VeriT racer: Context-enriched tracer for floating-point arithmetic analysis

ARITH 25, Amherst MA USA, June 2018

Yohan Chatelain\textsuperscript{1,5} \hspace{1cm} Pablo de Oliveira Castro\textsuperscript{1,5} \hspace{1cm} Eric Petit\textsuperscript{2,5} \hspace{1cm} David Defour\textsuperscript{3}

Jordan Bieder\textsuperscript{4,5} \hspace{1cm} Marc Torrent\textsuperscript{4,5}

\textsuperscript{1}Université de Versailles Saint-Quentin-en-Yvelines (UVSQ) \hspace{1cm} \textsuperscript{2}Intel

\textsuperscript{3}Université de Perpignan Via Domitia (UPVD) \hspace{1cm} \textsuperscript{4}CEA, DAM, DIF

\textsuperscript{5}Exascale Computing Research (ECR)
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

<table>
<thead>
<tr>
<th>Method</th>
<th>Name</th>
<th>Implementation</th>
</tr>
</thead>
</table>
| 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) |

### Optimization

<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Mixed-Precision</th>
<th>Any-Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verificarlo [4]</td>
<td>MCA (User specific)</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>HPC Craft [10]</td>
<td>Bitmask (ref value)</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Promise [8]</td>
<td>CESTAC/DSA (Δ – debug)</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Herbie [12]</td>
<td>EP (Rewriting)</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Muller’s sequence[2] overview

\[ u_n = \begin{cases} 
  u_0 = 2 \\
  u_1 = -4 \\
  u_{n+1} = 111 - \frac{1130}{u_n} + \frac{3000}{u_n u_{n-1}} 
\end{cases} \]

**exact solution** \[ \lim_{n \to \infty} u_n = 6 \]

**finite precision**

**CADNA**
- \( u_{30} \)
- Mean: 100
- Significant digits: 16

**Verificarlo**
- \( u_{30} \)
- Mean: 100
- Significant digits: 16

Stochastic methods execute several samples by introducing random errors
Muller’s sequence[2] overview

Without an exact solution

- Only checking the final result may lead 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

⇒ Need to explore the temporal dimension
Veritracer
**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

### C Code

```c
void twosum(float a, float b, ...) 
{
  float x = a + b
  float z = x - a
  ...
}
```

### IR

```llvm
twoSum:  
  %1 = load float* %a  
  %2 = load float* %b  
  %3 = fadd %1, %2  
  store float %3, float* %x  
  %4 = load float* %x  
  %5 = load float* %a  
  %6 = fsub %1, %2  
  store float %6, float* %z
```

### Debug Info

```
name : x  
type : float  
function: twoSum  
file : twoSum.c  
line : 4
```

```
name : z  
type : float  
function: twoSum  
file : twoSum.c  
line : 5
```

### Instrumentation

```
twoSum:  
  %1 = load float* %a  
  %2 = load float* %b  
  %3 = fadd %1, %2  
  store float %3, float* %x  
  veritracer_probe(%3, x, ...)  
  %4 = load float* %a  
  %5 = load float* %b  
  %6 = load float* %x  
  %7 = fsub %4, %5  
  store float %6, float* %x  
  veritracer_probe(%6, x, ...)
```
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

\[ f_{\delta e_{\text{max}}}(x) \leq \epsilon, \]

**Order** functions by *numerical stress resistance*

\[ \min_{\delta e} \{ f_{\delta e_{\text{min}}} \leq \epsilon \} \]

---

**Diagram:**

- **Narrowing**:
  - \[ f_{\delta e_{\text{max}}}(x) \leq \epsilon \]
  - \[ f_{\delta e_{\text{max}}}(x) > \epsilon \]

- **Ordering**:
  - \[ \min_{\delta e} \{ f_{\delta e}(x) \leq \epsilon \} \]
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](image)
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
- \(\xi\) a random variable between \([-\frac{1}{2}, \frac{1}{2}]\)

Random Rounding mode

- \(RR(x \circ y) = \text{inexact}(x \circ y), \circ \in \{+, -, \times, \div\}\)
Experiments
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 ($\text{MgSiO}_3$ perovskite with Al impurities) [1]
Ordering the search space

- Functions below 23 bits
  - are more resistant
  - can potentially be changed to single precision

- Functions above 23 bits
  - 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 Simpson’s 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))
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

- 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
Available on github:
github.com/verificarlo/verificarlo
github.com/verificarlo/verificarlo/tree/veritracer
Random Rounding mode (MCA)

Monte Carlo Arithmetic [parker1997monte]

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

\[ t = 21 \]

FP operations \( \circ \) are replaced by:

\[ RR(x \circ y) = \text{round}(\text{inexact}(x \circ y)) \]
Full MCA mode

\[ x \sim y \]

\[ mca(x) = \text{round}(\text{inexact}(\text{inexact}(x) - \text{inexact}(y))) \]

\[
\begin{array}{c}
\hline
x \\
\hline
y \\
\hline
x - y \\
\hline
\end{array}
\]

\[
\begin{array}{c}
? \\
\hline
x \\
\hline
y \\
\hline
x - y \\
\hline
\end{array}
\]

Outbound

Inbound

Different bits

Cancellation detection
**Bitmask backend**

**VERIFICARLO\_PRECISION=11**

IEEE754 Single precision 32-bit

**Mode ZERO**

3.1415927

**Mode INV**

3.1415927

**Mode RAND**

3.1415927

**Mode OR**

3.1416013

**Mode XOR**

3.1407475

**AND**

Mask

3.140625
**Errors propagation**

```fortran
subroutine pawpsp_calc(...)  
    ...  
    if (testval) then  
        ...  
        call simp_gen(qq, nhat, vale_mesh)  
        qq=zion/four_pi-qq  
    end if  
    call atompaw_shpfun(...)  
    nhat(1:msz)=qq*nhat(1:msz)  
    tnvale(1:msz)=tnvale(1:msz)+nhat(1:msz)  
    call pawpsp_cg(...)  

subroutine pawpsp_cg(...)  
    ...  
    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 if  
    end do  
    ...  
    call simp_gen(r1torm, ff, radmesh)  
```

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

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
```
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
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 $\Rightarrow$ 88 functions
Reduces the search space by $\times 30$
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].

*Introduction to the Special Issue ”Real Numbers and Computers”.* 
Springer, 1996.

*A dynamic program analysis to find floating-point accuracy problems.* 


*Automatically improving accuracy for floating point expressions.* 

*Monte Carlo Arithmetic: exploiting randomness in floating-point arithmetic.* 
University of California. Computer Science Department, 1997.

*Precimonious: Tuning assistant for floating-point precision.* 

*Finding root causes of floating point error with herbgrind.* 