VERROU: a valgrind tool dedicated to floating point error diagnosis

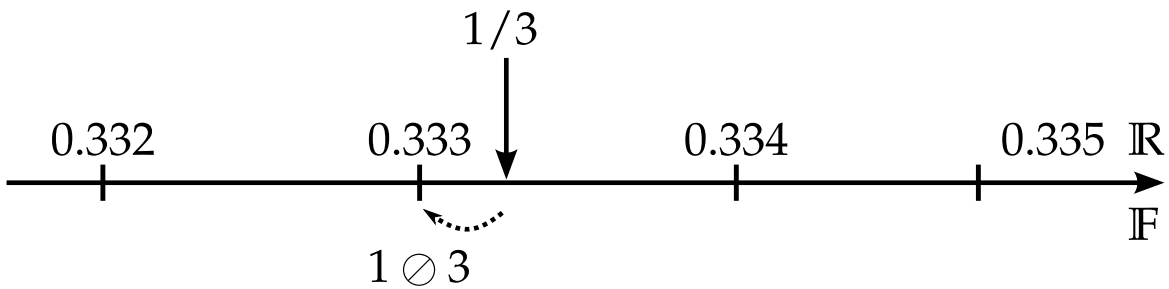
02/03/23
FOSDEM

Bruno Lathuilière (EDF R&D)

With financial support from:
ANR Interflop ANR-20-CE46-0009.

Floating point error

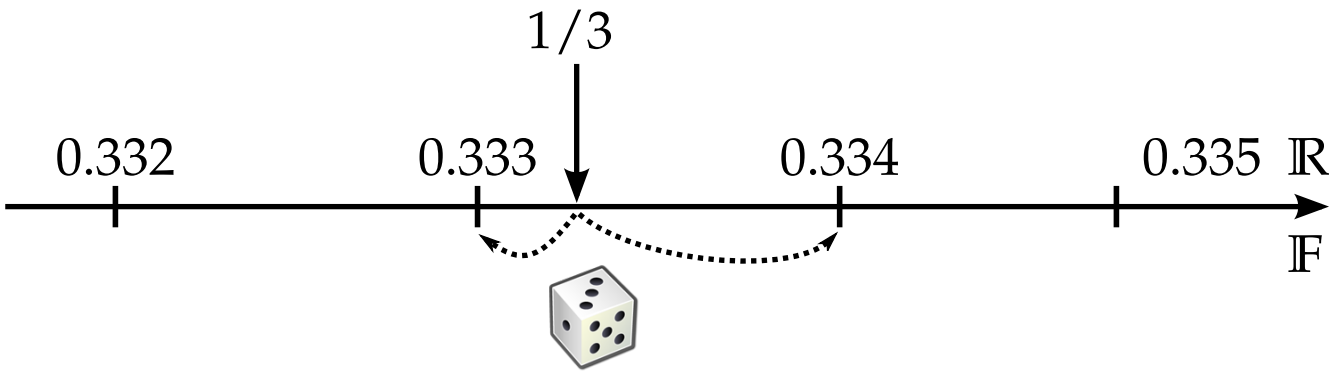
- Floating point representation with limited precision
  - [`float`] binary, 24 significand bits ($\simeq 10^{-7}$)
  - [`double`] binary, 53 significand bits ($\simeq 10^{-16}$)
  - [pedagogic example] decimal, 3 significand digits (% – ‰)



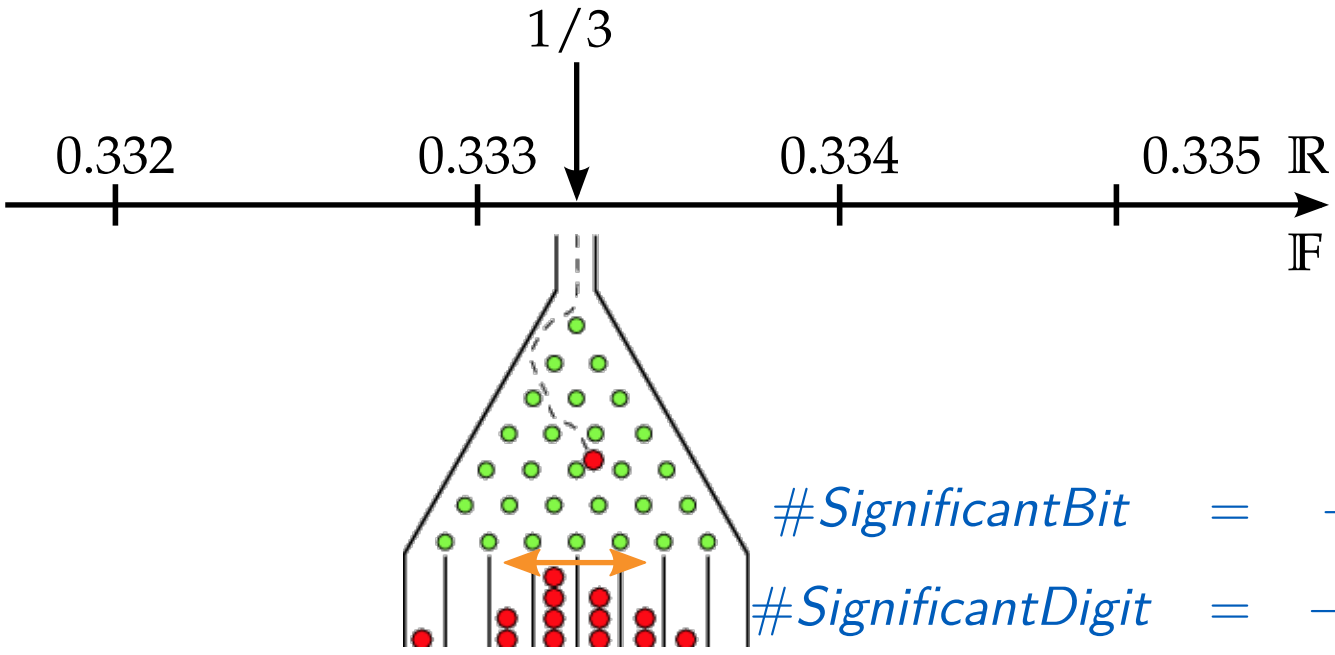
- Floating point computation $\neq$ Real computation
  - rounding error $a \oplus b \neq a + b$
  - associativity loss $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$
- Need a tool to do error estimation of industrial complex applications:

VERROU

Mathematical background: stochastic arithmetic



Mathematical background: stochastic arithmetic



$$\#SignificantBit = -log2\left(\frac{max_i(|X_i - X_{nearest}|)}{|X_{nearest}|}\right)$$

$$\#SignificantDigit = -log10\left(\frac{max_i(|X_i - X_{nearest}|)}{|X_{nearest}|}\right)$$

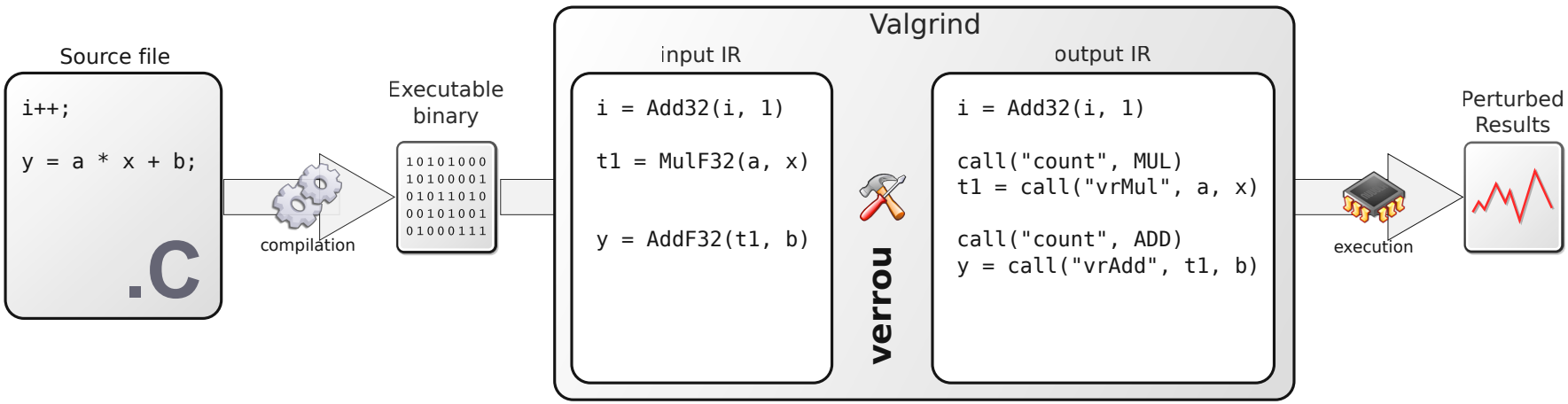
Instruction	Eval. Nearest	Eval. 1	Eval. 2	Eval. 3
$a = 1/3$	0.333	$0.333_{\downarrow}$	$0.334^{\uparrow}$	$0.334^{\uparrow}$
$b = a \times 3$	0.999	0.999	$1.00_{\downarrow}$	$1.01^{\uparrow}$

$\#SignificantDigit \approx 1.95$

The Verrou tool

Dynamic binary instrumentation thanks Valgrind

```
$ valgrind --tool=verrou --rounding-mode=random PROGRAM [ARGS...]
```



User point of view:

- need to run the code several times;
- need to extract value of interest;
- all languages (C/C++, Fortran, python . . .);
- external libraries;
- amd64 only.

Valgrind developer point of view:

- replace all FP operations;
- modify the results;
- no need of shadow memory.

Muller example

```
1  float muller(int nt, bool verbose=false){
2     float x0 = 11./2.;
3     float x1 = 61./11.;
4     std::cout << "begin iter"<<std::endl;
5     for(size_t it=0; it < nt ; it++){
6       float temp0 = 3000./x0;
7       float temp1 = 1130. - temp0;
8       float temp2 = temp1 /x1 ;
9       float x2 = 111. - temp2;
10      if(verbose){
11        cout <<"it: "<< it << "\tx2: "<<x2
12             << "\ttemp0: "<<temp0 <<"\ttemp1: "<<temp1<<"\ttemp2: "<<temp2 <<"addr: "<<&temp2<< endl;
13      }
14      x0 = x1;
15      x1 = x2;
16    }
17    std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
18    return x1;
19 }
```

```
begin iter
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.41addr: 0x7fffe0b19328
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x7fffe0b19328
it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x7fffe0b19328
it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x7fffe0b19328
it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x7fffe0b19328
it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x7fffe0b19328
it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x7fffe0b19328
it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x7fffe0b19328
it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x7fffe0b19328
it: 9 x2: 5.88354 temp0: 513.904 temp1: 616.096 temp2: 105.116addr: 0x7fffe0b19328
it: 10 x2: 5.93596 temp0: 511.851 temp1: 618.149 temp2: 105.064addr: 0x7fffe0b19328
it: 11 x2: 6.53442 temp0: 509.897 temp1: 620.103 temp2: 104.466addr: 0x7fffe0b19328
x[12]=6.53442
```

Muller example

```
1  float muller(int nt, bool verbose=false){
2     float x0 = 11./2.;
3     float x1 = 61./11.;
4     std::cout << "begin iter"<<std::endl;
5     for(size_t it=0; it < nt ; it++){
6       float temp0 = 3000./x0;
7       float temp1 = 1130. - temp0;
8       float temp2 = temp1 /x1 ;
9       float x2 = 111. - temp2;
10      if(verbose){
11        cout <<"it: "<< it << "\tx2: "<<x2
12             << "\ttemp0: "<<temp0 <<"\ttemp1: "<<temp1<<"\ttemp2: "<<temp2 <<"addr: "<<&temp2<< endl;
13      }
14      x0 = x1;
15      x1 = x2;
16    }
17    std::cout <<"x["<<nt<<"]="<<x1<<std::endl;
18    return x1;
19 }
```

```
begin iter
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.41addr: 0x7fffe0b19328
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x7fffe0b19328
it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x7fffe0b19328
it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x7fffe0b19328
it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x7fffe0b19328
it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x7fffe0b19328
it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x7fffe0b19328
it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x7fffe0b19328
it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x7fffe0b19328
it: 9 x2: 5.88354 temp0: 513.904 temp1: 616.096 temp2: 105.116addr: 0x7fffe0b19328
it: 10 x2: 5.93596 temp0: 511.851 temp1: 618.149 temp2: 105.064addr: 0x7fffe0b19328
it: 11 x2: 6.53442 temp0: 509.897 temp1: 620.103 temp2: 104.466addr: 0x7fffe0b19328
x[12]=6.53442
```

You've got a tricky floating point bug somewhere in your 2 million lines of code.

Delta-debug: *trial and error* search algorithm

runScript: ddRun.sh

```
1 #!/bin/bash
2 PREFIX="valgrind --tool=verrou
3                 --rounding-mode=random"
4 $PREFIX ./muller -v > $1/res.dat
```

cmpScript: extractOrCmp.py

```
1 #!/usr/bin/python3
2 def extractValue(rep):
3     for line in ↩
          (open(os.path.join(rep,"res.dat")).readlines()):
4         if line.startswith("x[12]="):
5             return float(line.partition("=")[2])
6
7 if __name__=="__main__":
8     if len(sys.argv)==2:
9         print(extractValue(sys.argv[1]))
10    if len(sys.argv)==3:
11        valueRef=extractValue(sys.argv[1])
12        value=extractValue(sys.argv[2])
13        relDiff=abs((value-valueRef)/valueRef)
14        if relDiff < 1.e-2:
15            sys.exit(0)
16        else:
17            sys.exit(1)
```

Delta-debug search:

```
1 verrou_dd_line --nruns=5 ddRun.sh extractOrCmp.py
```

```
ddmin0 (
muller.cpp:14 (muller(unsigned long, bool))):
...
ddmin1 (
muller.cpp:12 (muller(unsigned long, bool))):
```

Valgrind developer point of view:

- need to generate a search space: list of symbols (or line if compiled with -g) containing floating point operations;
- need to run a specific configuration (set instrumented /not instrumented).

Towards temporal localisation

New search space: wildcarded IO (automatic thanks regexp or manually defined)

```
begin iter
it: 0 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 1 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 2 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 3 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 4 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 5 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 6 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 7 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 8 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 9 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 10 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
it: 11 x2: * temp0: * temp1: * temp2: *addr: 0x???????????
x[12]=*
```

```
1 verrou_dd_stdout --nruns=5 ddRun.sh extractOrCmp.py
```

```
ddmin0 (
|begin iter|):
...
ddmin1 (
|it: 0 x2: * temp0: * temp1: * temp2: *addr: 0x??????????|):
```

- The stdout (or a file) *match* without temporal context;
- The user has to pay attention to bufferization;
- The empty line can be ignored;
- The stdout can be modified by a filter script.

Valgrind developer: file format to define interaction between *IO* and *verrou*

Conclusion

VERROU can be used to:

- estimate floating point error;
- search the origin of floating point error;
- search mixed precision configuration;
- search where errors are amplified.

Among the roadmap:

- New architectures (work in progress for arm64);
- New search spaces (backtrace);
- Error amplification localisation.

VERROU

Available on github:
`http://github.com/edf-hpc/verrou`

Documentation:
`http://edf-hpc.github.io/verrou/vr-manual.html`

Papers:

- François Févotte and Bruno Lathuilière. Debugging and optimization of HPC programs with the Verrou tool. In International Workshop on Software Correctness for HPC Applications (Correctness), 2019.
- François Févotte and Bruno Lathuilière. Studying the numerical quality of an industrial computing code: A case study on code_aster. In 10th International Workshop on Numerical Software Verification (NSV), 2017.
- François Févotte and Bruno Lathuilière. VERROU: a CESTAC evaluation without recompilation. In International Symposium on Scientific Computing, Computer Arithmetics and Verified Numerics (SCAN), 2016.

Performance

Program: stencil with fma (warning: huge variability between test case)

type	double		float	
compilation option	O0	O3	O0	O3
nearest	x12.2	x26.2	x12.4	x35.0
tool_none	x13.9	x55.7	x10.5	x59.1
fma_only	x7.1	x13.4	x7.4	x16.5
random	x19.1	x53.8	x20.2	x83.3
random_det	x19.6	x51.9	x20.7	x85.6
random_comdet	x20.1	x55.7	x21.4	x90.6
random_scomdet	x23.8	x70.1	x25.9	x117
average	x22.3	x60.8	x23.5	x97.5
average_det	x23.9	x66.0	x26.0	x108
average_comdet	x24.8	x69.6	x27.6	x126
average_scomdet	x25.5	x72.4	x28.9	x131
sr_monotonic	x23.8	x67.2	x27.0	x118
sr_smonotonic	x24.0	x68.2	x27.3	x118

valgrind framework: my feedback

Valgrind website citation: *Writing a new Valgrind tool is not easy, but the tools you can write with Valgrind are among the most powerful programming tools there are. Happy programming!*

- Light developer documentation... but sources and tools are well commented;
- Do not need (yet) any modification in valgrind code base;
- Programming in C-- (C++ without standard library);
- Hardware rounding mode not available (but fruitful constraint);
- Easy valgrind version upgrade;
- Gdb integration:
  - Easy but very slow
  - Watch interface designed for shadow memory
- Vectorized fma IR is missing;
- Port to arm64 (work in progress: not obvious);
- Thread sequential ordering: as there is parallelism over samples, these user's complains are rarely justified and as verrou developer it make my life easier.

My dreams:

- instrument vector operations without unvectorization.
- dynamic selection between two versions of instrumented BB.

Few numbers about verrou

Language:

```
ansic: 9676 (36.64%) (Warning code generation)
python: 7062 (26.74%)
cpp: 6636 (25.13%)
xml: 2816 (10.66%)
sh: 218 (0.83%)
```

Number of options:

valgrind –tool=verrou	38
verrou_dd_[line/sym]	16
verrou_dd_stdout	16+6
post_verrou_dd	10
verrou_plot_stat	17

Number of rounding modes: 22

```
random[|_det|_comdet|_scomdet] average[|_det|_comdet|_scomdet]
prandom[|_det|_comdet] sr_[s]monotonic
nearest native upward downward toward_zero away_zero farthest
float ftz
```

expect-clr format

The option `-expect-clr=EXPECT_FILE` (or environnement variable `VERROU_EXPECT_CLR`) enables interaction between stdout (or file) and verrou.
The format is separated into two sections (headers and temporal sections) separated by the key `begin:`.
In headers there are:

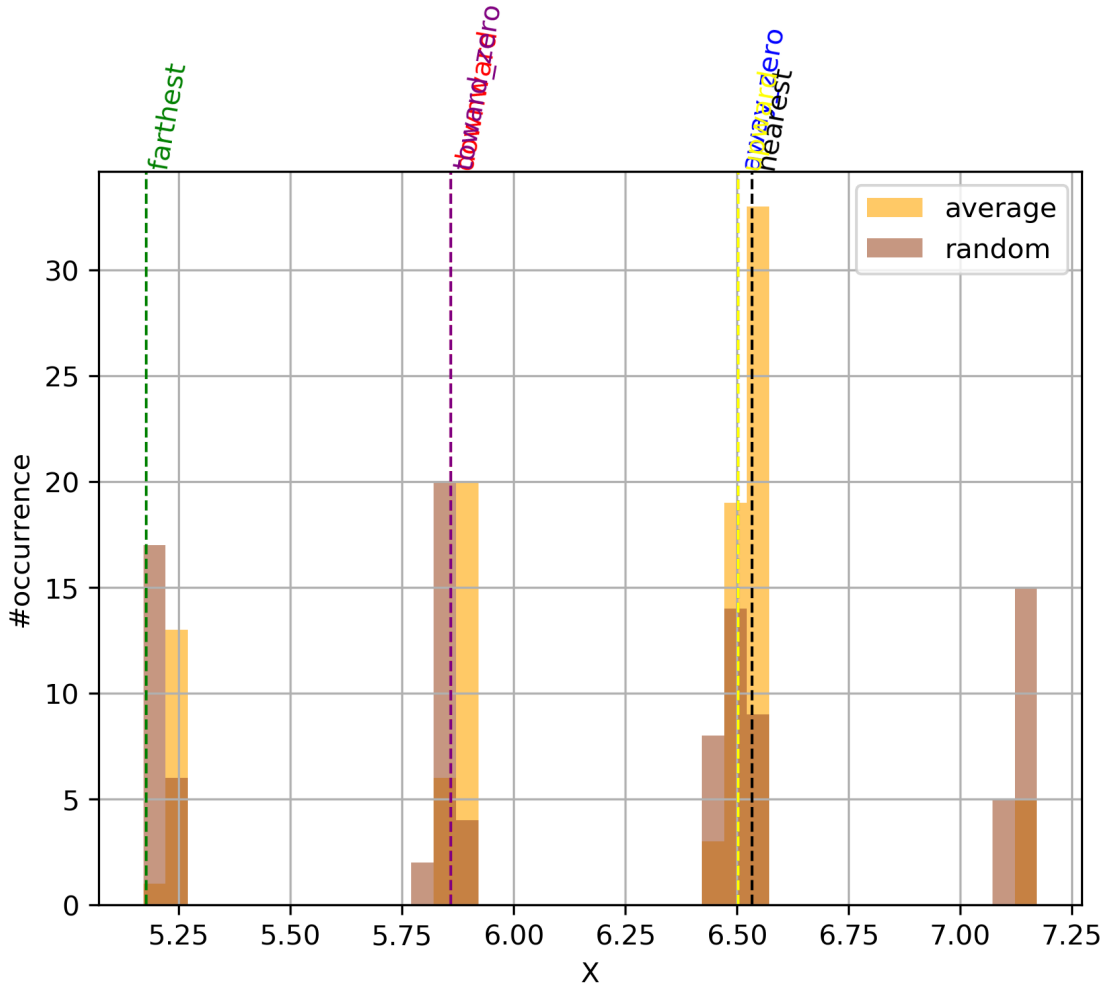
- setup keys: `verbose: LEVEL`, `ignore-empty-line:` , `filter_line_exec: CMD`, `dump-stdout: [FILENAME]`, `dump-filtered-stdout: [FILENAME]`
- definition of specific action: `default: ACTION`, `init: ACTION`, `post_init: ACTION`.
- definition of match action: `match: PATTERN`, `apply: ACTION`, `post-apply: ACTION`.

In temporal section there are:

- `except: PATTERN`, `apply: ACTION`.

The available actions are: `start`, `stop`, `nop`, `display_counter`, `nb_instr`, `reset_counter`, `dump_cover`, `panic`, `exit`, `default`, `init`, `post-init`

Histogram


farthest
toward_zero
away_zero
nearest
average
random
#occurrence
X
0
5
10
15
20
25
30
5.25
5.50
5.75
6.00
6.25
6.50
6.75
7.00
7.25

Tricky use of verrou_dd_stdout

- Iterative algorithm that corrects the errors

```
1 verrou_dd_stdout --expect-header=expect.header ddRun.sh extractOrCmp.py
```

File expect.header:
```
match: it: 15=*
apply: exit
```

- Untimely debug message;

```
begin iter
debug
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.41addr: 0x7fffe0b19328
debug
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x7fffe0b19328
debug
...
```

```
1 verrou_dd_stdout --filter-cmd="/usr/bin/sed -u s/debug.*//" ddRun.sh extractOrCmp.py
```

- the script is run once per binary (not once per line);
- the filter can by written in python (take care about bufferisation).
  - it is possible to reintroduce context (useful for iner/outer iteration);
  - it is possible to group iterations.

Other Interflop tools based on stochastic arithmetics

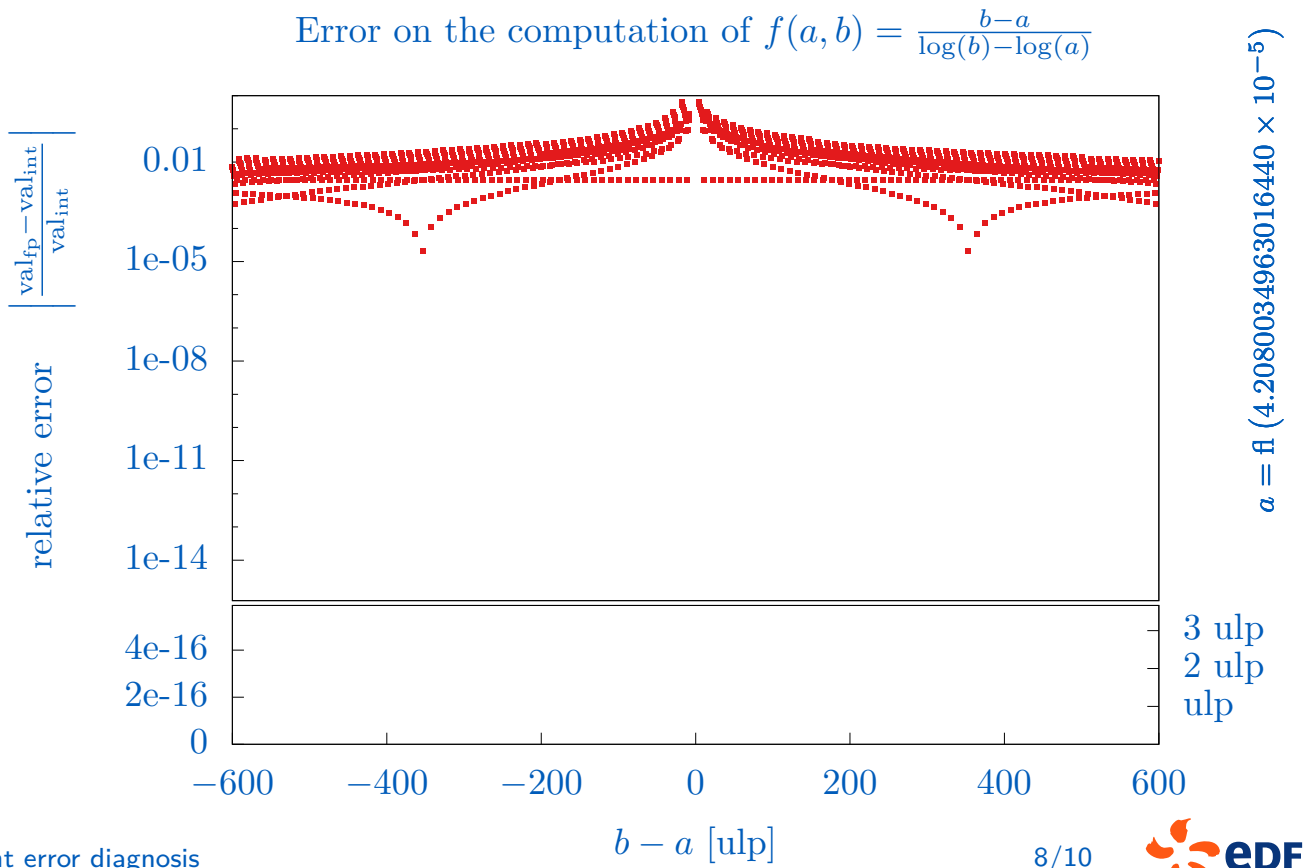
- CADNA
  - url: `https:/https://www-pequan.lip6.fr/cadna/`
  - instrumentation: source level
  - method DSA (discret stochastic arithmetic): CESTAC with 3 samples + synchronous approach
  - we have access the to precision for each variable (not only *post mortem*)
- Verificarlo
  - url: `https://github.com/verificarlo/verificarlo`
  - instrumentation: low-level LLVM pass
- PENE
  - url: `https://github.com/aneoconsulting/PENE`
  - instrumentation: PIN framework
  - windows portability

Bug fix example (1/3)

$$f(a,b) = \begin{cases} a & \text{si } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{sinon} \end{cases}$$

Empirical study

- outside the code
- around the problematic
- reference = interval arithmetic

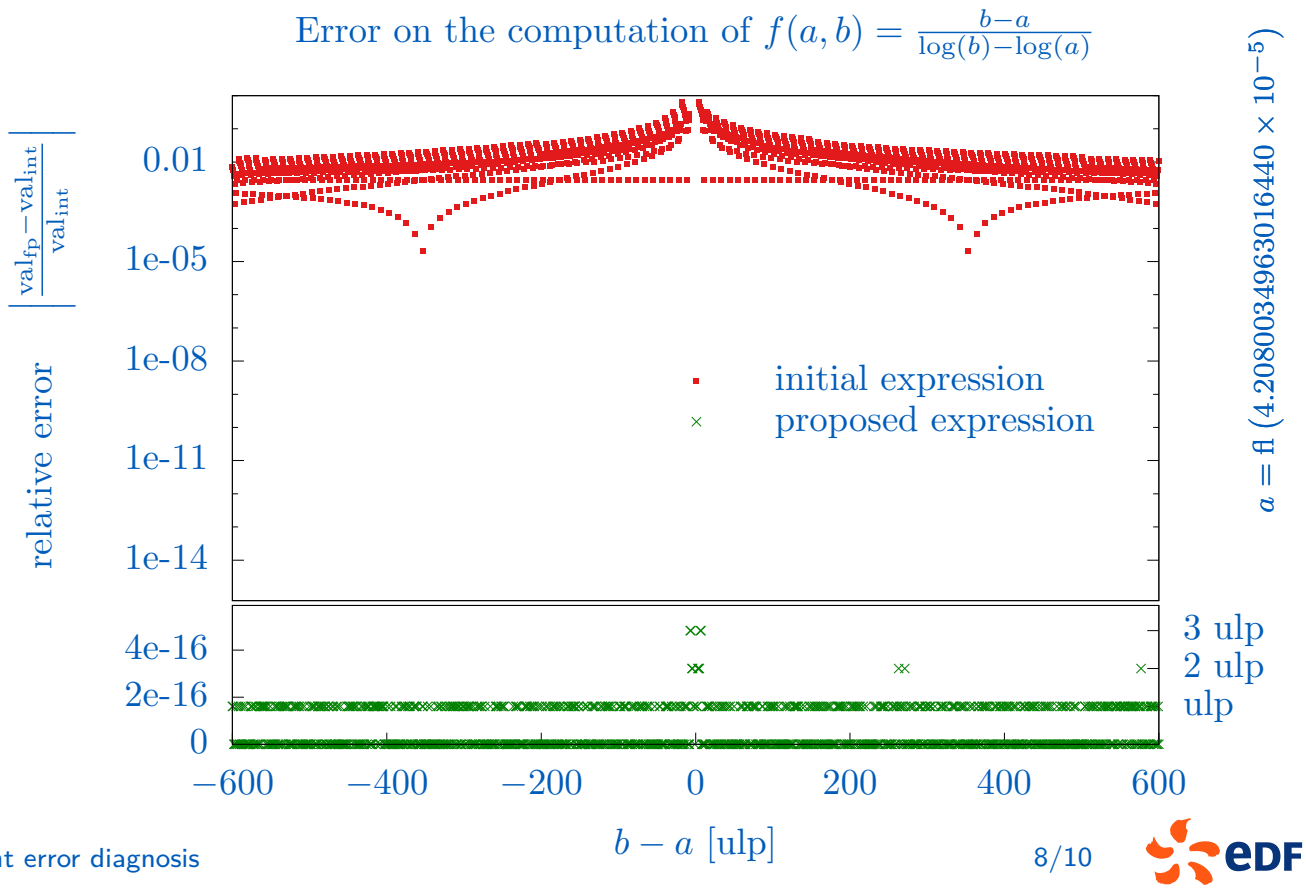


Bug fix example (1/3)

$$f(a,b) = \begin{cases} a & \text{si } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{sinon} \end{cases} \xrightarrow[\text{manuelle}]{\text{récriture}} f(a,b) = \begin{cases} a & \text{si } a = b \\ a\,\frac{\frac{b}{a}-1}{\log(\frac{b}{a})} & \text{sinon} \end{cases}$$

Empirical study

- outside the code
- around the problematic
- reference = interval arithmetic



Bug fix example (1/3)

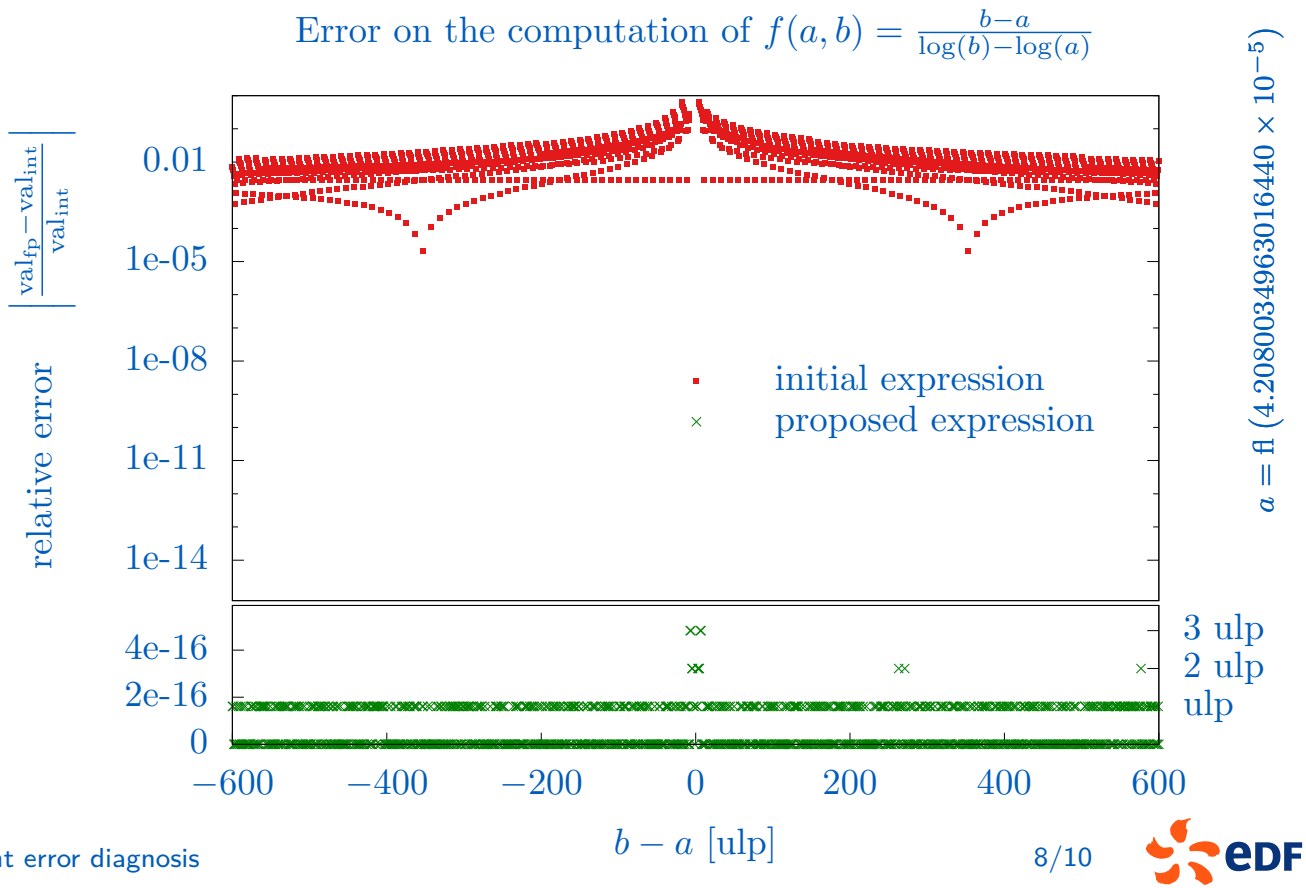
$$f(a,b) = \begin{cases} a & \text{si } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{sinon} \end{cases} \xrightarrow[\text{manuelle}]{\text{récriture}} f(a,b) = \begin{cases} a & \text{si } a = b \\ a\,\frac{\frac{b}{a}-1}{\log(\frac{b}{a})} & \text{sinon} \end{cases}$$

Empirical study

- outside the code
- around the problematic
- reference = interval arithmetic

Proof

- error bounded by 10 ulps

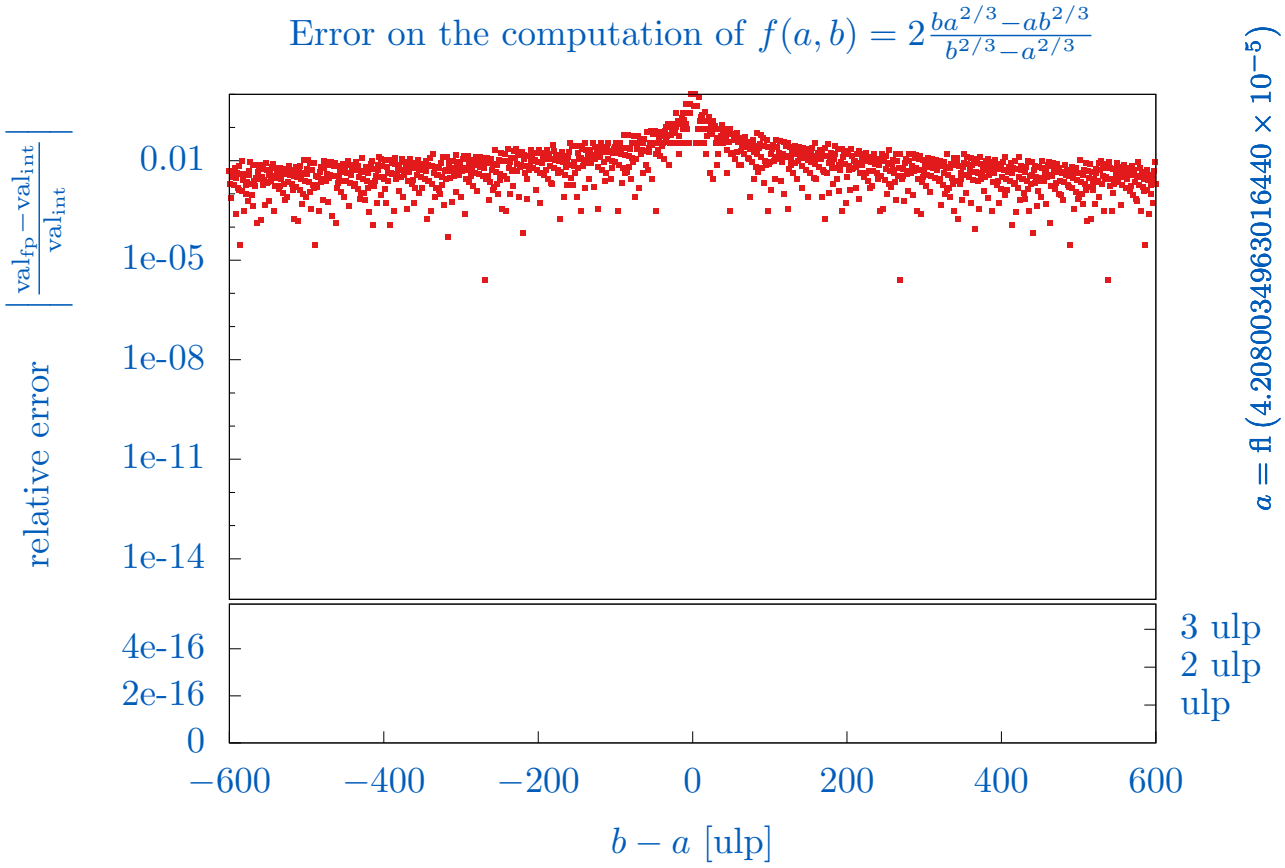


Bug fix example (2/3)

$$f(a,b) = \begin{cases} a & \text{if } a = b \\ 2\frac{ba^{2/3} - ab^{2/3}}{b^{2/3} - a^{2/3}} & \text{if not} \end{cases}$$

Empirical study

- outside the code
- around the problematic
- reference = interval arithmetic

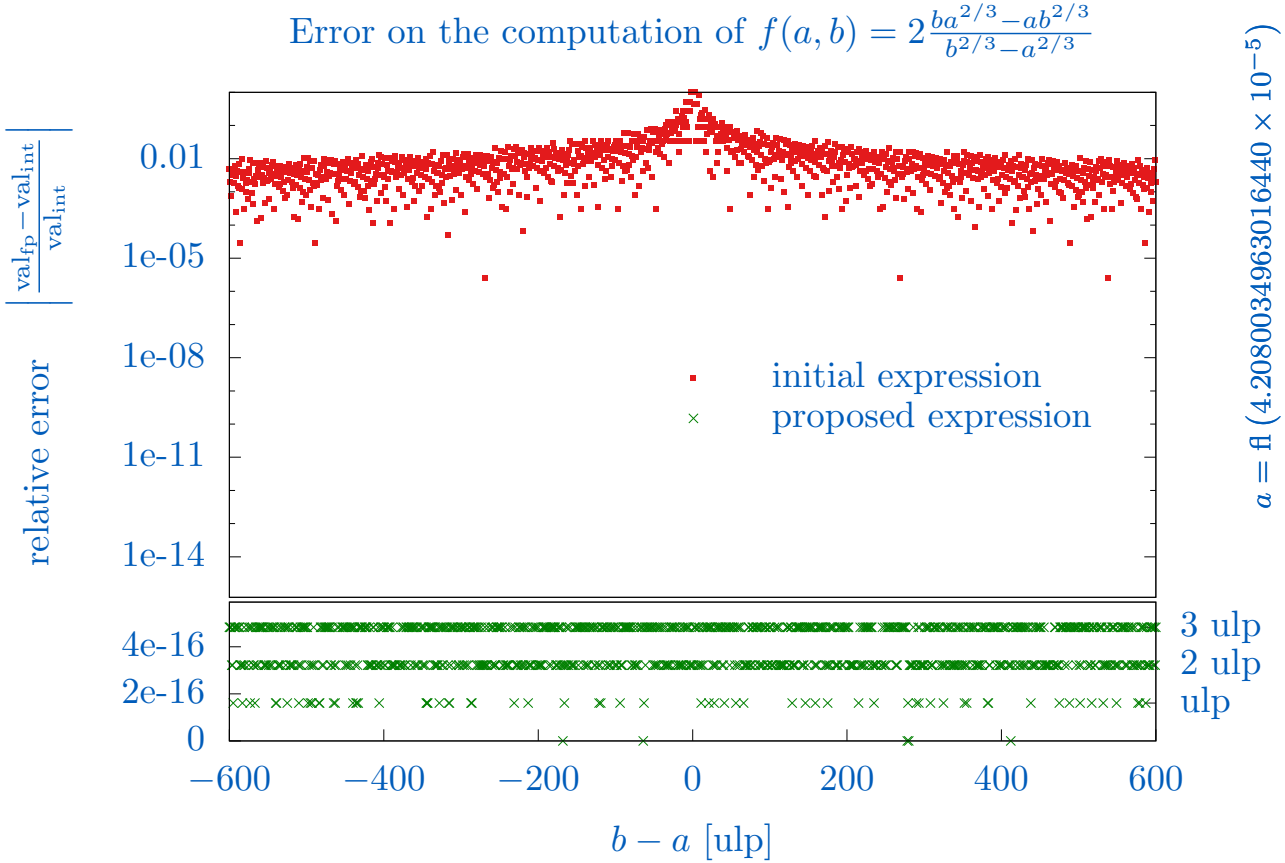


Bug fix example (2/3)

$$f(a,b) = \begin{cases} a & \text{if } a = b \\ 2\frac{ba^{2/3} - ab^{2/3}}{b^{2/3} - a^{2/3}} & \text{if not} \end{cases} \xrightarrow[\text{alpha}]{\text{wolfram}} f(a,b) = 2\frac{a^{2/3}b^{2/3}}{a^{1/3} + b^{1/3}}$$

Empirical study

- outside the code
- around the problematic
- reference = interval arithmetic

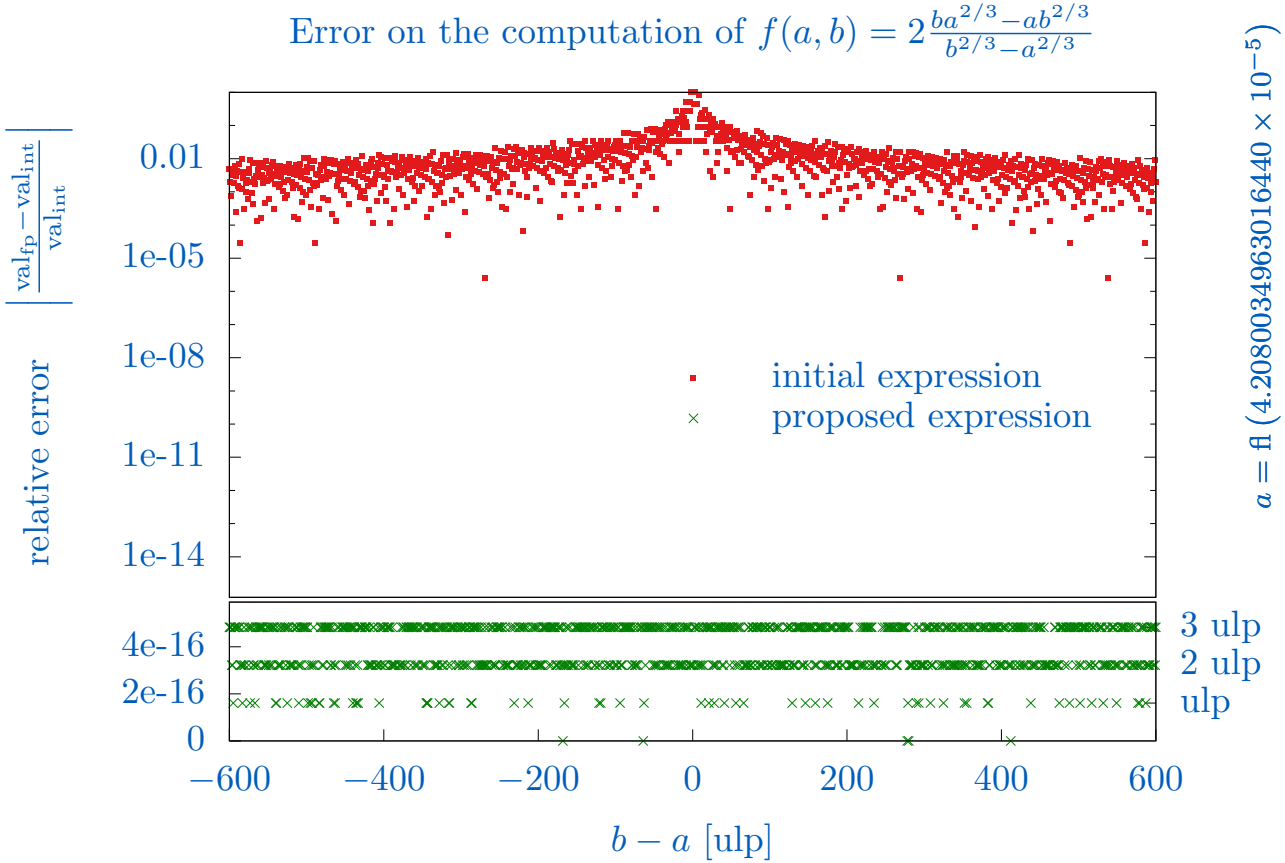


Bug fix example (2/3)

$$f(a,b) = \begin{cases} a & \text{if } a = b \\ 2\frac{ba^{2/3} - ab^{2/3}}{b^{2/3} - a^{2/3}} & \text{if not} \end{cases} \xrightarrow[\text{alpha}]{\text{wolfram}} f(a,b) = 2\frac{a^{2/3}b^{2/3}}{a^{1/3} + b^{1/3}}$$

Empirical study

- outside the code
- around the problematic
- reference = interval arithmetic



Bug fix example (3/3)

$$f_n(a,b) = \begin{cases} a & \text{si } a = b \\ (n-1)\dfrac{b^{\frac{1}{n}} - a^{\frac{1}{n}}}{a^{\frac{1}{n}-1} - b^{\frac{1}{n}-1}} & \text{sinon} \end{cases} \xrightarrow[\text{manual}]{\text{rewriting}} f_n(a,b) = \frac{n-1}{\sum_{i=1}^{n-1} a^{\frac{i-n}{n}} b^{\frac{-i}{n}}}$$