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 ($\approx 10^{-7}$)
  - [double] binary, 53 significand bits ($\approx 10^{-16}$)
  - [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: a valgrind tool dedicated to floating point error diagnosis
Mathematical background: stochastic arithmetic

$mathbf{Eval.\ Nearest}$

$a = \frac{1}{3} 
\begin{array}{l}
0.332 \\
0.333 \\
0.334 \\
0.335
\end{array}
$

$b = a \times 3
\begin{array}{l}
0.999 \\
1.000 \\
1.010 \\
\# \text{ Significant Digit} \\
\approx 1.000
\end{array}$

Verrou: a valgrind tool dedicated to floating point error diagnosis
Mathematical background: stochastic arithmetic

\[ \text{\#SignificantBit} = -\log_2 \left( \frac{\max_i |X_i - X_{\text{nearest}}|}{|X_{\text{nearest}}|} \right) \]

\[ \text{\#SignificantDigit} = -\log_{10} \left( \frac{\max_i |X_i - X_{\text{nearest}}|}{|X_{\text{nearest}}|} \right) \]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Eval. Nearest</th>
<th>Eval. 1</th>
<th>Eval. 2</th>
<th>Eval. 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a = 1/3 )</td>
<td>0.333</td>
<td>0.333↓</td>
<td>0.334↑</td>
<td>0.334↑</td>
</tr>
<tr>
<td>( b = a \times 3 )</td>
<td>0.999</td>
<td>0.999</td>
<td>1.00↓</td>
<td>1.01↑</td>
</tr>
</tbody>
</table>

\[ \text{\#SignificantDigit} \approx 1.95 \]
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.

Verrou: a valgrind tool dedicated to floating point error diagnosis
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 << " x2: " << x2 << " temp0: " << temp0 << " temp1: " << temp1 << " temp2: " << temp2 << " addr: " << &temp2 << endl;
12       }
13       x0 = x1;
14       x1 = x2;
15    }
16    std::cout << "x[" << nt << "]" = " << x1 << std::endl;
17    return x1;
18 }

begin iter
    it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.41 addr: 0x7ffe0b19328
    it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367 addr: 0x7ffe0b19328
    it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325 addr: 0x7ffe0b19328
    it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287 addr: 0x7ffe0b19328
    it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251 addr: 0x7ffe0b19328
    it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218 addr: 0x7ffe0b19328
    it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189 addr: 0x7ffe0b19328
    it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.31 temp2: 105.162 addr: 0x7ffe0b19328
    it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139 addr: 0x7ffe0b19328
    it: 9 x2: 5.88354 temp0: 513.904 temp1: 616.096 temp2: 105.116 addr: 0x7ffe0b19328
    it: 10 x2: 5.93596 temp0: 511.851 temp1: 618.149 temp2: 105.064 addr: 0x7ffe0b19328
    it: 11 x2: 6.53442 temp0: 509.897 temp1: 620.103 temp2: 104.466 addr: 0x7ffe0b19328
x[12]=6.53442

Verrou: a valgrind tool dedicated to floating point error diagnosis
Muller example

```cpp
float muller(int nt, bool verbose=false){
    float x0 = 11./2.;
    float x1 = 61./11.;
    std::cout << "begin iter" << std::endl;
    for(size_t it=0; it < nt; it++){
        float temp0 = 3000./x0;
        float temp1 = 1130. - temp0;
        float temp2 = temp1 /x1 ;
        float x2 = 111. - temp2;
        if( verbose ){
            cout <<"it : " << it << " \tx2 : " <<x2
            << "\ttemp0 : " <<temp0 <<"\ttemp1 : " <<temp1<<"\ttemp2 : " <<temp2 <<"addr : " <<&temp2 << endl;
        }
        x0 = x1;
        x1 = x2;
    }
    std::cout <<"x["<<nt<<"]=" <<x1<<std::endl;
    return x1;
}
```

begin iter
    it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.41addr: 0x7ffe0b19328
    it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367addr: 0x7ffe0b19328
    it: 2 x2: 5.67465 temp0: 536.657 temp1: 593.343 temp2: 105.325addr: 0x7ffe0b19328
    it: 3 x2: 5.71333 temp0: 532.535 temp1: 597.465 temp2: 105.287addr: 0x7ffe0b19328
    it: 4 x2: 5.74912 temp0: 528.667 temp1: 601.333 temp2: 105.251addr: 0x7ffe0b19328
    it: 5 x2: 5.78181 temp0: 525.088 temp1: 604.912 temp2: 105.218addr: 0x7ffe0b19328
    it: 6 x2: 5.81131 temp0: 521.819 temp1: 608.181 temp2: 105.189addr: 0x7ffe0b19328
    it: 7 x2: 5.83766 temp0: 518.869 temp1: 611.131 temp2: 105.162addr: 0x7ffe0b19328
    it: 8 x2: 5.86108 temp0: 516.234 temp1: 613.766 temp2: 105.139addr: 0x7ffe0b19328
    it: 9 x2: 5.88354 temp0: 513.904 temp1: 616.096 temp2: 105.116addr: 0x7ffe0b19328
    it: 10 x2: 5.93596 temp0: 511.851 temp1: 618.149 temp2: 105.064addr: 0x7ffe0b19328
    it: 11 x2: 6.53442 temp0: 509.897 temp1: 620.103 temp2: 104.466addr: 0x7ffe0b19328
x[12]=6.53442

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

Verrou: a valgrind tool dedicated to floating point error diagnosis
Delta-debug: *trial and error* search algorithm

runScript: `ddRun.sh`
```
#!/bin/bash
PREFIX="valgrind --tool=verrou --rounding-mode=random"
$PREFIX ./muller -v > $1/res.dat
```

cmpScript: `extractOrCmp.py`
```
#!/usr/bin/python3

def extractValue(rep):
    for line in open(os.path.join(rep, "res.dat")):
        if line.startswith("x[12]="):
            return float(line.partition("=")[2])

if __name__=='__main__':
    if len(sys.argv)==2:
        print(extractValue(sys.argv[1]))
    if len(sys.argv)==3:
        valueRef=extractValue(sys.argv[1])
        value=extractValue(sys.argv[2])
        relDiff=abs((value-valueRef)/valueRef)
        if relDiff < 1.e-2:
            sys.exit(0)
        else:
            sys.exit(1)
```

Delta-debug search:
```
verrou_dd_line --nruns=5 ddRun.sh extractOrCmp.py
```

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).

Valrou: a valgrind tool dedicated to floating point error diagnosis
Towards temporal localisation
New search space: wildcarded IO (automatic thanks regexp or manually defined)

```plaintext
begin iter
it: 0 x2: * temp0: * temp1: * temp2: *addr: 0x?????????????
it: 1 x2: * temp0: * temp1: * temp2: *addr: 0x?????????????
...```

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

```plaintext
ddmin0 (|begin iter|):
...```

```plaintext
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*

Verrou: a valgrind tool dedicated to floating point error diagnosis
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.
Available on github:
http://github.com/edf-hpc/verrou

Documentation:

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.


## Performance

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

<table>
<thead>
<tr>
<th>type</th>
<th>compilation option</th>
<th>double</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>O0</td>
<td>O3</td>
</tr>
<tr>
<td>nearest</td>
<td></td>
<td>×12.2</td>
<td>×26.2</td>
</tr>
<tr>
<td>tool_none</td>
<td></td>
<td>×13.9</td>
<td>×55.7</td>
</tr>
<tr>
<td>fma_only</td>
<td></td>
<td>×7.1</td>
<td>×13.4</td>
</tr>
<tr>
<td>random</td>
<td></td>
<td>×19.1</td>
<td>×53.8</td>
</tr>
<tr>
<td>random_det</td>
<td></td>
<td>×19.6</td>
<td>×51.9</td>
</tr>
<tr>
<td>random__comdet</td>
<td></td>
<td>×20.1</td>
<td>×55.7</td>
</tr>
<tr>
<td>random__scomdet</td>
<td></td>
<td>×23.8</td>
<td>×70.1</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td>×22.3</td>
<td>×60.8</td>
</tr>
<tr>
<td>average__det</td>
<td></td>
<td>×23.9</td>
<td>×66.0</td>
</tr>
<tr>
<td>average__comdet</td>
<td></td>
<td>×24.8</td>
<td>×69.6</td>
</tr>
<tr>
<td>average__scomdet</td>
<td></td>
<td>×25.5</td>
<td>×72.4</td>
</tr>
<tr>
<td>sr_monotonic</td>
<td></td>
<td>×23.8</td>
<td>×67.2</td>
</tr>
<tr>
<td>sr_smonotonic</td>
<td></td>
<td>×24.0</td>
<td>×68.2</td>
</tr>
</tbody>
</table>

**Verrou: a valgrind tool dedicated to floating point error diagnosis**
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:

ansi: 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

Verrou: a valgrind tool dedicated to floating point error diagnosis
expect-clr format

The option-\texttt{expect-clr=EXPECT\_FILE} (or environnement variable \texttt{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 \texttt{begin:}.

In headers there are:

- setup keys: \texttt{verbose: \textit{LEVEL}}, \texttt{ignore-empty-line: \textit{}} , \texttt{filter\_line\_exec: \textit{CMD}}, \texttt{dump-stdout: \textit{[FILENAME]}}, \texttt{dump-filtered-stdout: \textit{[FILENAME]}}
- definition of specific action: \texttt{default: \textit{ACTION}}, \texttt{init: \textit{ACTION}}, \texttt{post\_init: \textit{ACTION}}.
- definition of match action: \texttt{match: \textit{PATTERN}}, \texttt{apply: \textit{ACTION}}, \texttt{post-apply: \textit{ACTION}}.

In temporal section there are:

- \texttt{except: \textit{PATTERN}}, \texttt{apply: \textit{ACTION}}.

The available actions are: \texttt{start}, \texttt{stop}, \texttt{nop}, \texttt{display\_counter}, \texttt{nb\_instr}, \texttt{reset\_counter}, \texttt{dump\_cover}, \texttt{panic}, \texttt{exit}, \texttt{default}, \texttt{init}, \texttt{post-init}
Histogram

Verrou: a valgrind tool dedicated to floating point error diagnosis
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
dbg
it: 0 x2: 5.59016 temp0: 545.455 temp1: 584.545 temp2: 105.41 addr: 0x7fffe0b19328
dbg
it: 1 x2: 5.63343 temp0: 540.984 temp1: 589.016 temp2: 105.367 addr: 0x7fffe0b19328
dbg...
```

```
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.

Verrou: a valgrind tool dedicated to floating point error diagnosis
Other Interflop tools based on stochastic arithmetics

- **CADNA**
  - url: 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

Error on the computation of \( f(a, b) = \frac{b - a}{\log(b) - \log(a)} \)
Bug fix example (1/3)

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

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

Empirical study

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

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

Verrou: a valgrind tool dedicated to floating point error diagnosis
**Bug fix example (1/3)**

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

\[
\begin{aligned}
\text{récriture} & \quad \text{manuelle} \\
\end{aligned}
\]

\[
f(a, b) = \begin{cases} 
  a & \text{si } a = b \\
  a \frac{\frac{b}{a} - 1}{\log\left(\frac{b}{a}\right)} & \text{sinon}
\end{cases}
\]

---

**Empirical study**

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

**Proof**

- error bounded by 10 ulps

---

Verrou: a valgrind tool dedicated to floating point error diagnosis
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

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 (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

Error on the computation of \( f(a, b) = 2 \frac{a^{2/3}b^{2/3}}{a^{1/3} + b^{1/3}} \)

Verrou: a valgrind tool dedicated to floating point error diagnosis
Bug fix example (2/3)

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

\[ f(a, b) = 2 \frac{a^{2/3} b^{2/3}}{a^{1/3} + b^{1/3}} \]

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

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

Verrou: a valgrind tool dedicated to floating point error diagnosis
Bug fix example (3/3)

\[ f_n(a, b) = \begin{cases} 
   a & \text{si } a = b \\
   (n-1) \frac{b^{\frac{1}{n}} - a^{\frac{1}{n}}}{a^{\frac{1}{n-1}} - b^{\frac{1}{n-1}}} & \text{sinon}
\end{cases} \]

rewriting\[ \Rightarrow \]

\[ f_n(a, b) = \frac{n-1}{\sum_{i=1}^{n-1} a^{\frac{i-n}{n}} b^{\frac{-i}{n}}} \]

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 \( n = 7 \)

\[ a = \text{fl}(4.2080034963016440 \times 10^{-5}) \]

Verrou: a valgrind tool dedicated to floating point error diagnosis