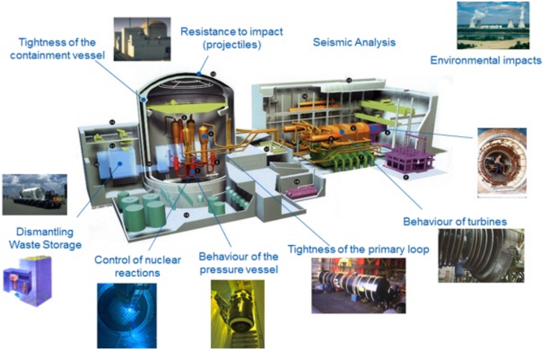
VERROU: panorama of localization method

CERN: Compute & Accelerator Forum
02/12/25
Bruno Lathuilière (EDF R&D)

Funded by ANR Interflop
ANR-20-CE46-0009.

Numerical simulation at EDF

- Guarantee safety
- Improve performances/costs
- Ageing issues


Tightness of the containment vessel
Resistance to impact (projectiles)
Seismic Analysis
Environmental impacts
Dismantling Waste Storage
Control of nuclear reactions
Behaviour of the pressure vessel
Tightness of the primary loop
Behaviour of turbines

Numerical software at EDF

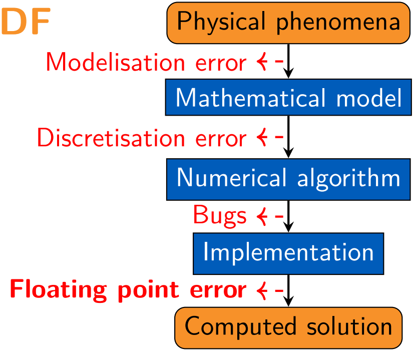
Large number of in-house codes:

- code_aster (Thermo-mechanic)
- code_saturn (CFD)
- open_telemac (free surface flow)
- salome (Simulation plateform)
- ...

Code properties:

- Huge code base
- Various languages (C/C++, Fortran, Python ...)
- External libraries (MUMPS, numpy ...)
- V&V

Numerical software at EDF



Physical phenomena

Modelisation error

Mathematical model

Discretisation error

Numerical algorithm

Bugs

Implementation

Floating point error

Computed solution

Large number of in-house codes:

- code_aster (Thermo-mechanic)
- code_saturn (CFD)
- open_telemac (free surface flow)
- salome (Simulation plateform)
- ...

Code properties:

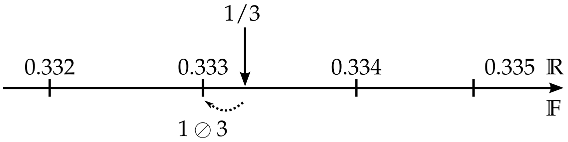
- Huge code base
- Various languages (C/C++, Fortran, Python ...)
- External libraries (MUMPS, numpy ...)
- **V&V**

VERROU development:

- Binary instrumentation based on valgrind
- Asynchronous stochastic arithmetic
- **Error estimation** and **error localization**

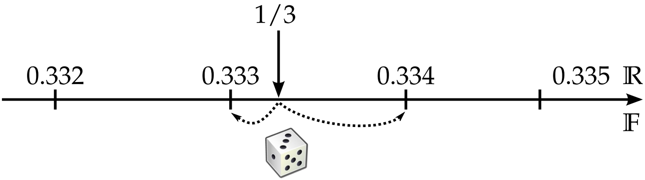
Floating point error

- 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 (% – ‰)

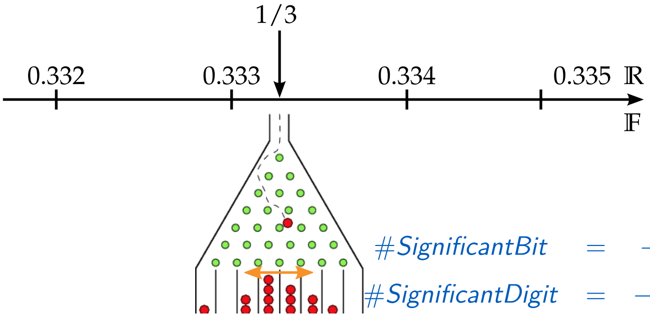


- Floating point computation $\neq$ Real computation
  - rounding error $a \oplus b \neq a + b$
  - associativity loss $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$

Stochastic arithmetic for numerical verification



Stochastic arithmetic for numerical verification



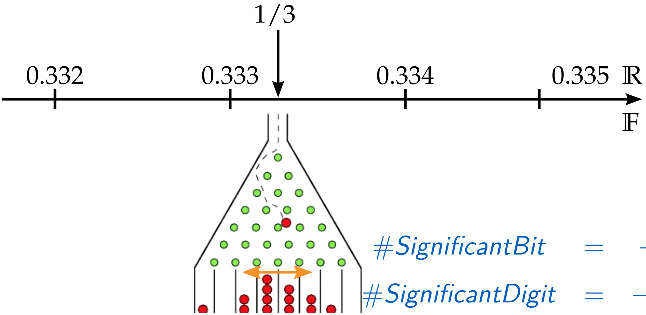
$$\#SignificantBit = -log2\left(\frac{max_i(|X_i - X_{nearest}|)}{|X_{nearest}|}\right)$$

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

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	$1.00_{\downarrow}$	$1.01^{\uparrow}$

$\#SignificantDigit \approx 1.95$

Stochastic arithmetic for numerical verification



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

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

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	$1.00_{\downarrow}$	$1.01^{\uparrow}$

$\#SignificantDigit \approx 1.95$

- Easily compatible with binary instrumentation (**Verrou** based on valgrind) or low-level LLVM pass (Verificarlo)
- Few false positive detection (due to asynchronous approach and dedicated stochastic rounding mode)

Example : rectangle method

$$\int_a^b f(x)dx \approx \sum_i^n \frac{(b-a)}{n} F(X(i)) \quad \text{avec} \quad X(i) = a + (i + 0.5)\frac{(b-a)}{n}$$

Listing – integrate.hxx

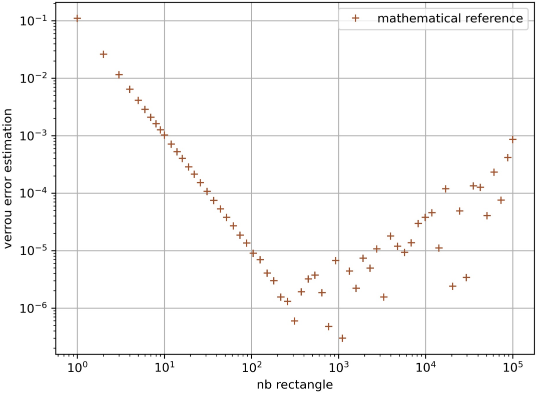
```
template <typename T, class REALTYPE>
REALTYPE integrate (const T& f, REALTYPE a, REALTYPE b, unsigned int n) {
  // Integration step
  const REALTYPE dx = (b - a) / n;
  // Naive integration Loop
  REALTYPE sum = 0.;
  for (REALTYPE x = a + (REALTYPE)0.5 * dx ;
       x < b ;
       x += dx) {
    sum += dx * f(x);
  }
  return sum;
}
```

Listing – unitTest.cxx (Verification: $\int_0^{\frac{\Pi}{2}} cos(x)dx = 1$)

```
void testConvergence (const float & step) {
  std::cout << std::scientific << std::setprecision(17);
  const size_t maxN= 100000;
  for (unsigned int n = 1 ; n <= maxN ;
       n = std::max (std::min(maxN, (size_t)(step*n)), n+1)) {

    float res = integrate (std::cos,0., M_PI_2, n);

    float err = std::abs(1 - res);

    // 3 columns output: Nrectangles Result Error
    std::cout << std::setw(10) << n << " " << res << " " << err << std::endl;
  }
}
```

Example: convergence analysis

```
     1 1.11072075366973877e+00 1.10720753669738770e-01
    10 1.00102877616882324e+00 1.02877616882324219e-03
   100 1.00001001358032227e+00 1.00135803222656250e-05
  1000 9.99992489814758301e-01 7.51018524169921875e-06
 10000 1.00000333786010742e+00 3.33786010742187500e-06
100000 9.99139010906219482e-01 8.60989093780517578e-04
```



Example: manual analysis

```
$ ./unitTest > unittest.output.txt
```

unittest.output.txt

```
     1  1.11072075366973877e+00  1.10720753669738770e-01
    10  1.00102877616882324e+00  1.02877616882324219e-03
   100  1.00001001358032227e+00  1.00135803222656250e-05
  1000  9.99992489814758301e-01  7.51018524169921875e-06
 10000  1.00000333786010742e+00  3.33786010742187500e-06
100000  9.99139010906219482e-01  8.60989093780517578e-04
```

Example: manual analysis

```
$ ./unitTest > unittest.output.txt
$ valgrind –tool=verrou –rounding-mode=random –libm=instrumented ./unitTest >
unittest.output-random_1.txt
```

```
unittest.output.txt
     1  1.11072075366973877e+00  1.10720753669738770e-01
    10  1.00102877616882324e+00  1.02877616882324219e-03
   100  1.00001001358032227e+00  1.00135803222656250e-05
  1000  9.99992489814758301e-01  7.51018524169921875e-06
 10000  1.00000333786010742e+00  3.33786010742187500e-06
100000  9.99139010906219482e-01  8.60989093780517578e-04
```

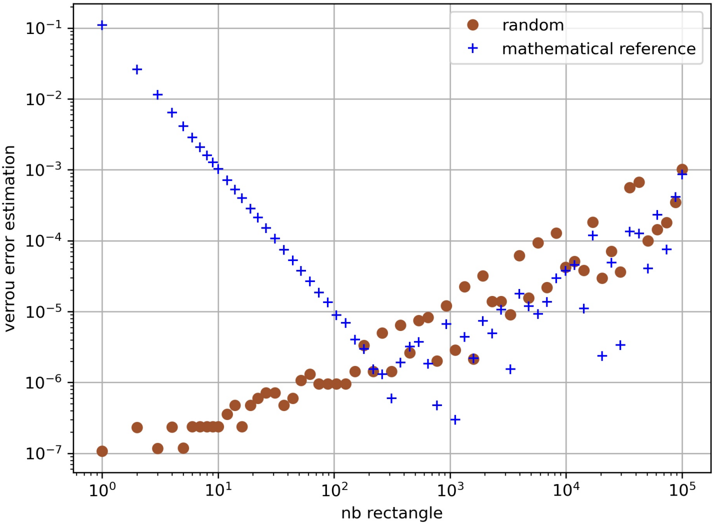
```
unittest.output_random_1.txt
     1  1.11072075366973877e+00  1.10720753669738770e-01
    10  1.00102853775024414e+00  1.02853775024414062e-03
   100  1.00001108646392822e+00  1.10864639282226562e-05
  1000  1.00000536441802979e+00  5.36441802978515625e-06
 10000  1.00001549720764160e+00  1.54972076416015625e-05
100000  1.00013923645019531e+00  1.39236450195312500e-04
```

Example: manual analysis

```
$ ./unitTest > unittest.output.txt
$ valgrind –tool=verrou –rounding-mode=random –libm=instrumented ./unitTest >
unittest.output-random_1.txt
$ meld unittest.output.txt unittest.output-random_1.txt
```

```
unittest.output.txt                                          | unittest.output_random_1.txt
     1  1.11072075366973877e+00  1.10720753669738770e-01     |      1  1.11072075366973877e+00  1.10720753669738770e-01
    10  1.00102877616882324e+00  1.02877616882324219e-03     |     10  1.00102853775024414e+00  1.02853775024414062e-03
   100  1.00001001358032227e+00  1.00135803222656250e-05     |    100  1.00001108646392822e+00  1.10864639282226562e-05
  1000  9.99992489814758301e-01  7.51018524169921875e-06     |   1000  1.00000536441802979e+00  5.36441802978515625e-06
 10000  1.00000333786010742e+00  3.33786010742187500e-06     |  10000  1.00001549720764160e+00  1.54972076416015625e-05
100000  9.99139010906219482e-01  8.60989093780517578e-04     | 100000  1.00013923645019531e+00  1.39236450195312500e-04
```

Error plot with 20 samples



Delta-debug: *trial and error* search algorithm

Delta-debug search:

```
verrou_dd_line --nruns=5 ddRun.sh ddCmp.py
```

cmpScript: `ddCmp.py`

runScript: `ddRun.sh`

```
#!/bin/bash
OUTDIR=$1
valgrind --tool=verrou --rounding-mode=random --libm=instrumented\
    ./unitTest >${OUTDIR}/res.dat
```

```
def extract(rep):
    lines=(open(os.path.join(rep, "res.dat"))).readlines()
    return re.split(" ",lines[-1].strip())[1]

if __name__=="__main__":
    if len(sys.argv)==2: #extract for verrou_plot_stat
        print(extract(sys.argv[1]))
    if len(sys.argv)==3: #cmp for verrou_dd_*
        refValue=float(extract(sys.argv[1]))
        value=float(extract(sys.argv[2]))
        relDist= abs((value -refValue)/refValue)
        if relDist < 1e-5: sys.exit(0)
        else:sys.exit(1)
```

ddmin	filename:line	demangled symbol name
ddmin0	`integrate.hxx:17`	testConvergence(float const &)
ddmin1	`integrate.hxx:16`	testConvergence(float const &)

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);

Code coverage for unstable branches detection

nearest coverage		random coverage	
-	`:8:template <typename T, class REALTYPE>`	-	`:8:template <typename T, ✂`
6	`:9:REALTYPE integrate (const T& f, REALTYPE a, ↩ REALTYPE b, unsigned int n) {`	6	`:9:REALTYPE integrate (co✂`
-	`:10:  // Integration step`	-	`:10:  // Integration step✂`
6	`:11:  const REALTYPE dx = (b - a) / n;`	6	`:11:  const REALTYPE dx =✂`
-	`:12:  // Naive integration Loop`	-	`:12:  // Naive integratio✂`
6	`:13:  REALTYPE sum = 0.;`	6	`:13:  REALTYPE sum = 0.;`
6	`:14:  for (REALTYPE x = a + (REALTYPE)0.5 * dx ;`	6	`:14:  for (REALTYPE x = a✂`
111**000**	`:15:      x < b ;`	111**179**	`:15:      x < b ;`
11**0994**	`:16:      x += dx) {`	11**1173**	`:16:      x += dx) {`
11**0994**	`:17:    sum += dx * f(x);`	11**1173**	`:17:    sum += dx * f(x);✂`
-	`:18:  }`	-	`:18:  }`
-	`:19:  return sum;`	-	`:19:  return sum;`
-	`:20:}`	-	`:20:}`

- Need to recompile with coverage option (`-fcoverage`).
- Need to rerun delta-debug.
- Need time to interpret results due to false-positive.

BasicBloc coverage

ddmin1 (integrate.hxx:16) | random		nearest	
1	:?iostream(74)unitTest.cxx(65)	1	:?iostream(74)unitTest.cxx(65)
1	:?unitTest.cxx(65)iostream(74)	1	:?unitTest.cxx(65)iostream(74)
11**1168**	:?cmath(185)	11**0988**	:?cmath(185)
6	:?integrate.hxx(15)cmath(185)	6	:?integrate.hxx(15)cmath(185)
11**1174**	:?integrate.hxx(15-17) F?	11**0994**	:?integrate.hxx(15-17) F?
4	:?iomanip(240)integrate.hxx(11,13-15) F?	4	:?iomanip(240)integrate.hxx(11,13-15) F?
1	:?iomanip(240)stl_algobase.h(237) ↩ integrate.hxx(11,13-15) F? ostream(196) F ios_base.h(84,88,731)integrate.hxx(11,13-15) F?	1	:?iomanip(240)stl_algobase.h(237) ↩ integrate.hxx(11,13-15) F? ostream(196) F ios_base.h(84,88,731)integrate.hxx(11,13-15) F?
	✂23 lines skipped ✂		

ddmin0 (integrate.hxx:17) | random		nearest	
1	:?iostream(74)unitTest.cxx(65)	1	:?iostream(74)unitTest.cxx(65)
1	:?unitTest.cxx(65)iostream(74)	1	:?unitTest.cxx(65)iostream(74)
110988	:?cmath(185)	110988	:?cmath(185)
6	:?integrate.hxx(15)cmath(185)	6	:?integrate.hxx(15)cmath(185)
110994	:?integrate.hxx(15-17) F?	110994	:?integrate.hxx(15-17) F?
4	:?iomanip(240)integrate.hxx(11,13-15) F?	4	:?iomanip(240)integrate.hxx(11,13-15) F?
1	:?iomanip(240)stl_algobase.h(237) ↩ integrate.hxx(11,13-15) F? ostream(196) F ios_base.h(84,88,731)integrate.hxx(11,13-15) F?	1	:?iomanip(240)stl_algobase.h(237) ↩ integrate.hxx(11,13-15) F? ostream(196) F ios_base.h(84,88,731)integrate.hxx(11,13-15) F?
	✂23 lines skipped ✂		

- Can be generated automatically with `post_verrou_dd`

Example : come back to code

Listing – integrate.hxx

```
template <typename T, class REALTYPE>
REALTYPE integrate (const T& f, REALTYPE a, REALTYPE b, unsigned int n) {
  // Integration step
  const REALTYPE dx = (b - a) / n;
  // Naive integration Loop
  REALTYPE sum = 0.;
  for (REALTYPE x = a + (REALTYPE)0.5 * dx ;
       x < b ;
       x += dx) {
    sum += dx * f(x);
  }
  return sum;
}
```

- Convert the floating point loop ($x+=dx$) into an integer loop ($i++$).
- Accumulation in double (*mixed precision*) or compensated algorithm.

Partial BasicBloc coverage

ddmin1 (integrate.hxx:16) | random		nearest	
1	:?iostream(74)unitTest.cxx(65)	1	:?iostream(74)unitTest.cxx(65)
1	:?unitTest.cxx(65)iostream(74)	1	:?unitTest.cxx(65)iostream(74)
11**1168**	:?cmath(185)	11**0988**	:?cmath(185)
6	:?integrate.hxx(15)cmath(185)	6	:?integrate.hxx(15)cmath(185)
	integrate.hxx(11,13-15) F?		integrate.hxx(11,13-15) F?
	ostream(196) F		ostream(196) F
	ios_base.h(84,88,731)integrate.hxx(11,13-15) F?		ios_base.h(84,88,731)integrate.hxx(11,13-15) F?

✂26 lines skipped ✂

partial (last iter) ddmin1 (integrate.hxx.16) | random		partial (last iter) | nearest	
100062	:?cmath(185)	**99883**	:?cmath(185)
1	:?integrate.hxx(15)cmath(185)	1	:?integrate.hxx(15)cmath(185)
100063	:?integrate.hxx(15-17) F?	**99884**	:?integrate.hxx(15-17) F?
1	:?iomanip(240)stl_algobase.h(237) ↩ integrate.hxx(11,13-15) F? ostream(196) F	1	:?iomanip(240)stl_algobase.h(237) ↩ integrate.hxx(11,13-15) F? ostream(196) F

✂11 lines skipped ✂

How to specify coverage point

- client request : `VERROU_DUMP_COVER`
- IOMatch script

```
cmatch:     100000 * *
apply: dump_cover
cmatch:      10000 * *
apply: dump_cover
```

Temporal Delta-debug

- verrou_dd_task (with manual definition in source or automatic for python line)
- verrou_dd_stdout (task automatically detected thanks to IOMatch)

```
verrou_dd_stdout --nruns=5 ddRun.sh ddCmp.py
```

```
__verrou__stdout__init__
        1 * *
       10 * *
      100 * *
     1000 * *
    10000 * *
   100000 * *
```

(a) Search space.

ddmin	match line
ddmin0	10000 * *

(b) Result.

- Warning : pay attention to bufferisation.
- It is possible to modify stdout (and so task) thanks a user script.

Mixed precision search for inner/outer iteration

```
CMD="etest2 100 100 1 res.dat rh.dat
          -e ii -i bicgstab -p ilu ↩
               -ilu_fill 2
          -print out -eprint out"
valgrind --tool=verrou
         --float=yes
         --exclude=exclude.ex
         $CMD > $1/res.out
```

```
> cat exclude.ex
lis_vector_dot *
```

```
 ...
iteration:    47  relative residual = 2.102138E-10
iteration:    48  relative residual = 2.340582E-11
iteration:    49  relative residual = 7.864099E-12
iteration:    50  relative residual = 4.231160E-12
iteration:    51  relative residual = 2.719284E-12
iteration:    52  relative residual = 8.079880E-13
linear solver status  : normal end

iteration:     2  relative residual = 7.941629E-02
initial vector x      : all components set to 0
precision             : double
linear solver         : BiCGSTAB
preconditioner        : ILU(2)
convergence condition : ||b-Ax||_2 <= 1.0e-12 * ||b-Ax_0||_2
matrix storage format : CSR
iteration:     1  relative residual = 9.351250E-02
iteration:     2  relative residual = 6.529751E-02
iteration:     3  relative residual = 2.403754E-02
iteration:     4  relative residual = 1.016325E-02
...
Inverse: mode number          = 0
Inverse: eigenvalue           = 1.934862e-03
Inverse: number of iterations = 1000
Inverse: elapsed time         = 3.840000e+02 sec.
Inverse:   preconditioner     = 1.280000e+02 sec.
Inverse:     matrix creation  = 0.000000e+00 sec.
Inverse:   linear solver      = 2.560000e+02 sec.
Inverse: relative residual    = 3.699645e-06
```

search space	ddmin subset	
	all	all \{dot}
outer-0 matrix storage format : CSR	ddmin15	OK
preconditioner : ILU	OK	OK
initial vector x : all components set to 0	OK	OK
outer-0 iteration: 10 relative residual = *	ddmin15	OK
outer-0 iteration: 20 relative residual = *	OK	OK
outer-0 iteration: 30 relative residual = *	OK	OK
outer-0 iteration: 40 relative residual = *	OK	OK
outer-0 linear solver status : normal end	OK	OK
outer iteration: 0 relative residual = *	OK	OK
outer-5 matrix storage format : CSR	ddmin0	ddmin0
outer-5 iteration: 10 relative residual = *	ddmin1	ddmin1
outer-5 iteration: 20 relative residual = *	ddmin16	ddmin2
outer-5 iteration: 30 relative residual = *	ddmin14	OK
outer-5 iteration: 40 relative residual = *	ddmin16	OK
outer-5 linear solver status : normal end	OK	OK
outer iteration: 5 relative residual = *	OK	OK
outer-10 matrix storage format : CSR	ddmin2	ddmin3
outer-10 iteration: 10 relative residual = *	ddmin3	ddmin4
outer-10 iteration: 20 relative residual = *	ddmin4	ddmin5
outer-10 iteration: 30 relative residual = *	ddmin5	ddmin6
outer-10 iteration: 40 relative residual = *	OK	OK
outer-10 linear solver status : normal end	OK	OK
outer iteration: 10 relative residual = *	ddmin6	ddmin7
outer-15 matrix storage format : CSR	ddmin7	ddmin8
outer-15 iteration: 10 relative residual = *	ddmin8	ddmin9
outer-15 iteration: 20 relative residual = *	ddmin9	ddmin10
outer-15 iteration: 30 relative residual = *	ddmin10	ddmin11
outer-15 linear solver status : normal end	ddmin11	ddmin12
outer iteration: 15 relative residual = *	ddmin12	ddmin13
outer-15 iteration: 40 relative residual = *	ddmin13	ddmin14

Interflop verification tools

Verrou is not the silver bullet tool. As a user I'm interested in other Interflop verification tools :

- **Verificarlo** `https://github.com/verificarlo/verificarlo`
  - Computation time (especially with share memory)
  - Better Debugger compatibility
- **Pene** `https://github.com/aneoconsulting/PENE`
  - Windows Portability
  - avx512
- **CADNA** `https://cadna.lip6.fr/`
  - Algorithm which take into account accuracy estimation
  - Useful to perform synchronous/asynchronous comparison.
- **FLDLib** `https://github.com/fvedrine/fldlib`
  - High level of confidence for tricky kernel.

Conclusions and perspectives

- VERROU provides an easy floating point error estimation ;
- VERROU provides error localization ;
- VERROU is used with industrial code ;
- VERROU is open-source and available: `https://github.com/edf-hpc/verrou`

Perspectives

- Performance improvements (instrumentation, vectorization, delta-debug parallelism . . .)
- Localization improvements (delta-debug search space, amplification detection . . .)

Question?

Performance

type	double		float	
compilation option	O0	O3	O0	O3
tool_none	x6.3	x6.4	x7.0	x8.4
nearest	x10.0	x22.3	x10.6	x27.9
nearest-nc	x10.1	x21.8	x10.4	x27.2
random	x15.4	x41.2	x17.7	x54.1
average	x17.4	x47.5	x21.0	x66.0
random_det	x16.6	x46.0	x19.8	x64.7
random_comdet	x16.8	x47.2	x20.2	x66.8
random_scomdet	x19.4	x56.2	x23.4	x79.8
average_det	x19.2	x53.1	x23.5	x77.5
average_comdet	x20.2	x56.5	x24.9	x85.3
average_scomdet	x20.0	x56.7	x25.3	x85.4
sr_monotonic	x20.4	x57.4	x25.0	x81.5
sr_smonotonic	x20.5	x57.0	x25.3	x83.1

Number of samples

Confidence level $1-\alpha$	Probability p								
	0.66	0.75	0.8	0.85	0.9	0.95	0.99	0.995	0.999
0.66	3	4	5	7	11	22	108	216	1079
0.75	4	5	7	9	14	28	138	277	1386
0.8	4	6	8	10	16	32	161	322	1609
0.85	5	7	9	12	19	37	189	379	1897
0.9	6	9	11	15	22	45	230	460	2302
0.95	8	11	14	19	29	59	299	598	2995
0.99	12	17	21	29	44	90	459	919	4603
0.995	13	19	24	33	51	104	528	1058	5296
0.999	17	25	31	43	66	135	688	1379	6905

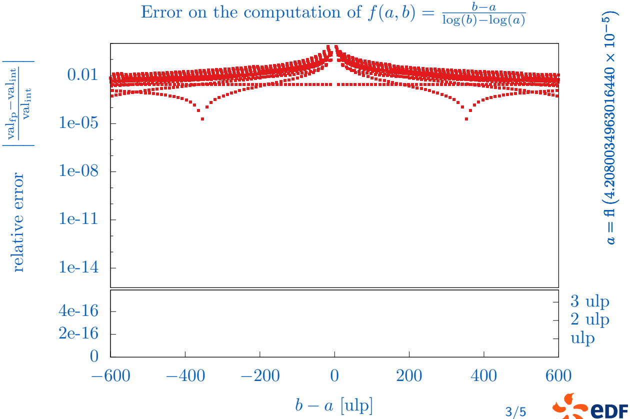
Confidence Intervals for Stochastic Arithmetic, Devan Sohier, Pablo De Oliveira Castro, François Févotte, Bruno Lathuilière, Eric Petit, Olivier Jamond

Bug fix example (1/3)

$$f(a,b) = \left| \begin{array}{ll} a & \text{if } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{if not} \end{array} \right.$$

Empirical study

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

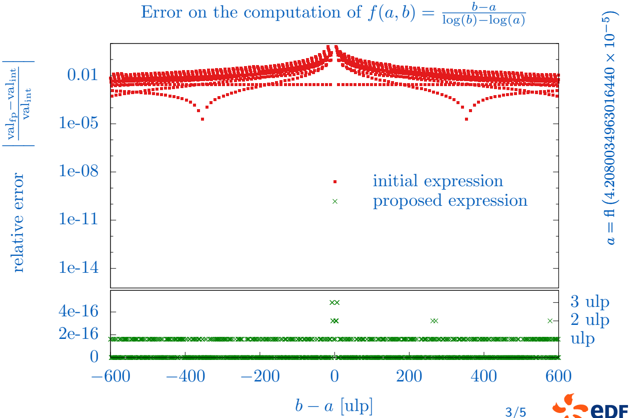


Bug fix example (1/3)

$$f(a,b) = \left|\begin{array}{ll} a & \text{if } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{if not} \end{array}\right. \quad \xrightarrow[\text{manual}]{\text{rewritting}} \quad f(a,b) = \left|\begin{array}{ll} a & \text{if } a = b \\ a\,\frac{\frac{b}{a}-1}{\log(\frac{b}{a})} & \text{if not} \end{array}\right.$$

Empirical study

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



Bug fix example (1/3)

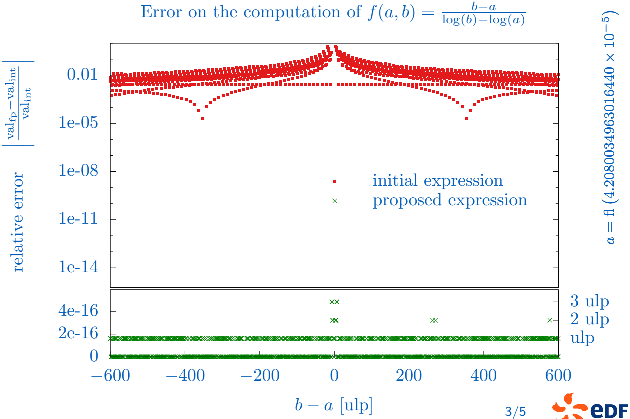
$$f(a,b) = \left|\begin{array}{ll} a & \text{if } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{if not} \end{array}\right. \xrightarrow[\text{manual}]{\text{rewritting}} f(a,b) = \left|\begin{array}{ll} a & \text{if } a = b \\ a\,\frac{\frac{b}{a}-1}{\log(\frac{b}{a})} & \text{if not} \end{array}\right.$$

Empirical study

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

Proof

- error bounded by 10 ulps



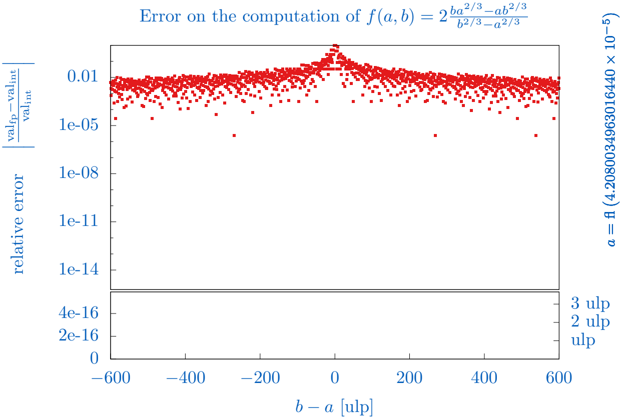
Error on the computation of $f(a,b) = \frac{b-a}{\log(b)-\log(a)}$

Bug fix example (2/3)

$$f(a,b) = \left| \begin{array}{ll} a & \text{if } a = b \\ 2\frac{ba^{2/3} - ab^{2/3}}{b^{2/3} - a^{2/3}} & \text{if not} \end{array} \right.$$

Empirical study

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

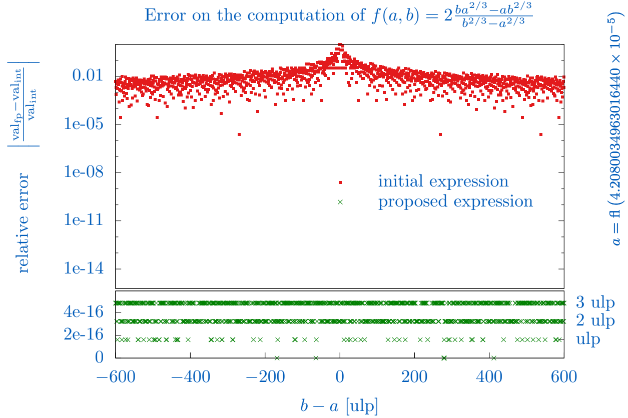


Bug fix example (2/3)

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

Empirical study

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



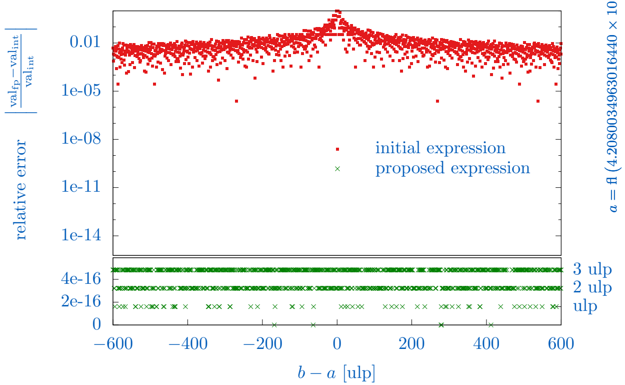
Bug fix example (2/3)

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

Empirical study

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

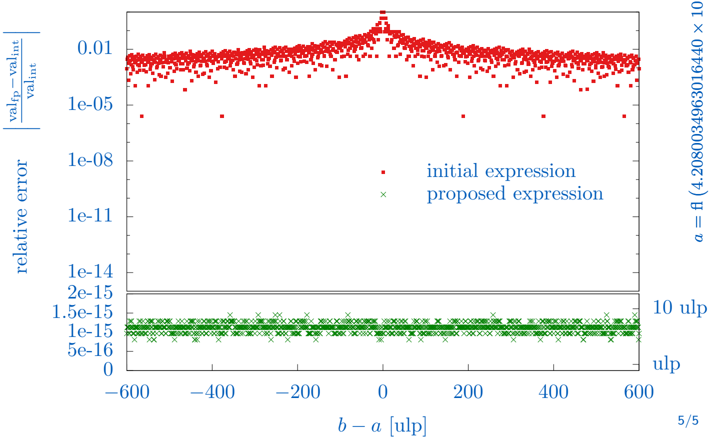
Error on the computation of $f(a,b) = 2\frac{ba^{2/3}-ab^{2/3}}{b^{2/3}-a^{2/3}}$



Bug fix example (3/3)

$$f_n(a,b) = \left| \begin{array}{ll} a & \text{if } a=b \\ (n-1)\frac{b^{\frac{1}{n}}-a^{\frac{1}{n}}}{a^{\frac{1}{n}-1}-b^{\frac{1}{n}-1}} & \text{if not} \end{array} \right. \xrightarrow[\text{rewriting}]{\text{manual}} \quad f_n(a,b) = \frac{n-1}{\sum_{i=1}^{n-1} a^{\frac{i-n}{n}} b^{\frac{-i}{n}}}$$

Error on the computation of $f_n(a,b) = \frac{(n-1)(b^{1/n}-a^{1/n})}{a^{1/n-1}-b^{1/n-1}}$ with $n=7$



relative error $\left|\frac{\mathrm{val_{fp}}-\mathrm{val_{int}}}{\mathrm{val_{int}}}\right|$
0.01
1e-05
1e-08
1e-11
1e-14
2e-15
1.5e-15
1e-15
5e-16
0
10 ulp
ulp
−600
−400
−200
0
200
400
600
$b-a$ [ulp]
$a = \mathrm{fl}(4.2080034963016440 \times 10^{-5})$
initial expression
proposed expression