Verificarlo applied to ABINIT: detecting numerical instabilities

Yohan Chatelain, Pablo De Oliveira, Eric Petit, Jordan Bieder and Marc Torrent
Introduction

Objective:
- Numerical debugger and analyzer of the floating-point model

Context:
- Complex HPC environment: heterogeneous parallel architecture, compiler optimization, parallelization paradigm
- ABINIT: large program with millions line of code

Proposal:
- Automatically pinpoint the impact of the floating-point model on the numerical stability of regions of code
Outline

- Verificarlo
  - How does it work?
  - Estimating output error
  - An example: Chebyshev polynomial

- Application on ABINIT
  - The Perovskite test case
  - Classify functions by numerical sensitivity
  - Fine-grained analysis
  - Numerical improvement

- Conclusion & future prospects
Verificarlo applied to ABINIT: detecting numerical instabilities
Verificarlo

- Open Source Project under GPL licence, developed by University of Versailles and ENS Paris-Saclay
- Automatically analyses the numerical stability of applications
- Introduces a stochastic error on each floating-point operation
Verificarlo: reminder about floating-point

IEEE-754 Standard for Floating-Point Arithmetic

\[ f = (-1)^s \cdot b^e \cdot x_0 x_1 \ldots x_{m-1} \]

\[ 0 \leq x_i \leq b - 1 \]

<table>
<thead>
<tr>
<th>Format</th>
<th>sign</th>
<th>exponent</th>
<th>mantissa</th>
<th>Total bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>Single</td>
<td>1</td>
<td>8</td>
<td>23</td>
<td>32</td>
</tr>
<tr>
<td>Double</td>
<td>1</td>
<td>11</td>
<td>52</td>
<td>64</td>
</tr>
<tr>
<td>Quad</td>
<td>1</td>
<td>15</td>
<td>112</td>
<td>128</td>
</tr>
</tbody>
</table>
Monte Carlo Arithmetic [parker1997monte]

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

\[ t = 21 \]

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

\[ RR(x \circ y) = \text{round}(\text{inexact}(x \circ y)) \]
Verificarlo: Estimating output error

Rounding errors distribution:
- Estimated by using \( N \) Monte Carlo samples

Significant digits number:

\[
\tilde{s}(\chi) = - \log_{10} \left( \frac{\tilde{\sigma}}{\tilde{\mu}} \right) \xrightarrow{N \to \infty} s = - \log_{10} \left( \frac{\sigma}{\mu} \right)
\]

\( \tilde{\mu} \): empirical expected value
\( \tilde{\sigma} \): empirical standard deviation

Estimate the number of correct significant digits
Verificarlo: An example

**C/C++/Fortran Application**

Replace FP operation by MCA equivalents

MCA backends:
- mpfr (libmca)
- quad
- bitmask

**Executable**

Sampling (embarrassingly parallel)

**Chebyshev polynomial:**
- $T_{20}(x), x \in [0, 1]$
- 100 points across [0, 1]
- 500 samples for each point
- Instability around 1
- $T_{20}(x) = \cos(20\cos^{-1}(x))$
Application on ABINIT
Scientific computing code developed by an international community of industrial and academic scientists
Allows to find the total energy of a quantum system within Density Functional Theory (DFT)
Code large and complex (~ 1,000,000 lines of Fortran)
Suffering from numerical instabilities when vectorized
ABINIT: The hydrogen test case

- The test case:
  - Find optimal inter-atomic distance for two hydrogen atoms
  - Simple example without non-local effects
  - Proof of concept to evaluate the cost of the method
  - Measure mean and standard deviation of MCA errors
  - Global analysis
ABINIT Virtual precision exploration

Verificarlo applied to ABINIT: detecting numerical instabilities
ABINIT Virtual precision exploration

Verificarlo applied to ABINIT: detecting numerical instabilities
Verificarlo applied to ABINIT: detecting numerical instabilities
ABINIT: A virtual precision exploration

Verificarlo applied to ABINIT: detecting numerical instabilities
ABINIT: Perovskite, the realistic use case

Perovskite ($\text{BaTiO}_3$)
- Physic more complex
- Suffering from numerical instabilities when vectorized
- Not converging with Random Rounding mode

Problem: How to pinpoint the numerical unstabilities?
- Verificarlo is time consuming
- Exhaustive analyse of the coupling of functions is impracticable: $2^\#Funct\text{ions} \times t\text{ precisions} \times N\text{ samples}$

Idea:
- Reduce the set of function to test: some function does not impact the final result
- Modify the introduction of error: MCA is costly, low-order model (BITMASK backend)
Numerical sensitive functions

Functions that impact the final result:
- For each function, plot the minimal precision required to reach machine accuracy
- Only 88 functions among 3400 functions

Functions < 23:
- 1/3 of the functions are below 23bits
- Possible transformation:
  double precision $\rightarrow$ single precision
  memory scaled down, computation faster

Functions $\geq$ 23:
- Sensitive functions
- Require a fine-grained analysis

! No coupling effects
Numerical sensitive functions

Functions that impact the final result:
- For each function, plot the minimal precision required to reach machine accuracy
- Only 88 functions among 3400 functions

Functions < 23:
- 1/3 of the functions are below 23 bits
- Possible transformation: double precision → single precision
  - memory scaled down, computation faster

Functions ≥ 23:
- Sensitive functions
- Require a fine-grained analysis

! No coupling effects

Verificarlo applied to ABINIT: detecting numerical instabilities
Simpson’s integral:
- Compute the integral over a generalized grid using Simpson’s rule
- One of the most numerical sensitive function

Fine-grained analysis:
- Trace the numerical quality over time
- Detect parts of code causing the numerical unstabilites
- Find the errors propagation across variables
**VerificarloTracer**

- Allows one to trace the numerical quality of a variable over time
- Automatically instrument store instruction to output values
- Plot information gracefully

**Example** [bajard1996introduction]:

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

- Numerical limit \(\neq\) Theoretical limit
- Need to track the evolution of the numerical quality
ABINIT: Numerical quality of `simp_gen`

- Evolution of the number significant digits.
- 24 samples, virtual precision $t = 53$ in Random Rounding mode (RR)
- Running on Occigen GENCI cluster
ABINIT: \textit{simp\_gen} function

\texttt{subroutine simp\_gen(intg, func, radmesh)}

\ldots

\texttt{nn=radmesh\%int\_meshsz}

\texttt{simp=zero}

\texttt{do i=1,nn}

\hspace{1cm} \texttt{simp=simp+func(i)*radmesh\%simfact(i)}

\texttt{end do}

\ldots

\texttt{intg=simp+resid}

\texttt{end subroutine}

\textit{simp\_gen} function, can be seen as a dot product
subroutine simp_gen(intg, func, radmesh)
...

nn=radmesh%int_meshsz
simp=zero

! Dot product in twice the working precision from
! Ogita, Rump and Oisha [ogita2005accurate]
! Using library libeft github.com/ffevotte/libeft

call Dot2(simp, func, radmesh%simfact)
...

intg=simp+resid
end subroutine

simp_gen function, can be seen as a dot product
ABINIT: A first approach

- Evolution of the number significant digits.
- Compensated version of simp_gen using Dot2 [ogita2005accurate].
subroutine pawpssp_calc(...)  
...
if (testval) then
  nhat(1:msz)=tnvale(1:msz)*vale_mesh%rad(1:msz)**2
  call simp_gen(qq,nhat,vale_mesh)
  qq=zion/four_pi-qq
end if
call atompaw_shpfun(0,vale_mesh,intg,pawtab,nhat)
nhat(1:msz)=qq*nhat(1:msz)
tnvale(1:msz)=tnvale(1:msz)+nhat(1:msz)
...
call pawpssp_cg(..., tnvale, ...)
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)
ABINIT: Compensated version

- Evolution of the number significant digits.
- Compensated version + compensated parts outside simp_gen
Conclusion & Future Prospects
Future prospects

- Reproduce with stochastic errors “realife” bugs such as when vectorizing

- Explore benefits of optimizations such as mixed-precision and approximate computing
Conclusion

Verificarlo: a tool for automatically detecting numerical instability
- Visualises the numerical quality of a variable over time
- Pinpoints functions causing global errors
- Reveals possible optimizations such as precision reduction
Chebyshev

\[ T_{20}(x) = \cos(20 \cos^{-1}(x)) \]
\[ = 524288x^{20} - 2621440x^{18} \]
\[ + 5570560x^{16} - 6553600x^{14} \]
\[ + 4659200x^{12} - 2050048x^{10} \]
\[ + 549120x^{8} - 84480x^{6} \]
\[ + 6600x^{4} - 200x^{2} + 1 \]
Backend BITMASK

VERIFICARLO\_PRECISION=11

IEEE754 Single precision 32-bit

Mode ZERO

AND

Mode INV

OR

Mode RAND

exponent
mantissa

0 1

VERIFICARLO applied to ABINIT: detecting numerical instabilities
Inbound / Outbound

\[ x \sim y \]

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

---

Cancellation detection
Verificarlo overhead

<table>
<thead>
<tr>
<th>version</th>
<th>samples</th>
<th>total time (s)</th>
<th>time sample (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>original program</td>
<td>1</td>
<td>0.056</td>
<td>0.056</td>
</tr>
<tr>
<td>Verificarlo MASK</td>
<td>128</td>
<td>12.45</td>
<td>0.097</td>
</tr>
<tr>
<td>Verificarlo MPFR</td>
<td>128</td>
<td>834.57</td>
<td>6.52</td>
</tr>
<tr>
<td>Verificarlo QUAD</td>
<td>128</td>
<td>198.58</td>
<td>1.55</td>
</tr>
<tr>
<td>Verificarlo MPFR 16 thds.</td>
<td>128</td>
<td>54.39</td>
<td>0.42</td>
</tr>
<tr>
<td>Verificarlo QUAD 16 thds.</td>
<td>128</td>
<td>12.54</td>
<td>0.098</td>
</tr>
</tbody>
</table>

Figure 1: Verificarlo overhead on a compensated sum algorithm (double) on a 16-core 2-socket Xeon E5@2.70GHz.

- Monte Carlo Arithmetic requires additional precision which is costly
- No size fits all
  - MASK backend is cheap (x2 per iteration) but imprecise
  - QUAD backend implements exact MCA model but costly (x27 per iteration)
  - MPFR used only for validation
- Embarrassingly parallel across executions