
  - 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.
    - 3.14159+0.00141421=3.14300{421}
  - Cancellation, relative error when subtracting variables with very close values
    - 3.14300-3.14159=0.00141



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...

Reproducibility, accuracy, and precision

- Quoting J. Demmel et al. about reproBLAS:
  Reproducibility : obtaining bit-wise identical results from different runs of the program on the same input data, regardless of different available resources.
  Reproducibility is needed for debugging and for understanding the reliability of the program.
- Architecture, mem layout, nb process, threads, vector size...
- Reproducible $\sim$ strictly deterministic
- And (hopefully) many physicists/numericians doesn't have the same definition for their code:
  - Reproducible = the significant part of the meaningful result are similar across platform and architecture
  - Use-case dependent, only the scientist can tell.
    - Conservation of energy, flow speed on the output, quantity of pollutant, total absorbed energy, maximum deformation...

Does it always make sense to be reproducible?

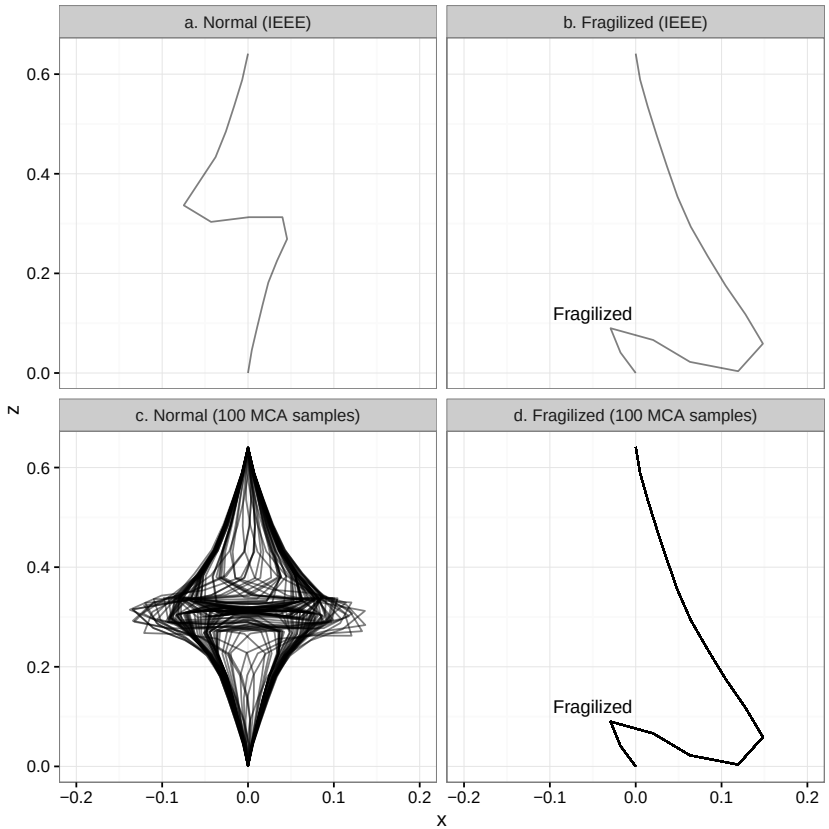


Figure 2: Buckling simulation of a 1D beam with EPX

Precision, accuracy and reproducibility, which one do you really want?

- Reproducible and/or accurate operators and libraries exists for many operations (extension and improvement are still an active domain).
  - Specific hardware such has Kulish accumulator are largely explored. Will it be implemented? Where?
  - Trend (DL..) is more to explore lower precision and accuracy hw operators.
  - Different techniques: extended precision, compensation, formula rewriting...
  - Libraries: reproblas, mpfr, boost, libeft...
- Accurate $\Rightarrow$precise and reproducible
- But the opposite is not true! *e.g.* See J.M Muller example in the following verificarlo part of the presentation

$\Rightarrow$ You can be precisely wrong, and you can reproduce it ;)

BTW is this accurate?

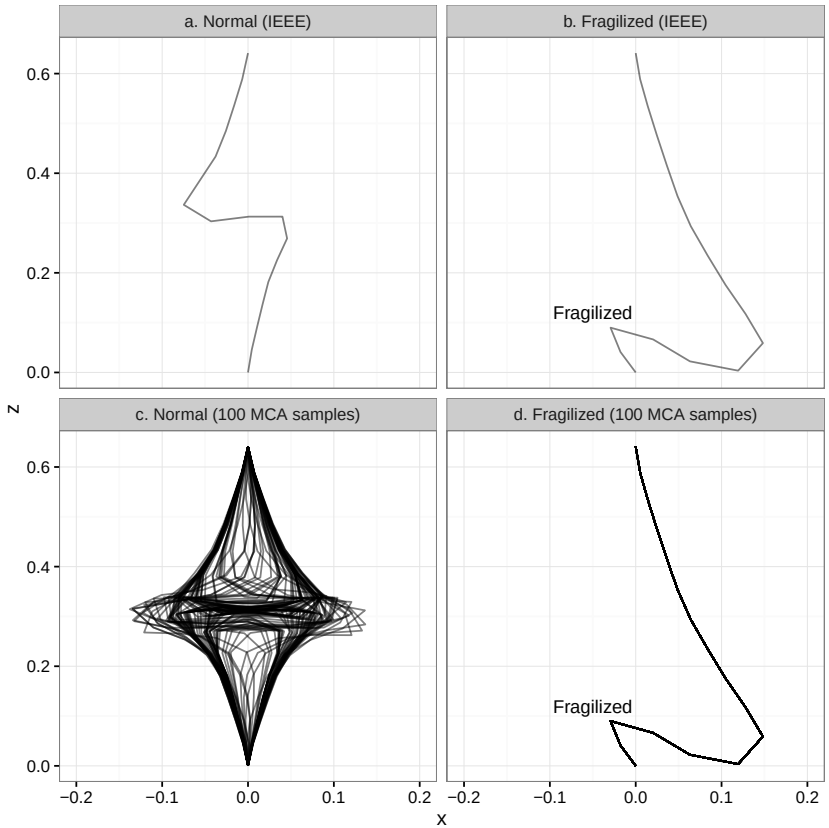


Figure 3: Buckling simulation of a 1D beam with EPX

Motivating example: Numerical bugs in Abinit

- In 2016 a first bug was solved by T. Guillet (Intel)
  - With the vectorization enabled, the code was diverging
  - The inaccuracy in an integral computation (summation) was giving slightly different results among the MPI ranks
  - This differences were introducing new "bad" eigenvalues to the problem definition
  - This has been solved using accurate summation algorithm, or locally disabling the vectorization
- Quite recently a similar bug appears
  - A well define use cases prepared for a grand Challenge on Oxygen2 at CINES refuse to converge
  - The convergence oscillate around a constant residual
  - Again switching off the vectorization solved the issue
  - ...but degrade the performance...

Target objectives

- Can we spot and fix these numerical bug in systematic way?
- Can we prepare the portability on future systems?
- Can we elaborate unitary test for non regression?
- Can we optimize while reducing locally the accuracy? (mixed precision)
  - Without computing the exact theoretical answer
  - On a whole full scale scientific code
  - Without modifying the application code
  - And taking into account compiler optimization and special instructions
  - And ultimately, can we, computer arithmetic expert, be replaced by some scripts to have the programmer doing it himself? ;)

Section 3

Verificarlo: Checking Floating-Point Accuracy Through Monte Carlo Arithmetic

Example by W. Kahan, condition number 2.497e8

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

- Results obtained using the LAPACK routines

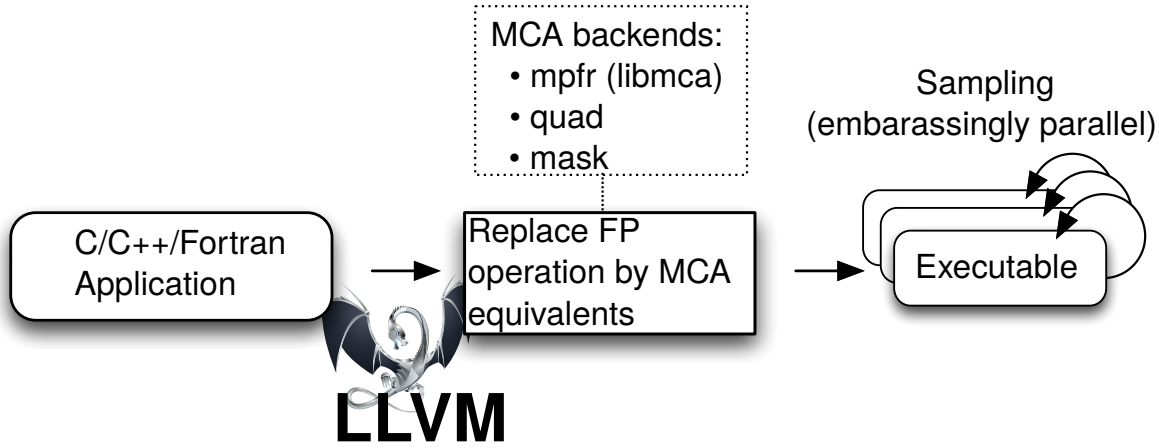
$$x_{single} = \begin{pmatrix} 1.33317912 \\ -1.00000000 \end{pmatrix} x_{double} = \begin{pmatrix} 2.0000000\mathbf{240030218} \\ -2.0000000\mathbf{359962060} \end{pmatrix}$$

- 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

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`
  - Contributors: Pablo Oliveira (UVSQ), Yohan Chatelain (UVSQ), Eric Petit (Intel) and Christophe Denis (CMLA)

Verificarlo



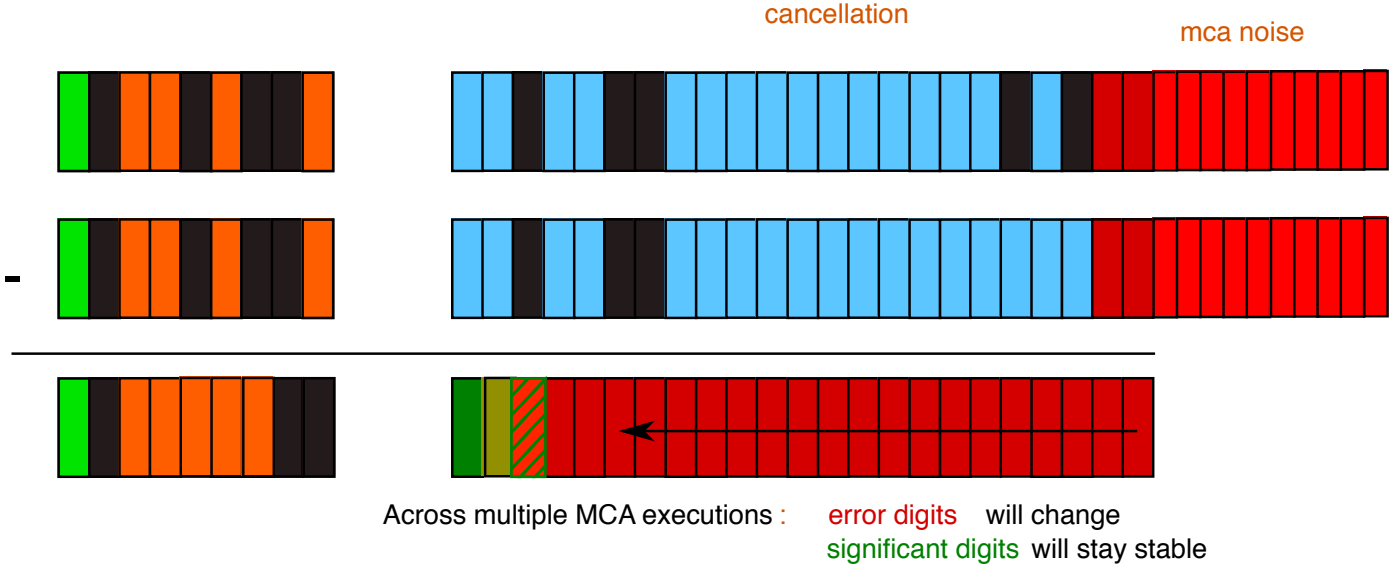
- Transparently transforms code to Monte Carlo Arithmetic
- Operates on optimized code: evaluates floating point errors introduced by compiler optimizations

```
-   %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)
```

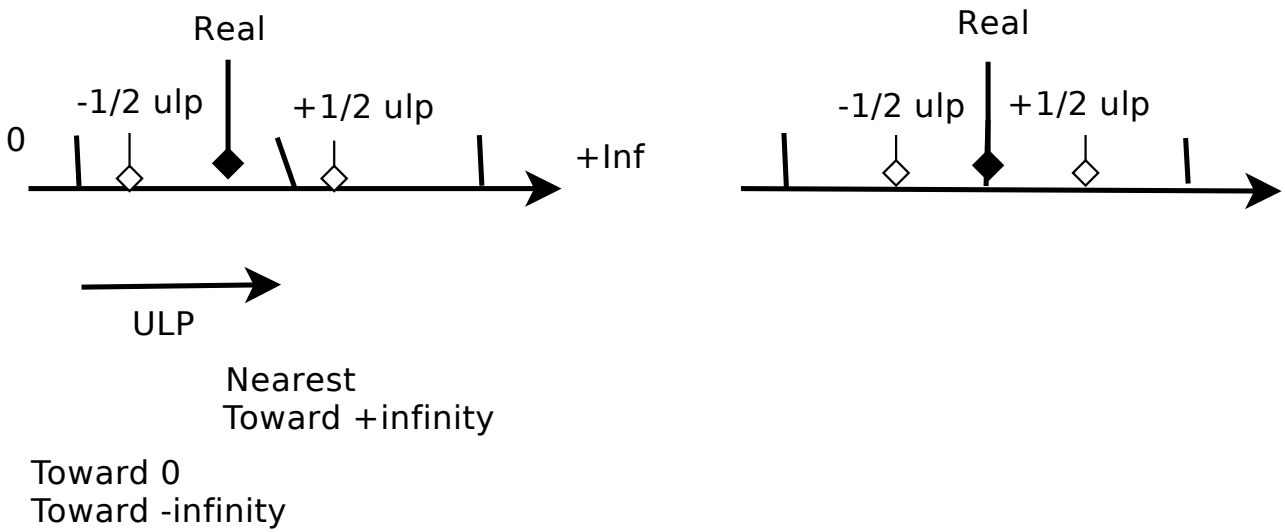
Monte Carlo Arithmetic: models FP Error Propagation [S. Parker, 1999]

- Verificarlo estimate the number of significant digits by using N Monte Carlo samples $\tilde{s} = -\log_{10}\left(\frac{\tilde{\sigma}}{\tilde{\mu}}\right)$
  - $\tilde{\mu}$: empirical mean and $\tilde{\sigma}$: empirical standard deviation

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



Random Rounding = MCA 53/27



Compiler optimizations are instrumented

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

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

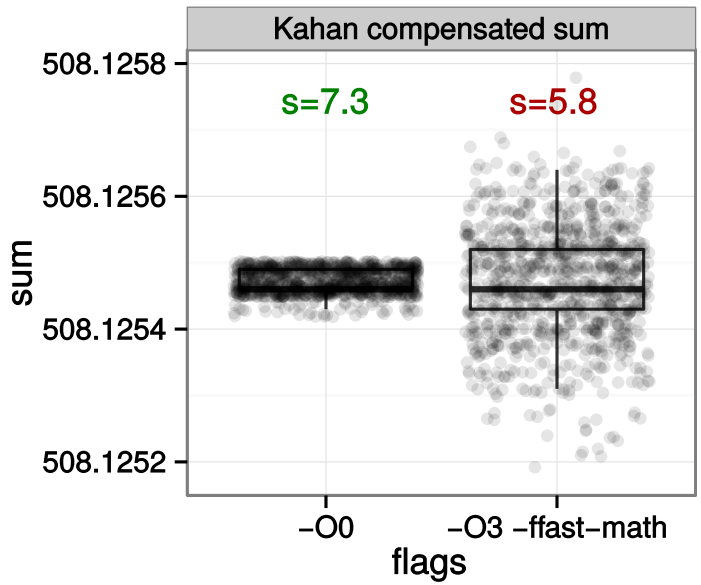


Figure 4: Analysis of the effect of compiler flags on a Kahan compensated sum algorithm (Random Rounding with p=53)

Example by W. Kahan, condition number 2.497e8

- Computations using IEEE-754 FP numbers

Precision	Result	s
SP	$x(1) = 1.33317912$	0
	$x(2) = -1.00000000$	0
DP	$x(1) = 2.00000000240030218$	9
	$x(2) = 2.00000000359962060$	9

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

Precision	$\hat{\mu}$	$\hat{\sigma}$	$\hat{s}$
Verificarlo SP	$\hat{\mu}(x(1)) = 1.02463705$	$\hat{\sigma}(x(1)) = 6.4...$	0.0
	$\hat{\mu}(x(2)) = 6.46717332$	$\hat{\sigma}(x(2)) = 9.6...$	0.0
Verificarlo DP	$\hat{\mu}(x(1)) = 1.9999999992$	$\hat{\sigma}(x(1)) = 8.4... \times 10^{-9}$	8.3
	$\hat{\mu}(x(2)) = -1.9999999988$	$\hat{\sigma}(x(2)) = 1.2... \times 10^{-8}$	8.2

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

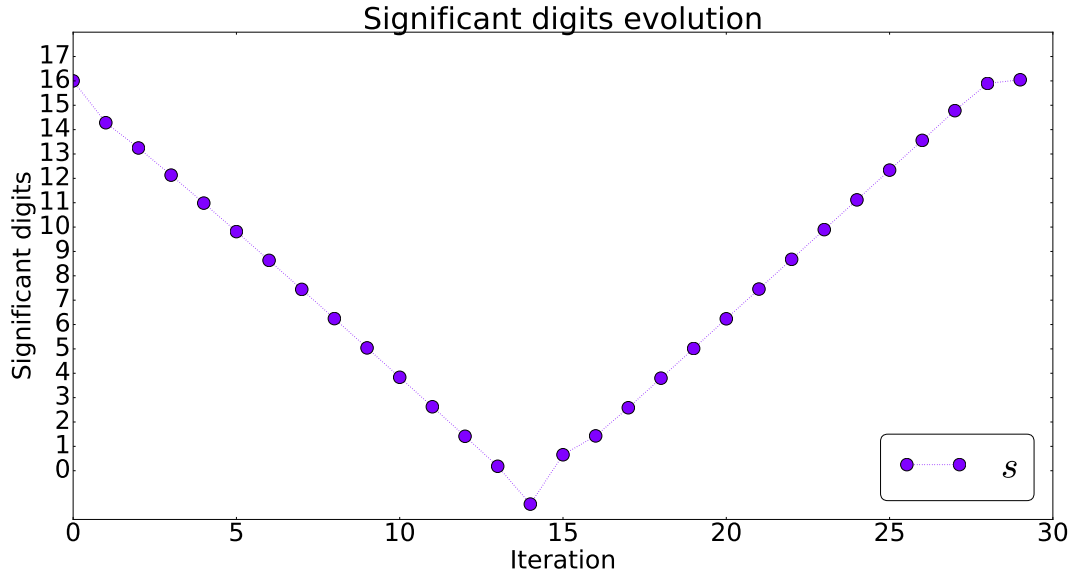
Tracing precision

Computing the numerical limit of the following sequence:

$$u_{n+1} = 111 - \frac{1130}{u_n} + \frac{3000}{u_n u_{n-1}}$$
$$with \quad u_0 = 1, u_1 = -4$$

- The accurate result is 6
- Whatever the precision, a computer will answer 100

VeriTracer



- At $n = 30$, $s = 16 \Rightarrow$ computation is accurate ?
- for $n = 14$, $s < 0 \Rightarrow u_{14}$ has no correct decimal digits, *ie* you can introduce any random number instead of your variable at that point!
- Only checking the final results is not enough to detect the accuracy loss.
- Most of FP analysis tools will conclude on a fully precise result.
- Veritracer allow to trace the precision and highlight the problem
- That is an example of 'precisely wrong' ;)

VeriTracer: context-enriched precision tracer

- Our last paper to be published at arith25, release planned Q3 2018.
- Tracing the precision and contextual information of the FP computation
- A function can be called by different parents hierarchy: the *Call Site Path* (CSP)
- Depending on its CSP, a function may have different behaviors
- Tracing CSP avoid possible false correlations in the analysis

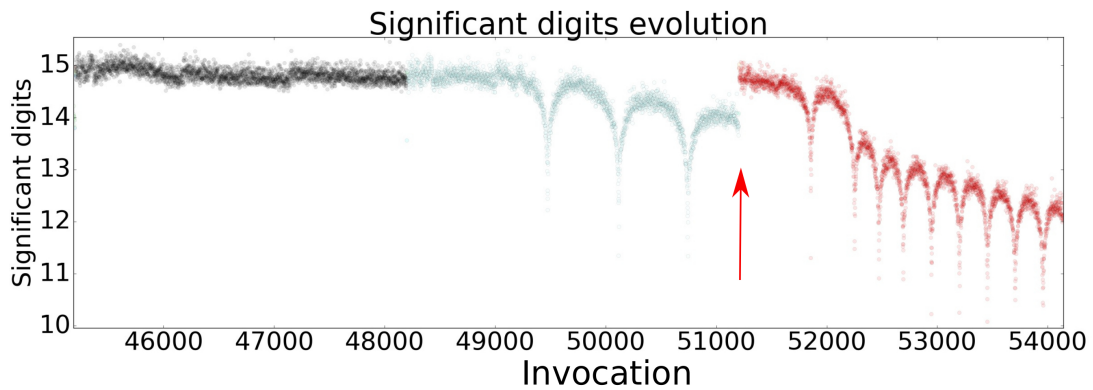


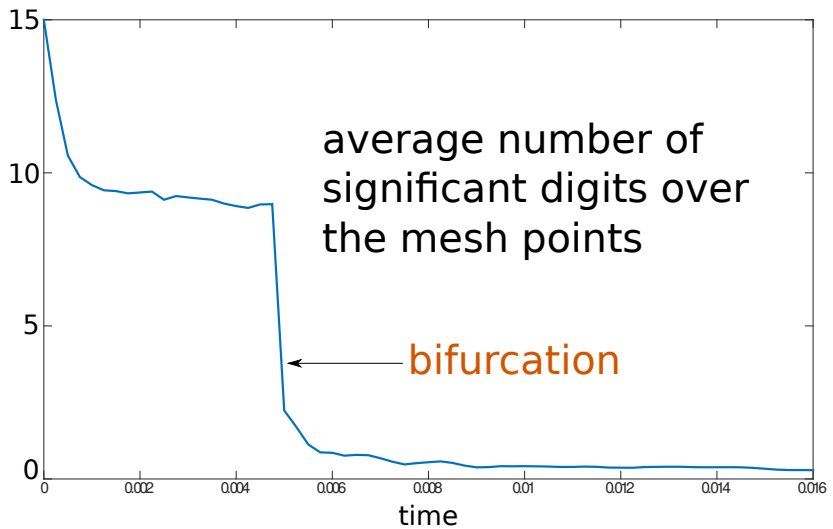
Figure 5: Coloring Call Sites Path to differentiate call contexts. Numerical enhancement pinpoints by red arrow is an illusion since blue points and red points do not belong to the same call site path

Section 4

Some Verificarlo Results

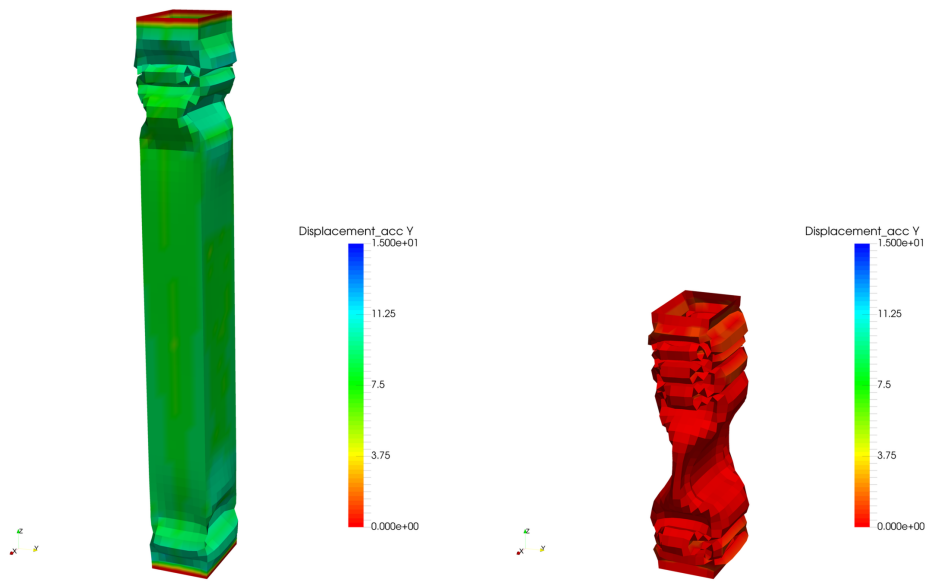
Study on Europlexus (EPX)

- 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
- Users complain about non reproducibility across hardware platform
- Precision on a rectangular beam buckling experiment:



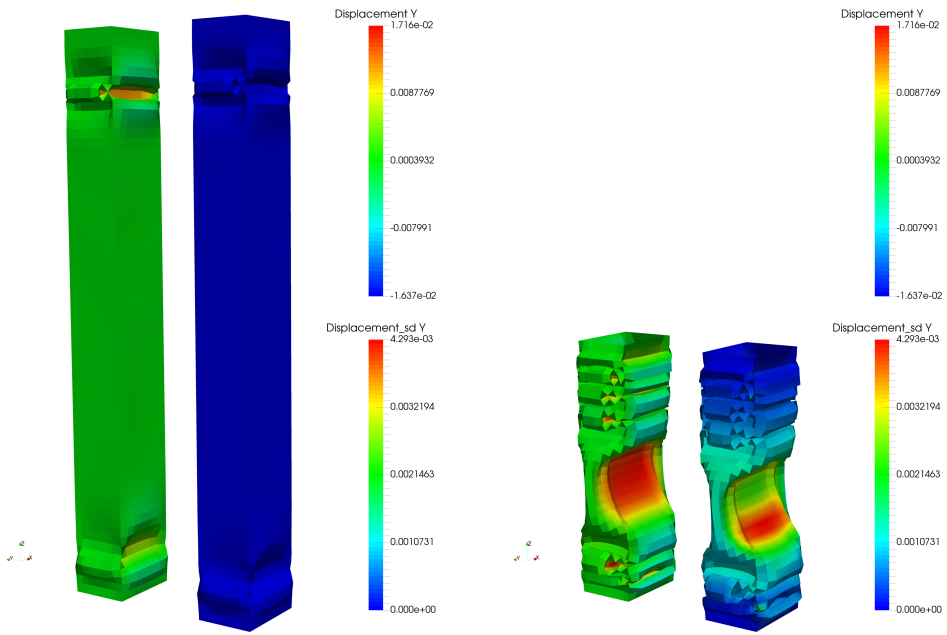
VTK plugin for Europlexus - covariance

- Europlexus dynamic buckling of a rectangular section beam
- Visualized using the Verificarlo VTK post-process plugin
- Left side is the initial ripples after impact at $t = 4.25ms$
- The right side shows the end of the simulation at $t = 16ms$.
- At $t = 4.25ms$ the number of significant digits is high ($s > 8$).
- At $t = 16ms$ the precision is very low since there are no significant digits.



VTK plugin for Europlexus - standard deviation

- Europlexus dynamic buckling of a rectangular section beam
- Visualized using the Verificarlo VTK post-process plugin
- The left beam is the displacement
- The right beam is its empirical standard deviation
- The standard deviation is low compare to the displacement even if no significant digits remains.

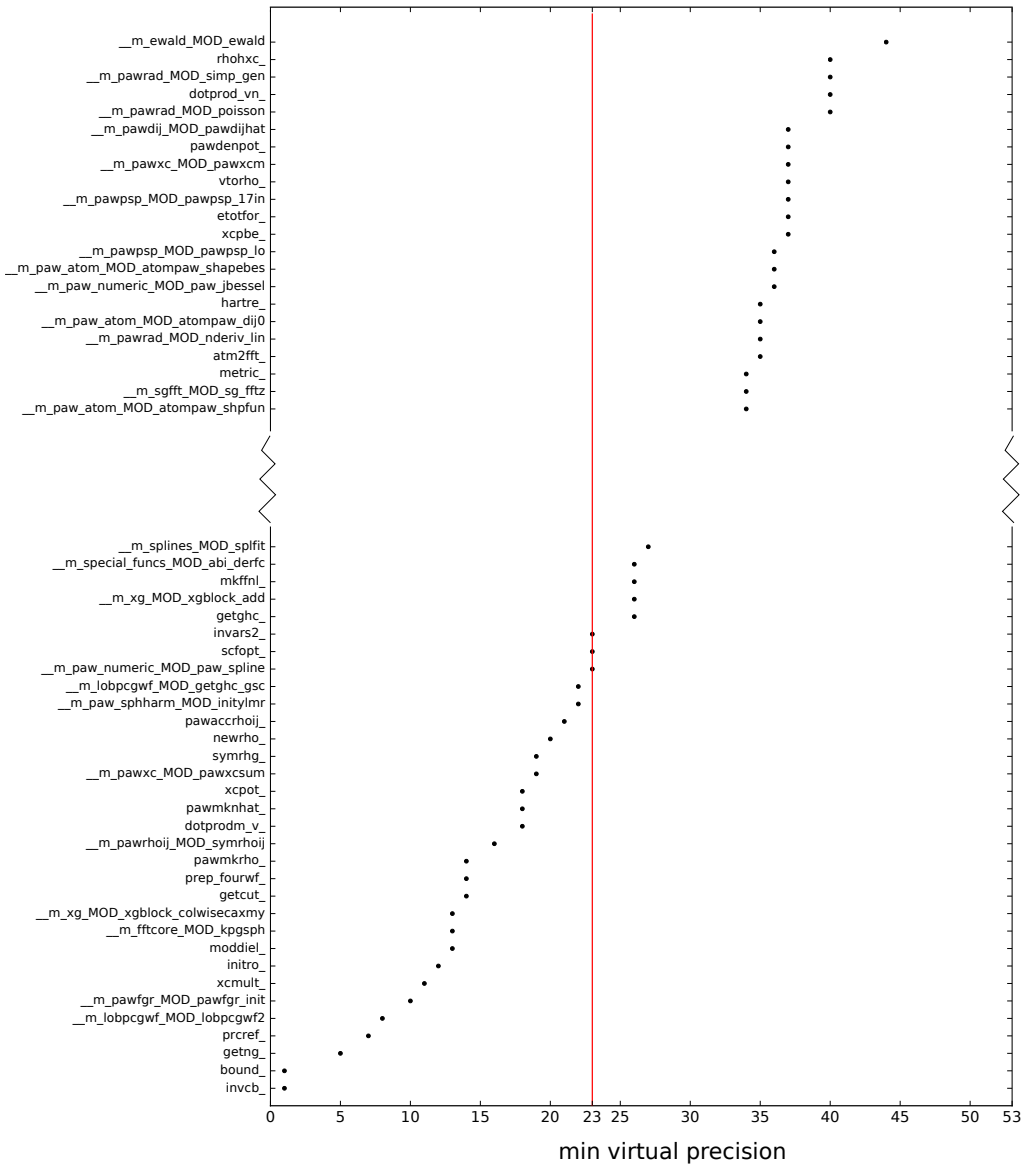


ABINIT: Presentation

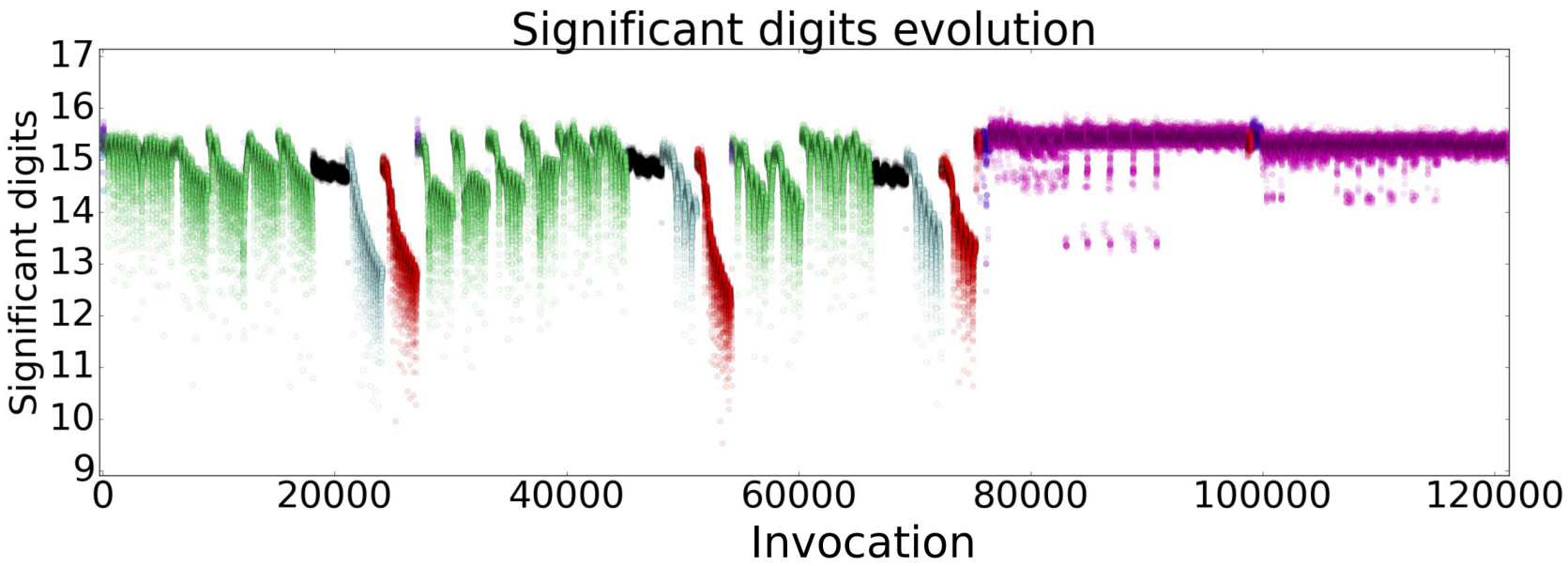
- Scientific computing code developed by an international community of industrial and academic scientifics
- Allows finding the total energy of a quantum system within Density Functional Theory (DFT)
- Code large and complex ($\sim 10^6$ lines of Fortran)
- Suffering from numerical instabilities when vectorized

Checking function sensitivity to precision loss with Verificarlo

- Perovskite ($BaTiO_3$) use case
- Find the minimal virtual precision for each function that guarantees accurate results
  - pinpoint fragile routines
  - detect routines that can be potentially converted to single precision
- The $simp\_gen$ function:
  - computes an integral by the Simpsons' rule
  - 3^{rd} most sensitive function
  - suspected of numerical instabilities by the ABINIT community



VeriTracer analysis on $simp\_gen$ (original)



- Evolution of the number of significant digits over time
- Isolating *Call Sites Path* to avoid confusions
- 24 samples, $t_{virtual} = 53$ with Random Rounding mode
- Running on Occigen GENCI cluster

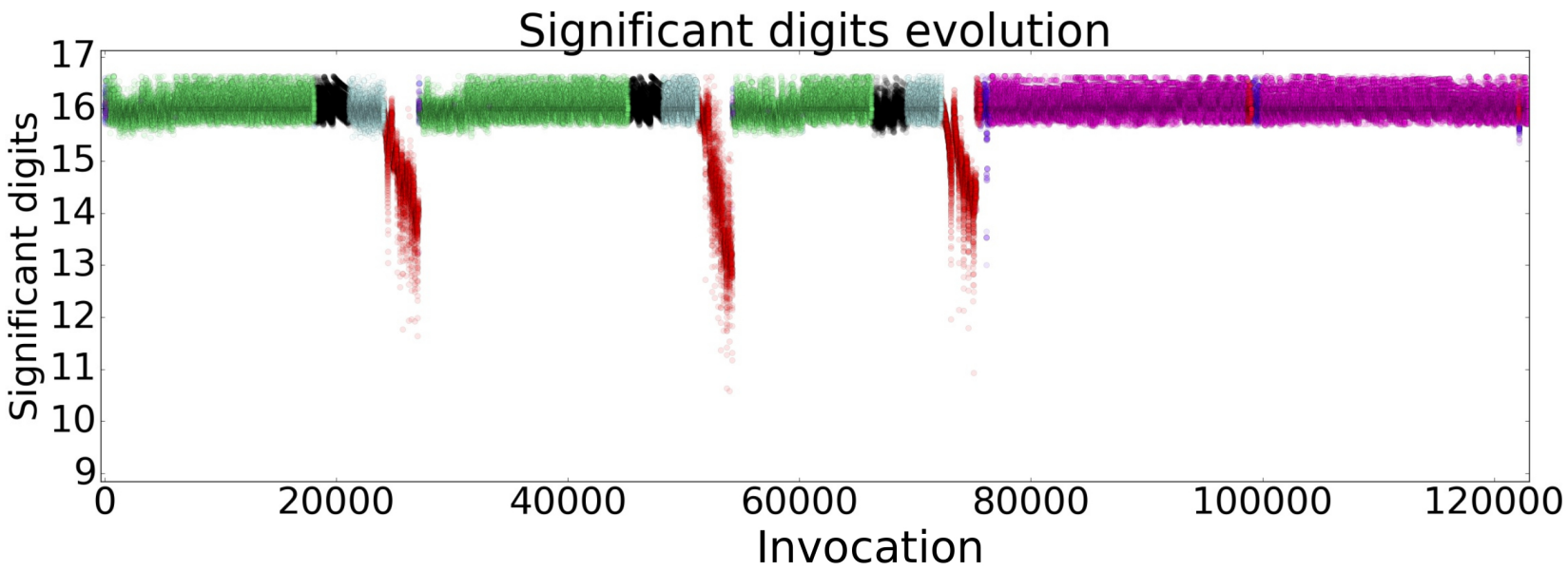
simp_gen code

- 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 Oisha [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
```

VeriTracer analysis on $simp\_gen$ (compensated version)



- *Dot2* compensates numerical errors of 30 CSP among 31
- Red points require deeper analyses

Concluding remarks

- The assessment of the numerical accuracy of scientific codes becomes crucial
  - When porting a scientific code on another programming
  - To find the best compromise between performance and precision
  - Transparent instrumentation eases large code bases analysis.
- Verificarlo is a young project we tested (mainly) on
  - Code ASTER and Telemac (EDF), Europlexus, Abinit (CEA), Yales2 (CORIA), Intel products
- Verrou appears at the same time and propose a complementary approach tested on:
  - EDF Codes: ASTER, Telemac , Cocagne, Athena, Saturn...

Future work: Interflop

- Open framework for numerical analysis tools
- Initiated by EDF, Intel, and UVSQ
- Enables composing Verrou and Verificarlo back-ends (MCA/Random Rounding/...) and front-ends (Valgrind / LLVM instrumentation)
- Unite efforts on processing and analysis tools
- If interested, follow us on github, or even join us!

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

Section 5

Appendix

Monte Carlo Arithmetic: Estimating output error

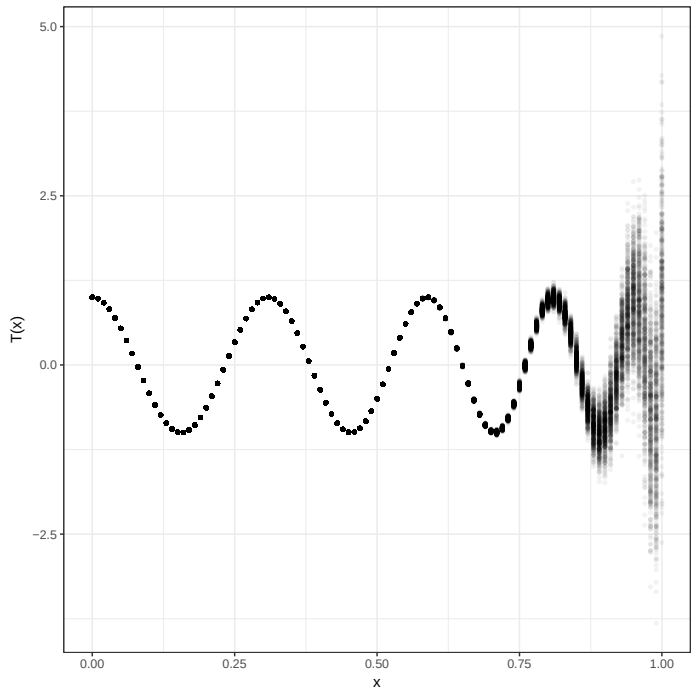


Figure 6: Tchebychev Polynomial, catastrophic cancellation near 1 [Wilkinson, 1994]

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

- Using LLVM brings advantages:
  - The instrumentation library is an independent module which can be tuned for other tools
  - LLVM supports multiple languages and multiple ISA
  - It benefits from the powerful analysis of the LLVM compiler based on code semantics
    - e.g. per function/loop analysis, access to debug info to relate the observation to the source code...
- But also some constraints:
  - Tied to LLVM compiler, addressing a new compiler would require to rewrite the compiler pass (but it is a reasonably short and simple piece of software)
  - Cannot handle precompiled libraries
  - No access to language abstraction makes some approaches tedious (e.g. type overloading)

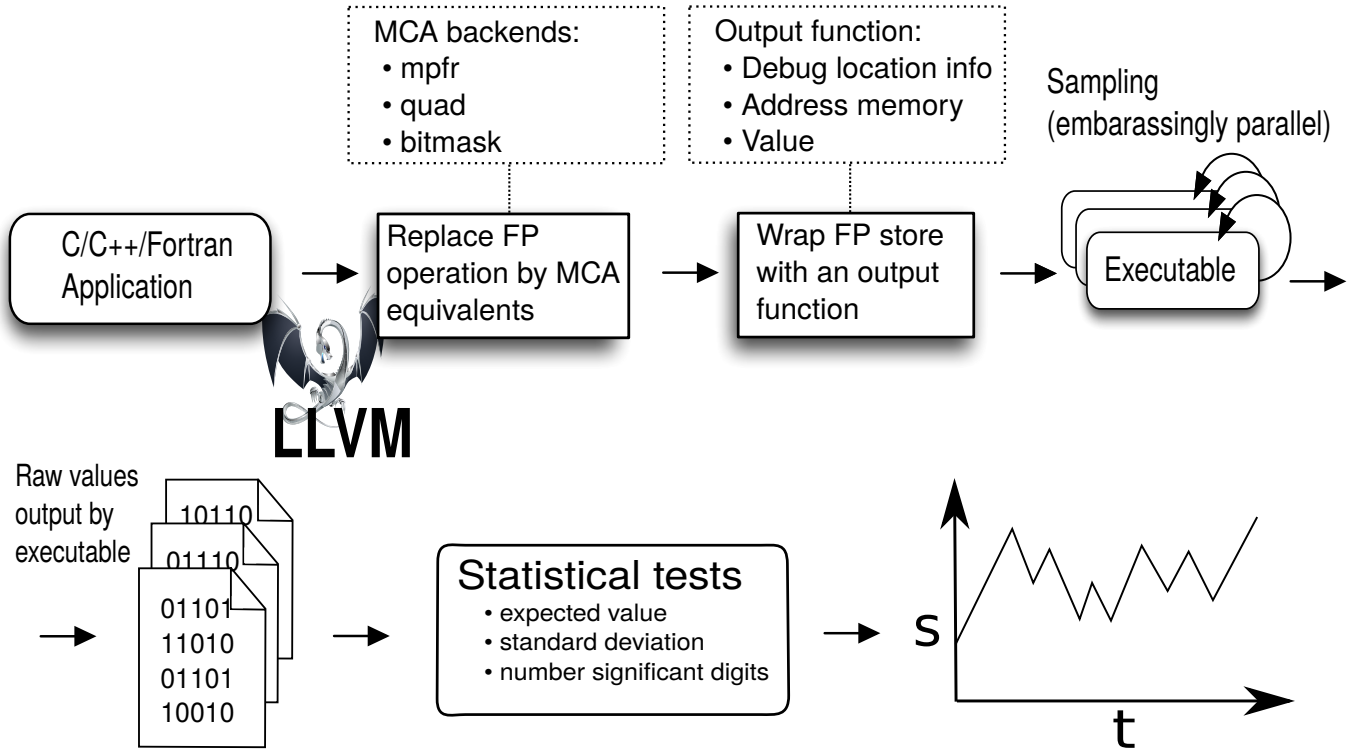
Verificarlo Overhead

version	samples	total time (s)	$\frac{time}{sample}$ (s)
original program	1	.056	.056
Verificarlo MASK	128	12.42	.097
Verificarlo MPFR	128	834.57	6.52
Verificarlo QUAD	128	198.58	1.55
Verificarlo MPFR 16 thds.	128	54.39	.42
Verificarlo QUAD 16 thds.	128	12.54	.098

Table 1: Verificarlo overhead on a compensated sum algorithm (double) on a 16-core 2-socket Xeon E5@2.70GHz.

- Monte Carlo Arithmetic requires additional precision which is costly
- No size fits all
  - MASK backend is cheap (x2 per iteration) but statistically imprecise
  - QUAD backend implements exact MCA model but costly (x27 per iteration)
  - MPFR used only for validation
- Embarrassingly parallel across executions

VeriTracer



- An LLVM pass that insert probes after FP operations
- After executing the code multiple times ...
- ... postprocess raw data and enriches with contextual information
- Visualization of the numerical quality over time

Precision

- Precision is the amount of bits used for the floating point representation and computation
- Precision $\sim$ rounding error

Accuracy

- Accuracy is the distance to the result in the $\mathbb{R}$eal number domain