VeriTracer: Context-enriched tracer for floating-point arithmetic analysis

ARITH 25, Amherst MA USA, June 2018

Yohan Chatelain[1,5] Pablo de Oliveira Castro[1,5] Eric Petit[2,5] David Defour[3]
Jordan Bieder[4,5] Marc Torrent[4,5]

[1]Université de Versailles Saint-Quentin-en-Yvelines (UVSQ)
[2]Intel
[3]Université de Perpignan Via Domitia (UPVD)
[4]CEA, DAM, DIF
[5]Exascale Computing Research (ECR)

Table of contents

1. Introduction

2. Veritracer

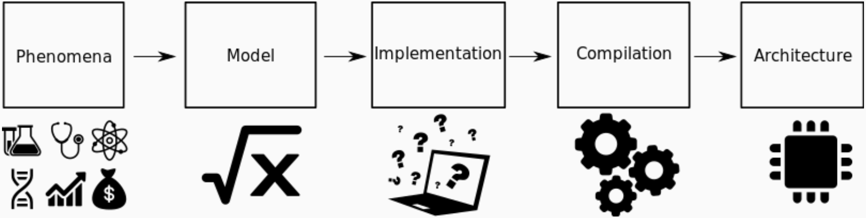
3. Experiments

4. Conclusion

Introduction

Context

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



- At each step, numerical bugs can be introduced

Objective: Tools to track and analyze numerical bugs

Existing tools

	Method	Name	Implementation
Debugging	Stochastic Arithmetic	CADNA [9] VERROU [5] Verificarlo [4]	CESTAC/DSA (library) CESTAC (Valgrind) MCA (LLVM)
	Extended Precision (EP)	HPC Craft [10] FpDebug [3] Herbgrind [15]	Exponent comparison (DynInst) MPFR (Valgrind) MPFR (Valgrind)

	Name	Method	Mixed-Precision	Any-Precision
Optimization	Verificarlo [4]	MCA (User specific)	✓	✓
	HPC Craft [10]	Bitmask (ref value)	✓	✓
	Promise [8]	CESTAC/DSA ($\Delta - debug$)	✓	
	Precimonious [14]	EP ($\Delta - debug$)	✓	
	Herbie [12]	EP (Rewriting)		

Muller's sequence[2] overview

$$u_n = \left\{ \begin{array}{l} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{array} \right\} \lim_{n\to\infty} u_n =$$

exact solution

6

100

finite precision

CADNA	Verificarlo
u_{30} mean: 100 significant digits: 16	u_{30} mean: 100 significant digits: 16

Stochastic methods execute several samples by introducing random errors

Muller's sequence[2] overview

exact solution

6

$$u_n = \left\{ \begin{array}{l} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{array} \right\} \quad \lim_{n \to \infty} u_n =$$

100

finite precision

~~CADNA~~	~~Verificarlo~~
~~u_{30}~~	~~u_{30}~~
~~mean: 100~~	~~mean: 100~~
~~significant digits: 16~~	~~significant digits: 16~~

Without an exact solution

- Only checking the final result may leads to wrong conclusions !

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

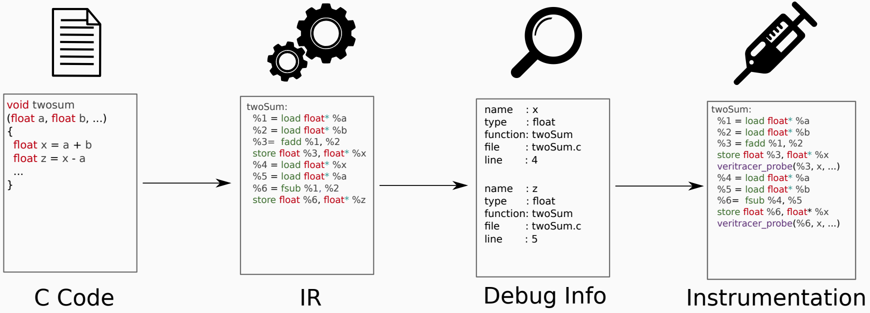
$\Rightarrow$ **Need to explore the temporal dimension**

Veritracer

Instrumentation

Veritracer LLVM pass

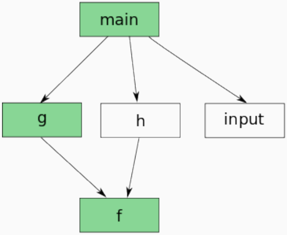
- Accepts any source code accepted by the LLVM frontend
- Transforms into an LLVM Intermediate Representation
- Searches debug information through LLVM IR
- Instruments the IR code with veritracer probes

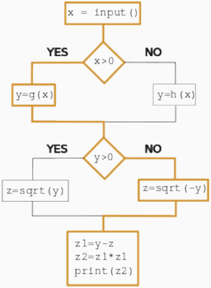


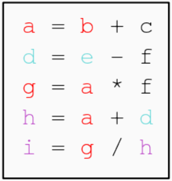
Information Flow Analysis

Ansychronous analysis

- Veritracer works in an asynchronous way
- Information flow is necessary for gathering the traces
- Three levels of granularities are analyzed:
  - **Context Analysis**: considers the context of a function call
  - **Control-Flow Analysis**: considers the flow path taken
  - **Data-Flow Analysis**: considers the dependencies between variables







Pinpointing relevant function

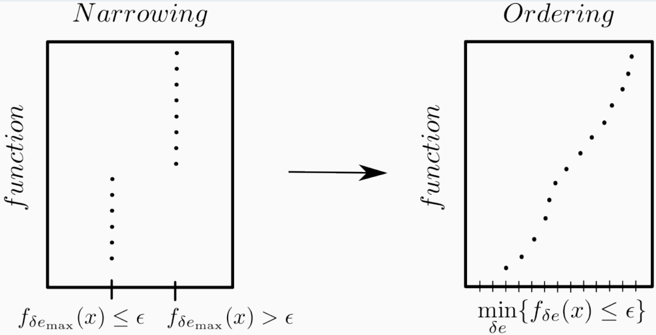
Narrowing the search space

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* $\min_{\delta e}\{f_{\delta e_{\min}} \leq \epsilon\}$



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

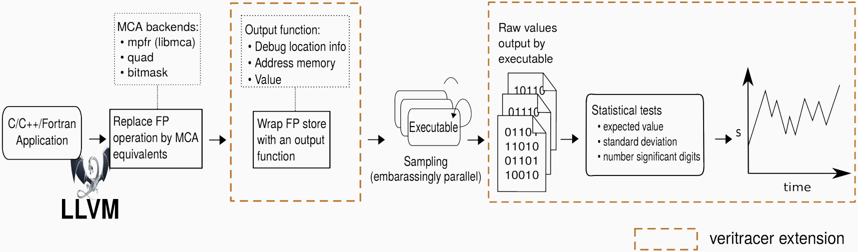


Figure 1: Veritracer's workflow

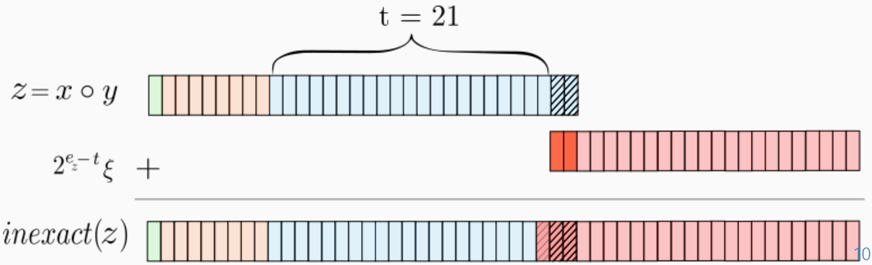
Monte Carlo Arithmetic (MCA) [13]

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

- e_x is the magnitude of x
- t the virtual precision
- ξ a random variable between $\left]-\frac{1}{2}, \frac{1}{2}\right[$

Random Rounding mode

- $RR(x \circ y) = inexact(x \circ y), \ \circ \in \{+, -, \times, \backslash\}$



Experiments

Muller's sequence with veritracer

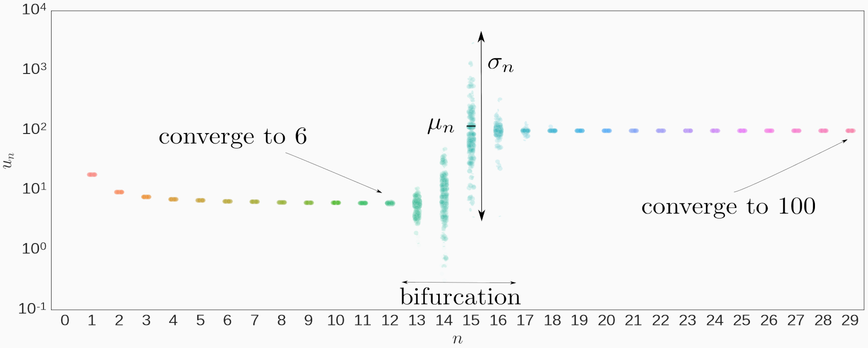


Figure 2: Muller's sequence evaluation over 500 samples with veritracer. The sequence converges to 6 before bifurcating and then converges to 100.

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

ABINIT [7]

- Calculates observable properties of materials (optical, mechanical, vibrational)
- Works on any chemical composition (molecules, nanostructures, solids)

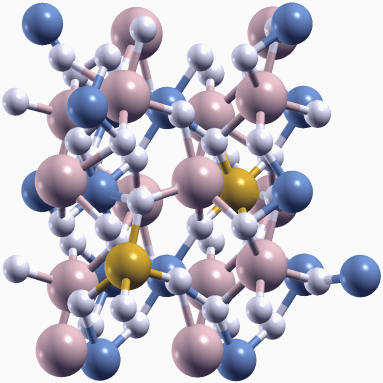
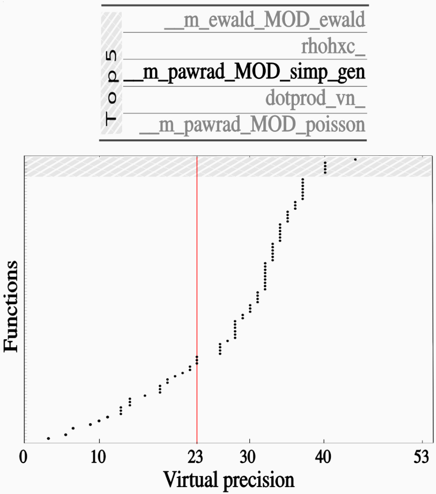


Figure 3: Sound velocity calculation in an earth mantle component ($MgSiO_3$ perovskyte with Al impurities) [1]

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

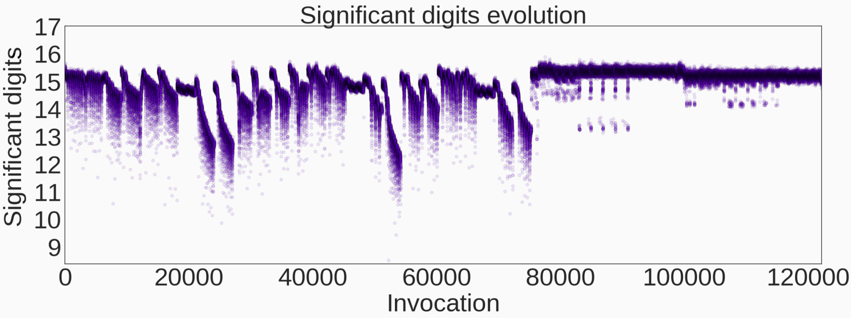
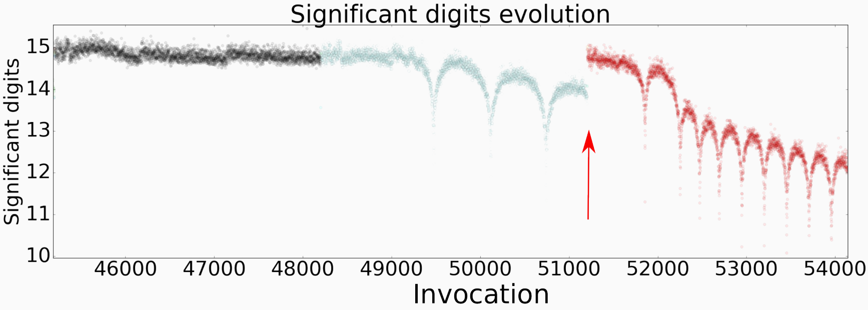


Figure 4: Evolution of the value returned by simp_gen with $t = 53$ in Random Rounding mode with 24 samples in parallel on the **CINES Occigen** cluster (2106 nodes x2 Intel 12-Cores (E5-2690V3@2.6 GHz))

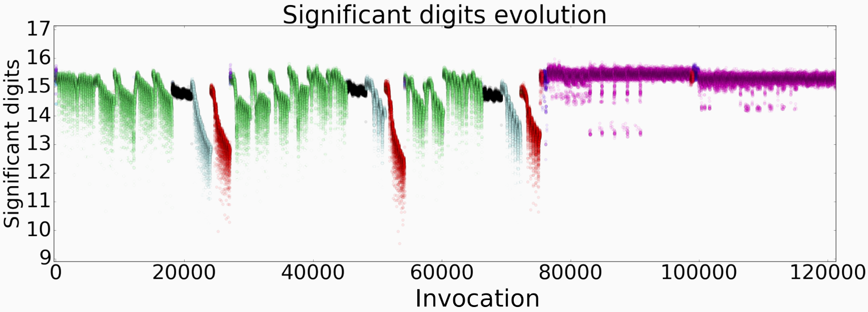
Simp_gen: the importance of the context analysis

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

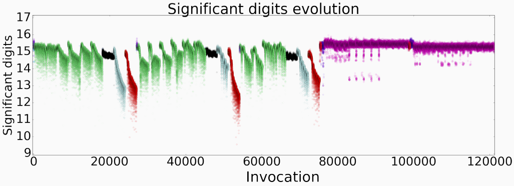


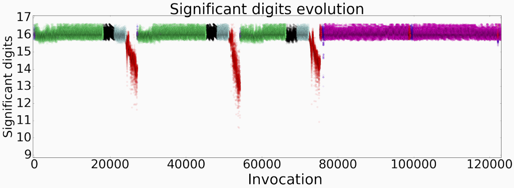
Simp_gen: the importance of the context analysis

- 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



Simp_gen: A compensated version





Simp_gen: Altered version

- Can be seen a dot product
- Replaces by a compensated version Dot2 [11]
- Implemented with `libeft` [6]
- The precision of 30/31 CSPs is improved
- 1 CSP has a low precision due to reentrance of the error

Conclusion

Conclusion

Veritracer is a numerical debugger/optimizer which

- Automatically instruments codes through the verificarlo framework
- Automatically plots the temporal 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 according to their numerical sensitivity
- Observe the numerical behavior of functions over time

Thank you for your attention

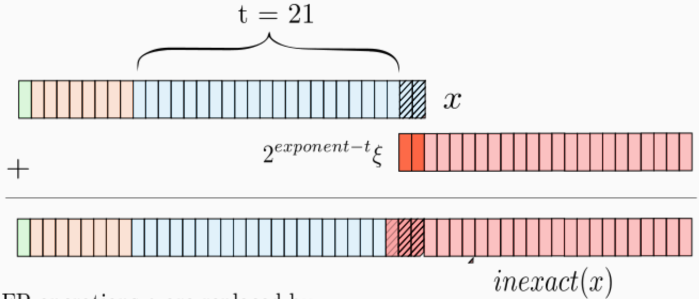
Available on github:

`github.com/verificarlo/verificarlo`
`github.com/verificarlo/verificarlo/tree/veritracer`

Random Rounding mode (MCA)

Monte Carlo Arithmetic [parker1997monte]

$$inexact(x) = x + 2^{exponenent-t}\xi, \quad \xi \in [-\frac{1}{2}, \frac{1}{2}], \quad \text{t virtual precision}$$



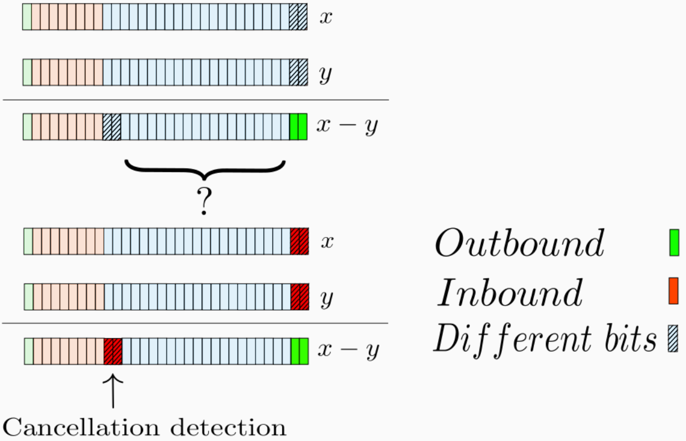
FP operations $\circ$ are replaced by:

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

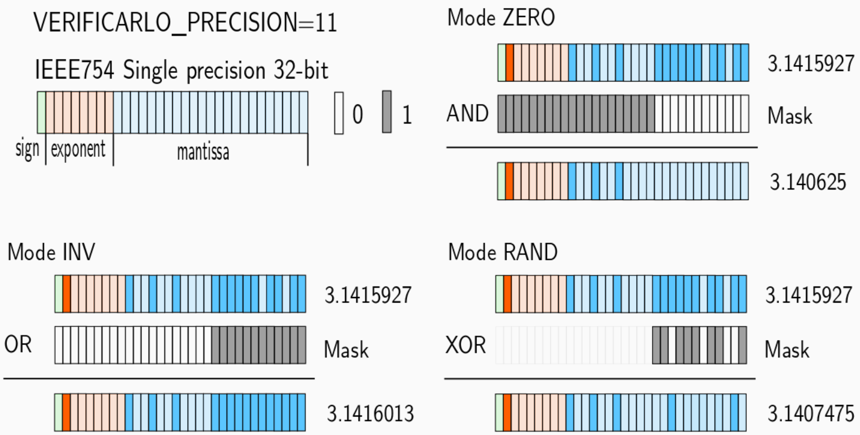
Full MCA mode

$$x \sim y$$

$$mca(x) = round(inexact(inexact(x) - inexact(y)))$$



Bitmask backend



Errors propagation

```
subroutine pawpsp_calc(...)
...
if (testval) then
    ...
    call simp_gen(qq, nhat, vale_mesh)
    qq=zion/four_pi-qq
end if

call atompaw_shpfun(..., intg, nhat)

♠ nhat(1:msz)=qq*nhat(1:msz)
tnvale(1:msz)=tnvale(1:msz)+nhat(1:msz)
...
call pawpsp_cg(..., tnvale, ...)
```

```
subroutine atompaw_shpfun(..., norm, shapefunc)
...
simp_gen(norm,r2k,mesh)
norm=one/norm
shapefunc(1:ishp)=shapefunc(1:ishp)*norm
```

out

inout

```
subroutine pawpsp_cg(..., nr, ...)
...
do ir=1,mesh_size
    rnr(ir)=radmesh%rad(ir)*nr(ir)
end do
...
do ir=1,mesh_size
    if (abs(rnr(ir))>1.d-20)
        ff(ir)=sin(arg*radmesh%rad(ir))*rnr(ir)
end do
...
♣ call simp_gen(r1torm, ff, radmesh)
```

out

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

Veritracer: the LLVM IR

```
void twoSum
(double a, double b, double *x_ptr, double *err_ptr) {
  double x = a + b;
  double z = x - a;
  double e = (a - (x - z)) + (b - z);
  *x_ptr = x;
  *err_ptr = e;
}
```

```
define void @twoSum
(float %a, float %b, float* %x_ptr, float* %e_ptr)
{
  %5 = load float* %a          ;
  %6 = load float* %b          ;
  %7 = fadd float %5, %6       ;     a+b
  store float %7, float* %x    ; x = a+b
  %8 = load float* %x          ;
  %9 = load float* %a          ;
  %10 = fsub float %8, %9      ;     x-a
  store float %10, float* %z   ; z = x-a
  %11 = load float* %a         ;
  %12 = load float* %x         ;
  %13 = load float* %z         ;
  %14 = fsub float %12, %13    ;     x-z
  %15 = fsub float %11, %14    ;     a-(x-z)
  %16 = load float* %b         ;
  %17 = load float* %z         ;
  %18 = fsub float %16, %17    ;     b-z
  %19 = fadd float %15, %18    ;     a-(x-z)+(b-z)
  store float %19, float* %e   ; e = a-(x-z)+(b-z)
  %20 = load float* %x         ;
  %21 = load float** %3        ;
  store float %20, float* %21 ; *x_ptr = x
  %22 = load float* %e         ;
  %23 = load float** %4,       ;
  store float %22, float* %23 ; *e_ptr = e
  ret void
}
```

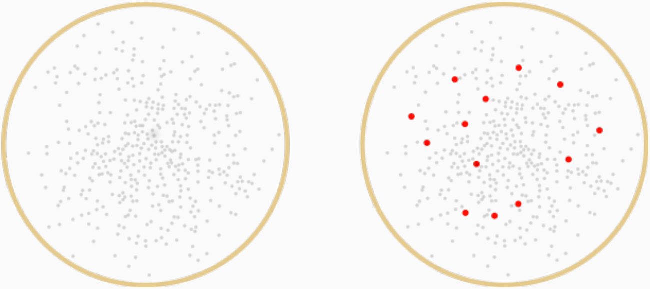
```
define void @twoSum
(float %a, float %b, float* %x_ptr, float* %err_ptr)
{
  %1 = fadd float %a, %b        ; a+b     (x)
  %2 = fsub float %1, %a        ; (a+b)-a (z)
  %3 = fsub float %1, %2        ; x-z
  %4 = fsub float %a, %3        ; a-(x-z)
  %5 = fsub float %b, %2        ; b-z
  %6 = fadd float %5, %4        ; (b-z)+(a-(x-z))
  store float %1, float* %x_ptr ; *x_ptr = x
  store float %6, float* %e_ptr ; *e_ptr = e
  ret void
}
```

Reducing the search space

Instrumenting one function of the program at a time

- Executes each version with the maximal MCA error
- Compares the MCA computed value with the reference:

  Different : the function is tagged as critical
  Identical : the function is tagged not critical



2952 function ⇒ 88 functions
Reduces the search space by ×30

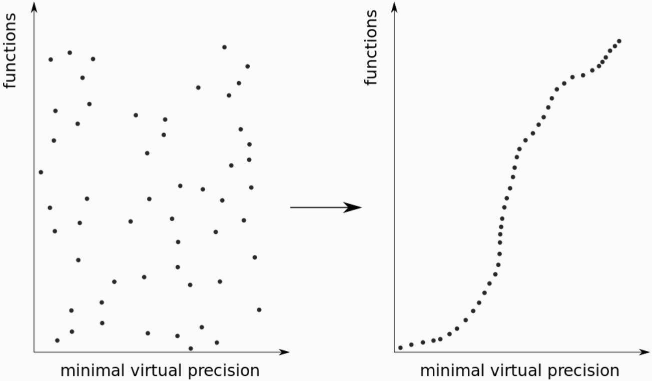
Ordering the search space

Problem

Functions do not have the same numerical behavior: some functions are less sensitive to numerical errors than others

Idea

Classifying functions according to their numerical stress resistance



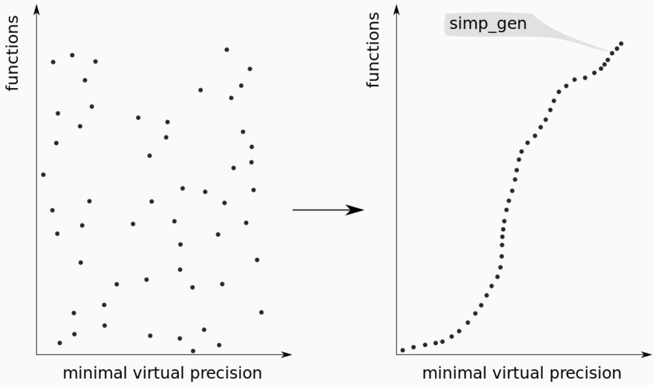
Ordering the search space

Problem

Functions do not have the same numerical behavior: some functions are less sensitive to numerical errors than others

Idea

Classifying functions according to their numerical stress resistance



[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.

[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. Lathuiliere.
VERROU: a CESTAC evaluation without recompilation.
SCAN 2016, page 47, 2016.

[6] F. Févotte and B. Lathuilière.
LibEFT: a library implementing Error-Free transformations, 2017.

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

[8] 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.

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

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

[11] T. Ogita, S. M. Rump, and S. Oishi.
Accurate sum and dot product.
SIAM Journal on Scientific Computing, 26(6):1955–1988, 2005.

[12] 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.

[13] D. S. Parker.
Monte Carlo Arithmetic: exploiting randomness in floating-point arithmetic.
University of California. Computer Science Department, 1997.

[14] 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.

[15] 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.