Floating-point profiling of ACTS using Verrou

Hadrien Grasland    David Chamont    François Févotte    Bruno Lathuilière

CNRS – LAL            EDF R&D – PERICLES

This project has received funding from the European Union’s Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.
Verrou: a floating-point error checker

- Run any program in Valgrind
- Verrou alters the rounding of its floating-point operations
  - Small effect on a stable numerical computation
  - Large impact if unstable (→ caught by test suite)
- Also points out presence of NaNs (→ often symptom of silent failure)
- Underlying theory: asynchronous CESTAC method, Monte Carlo Arithmetic
Choices of rounding mode

- **Stochastic modes:**
  - *Random:* 50/50 choice between upward/downward
  - *Average:* upward/downward probability determined from exact result
  - Few false positives (no change on average), but non-deterministic
  - Best for initial exploration, can force an RNG seed to reproduce a run

- **Deterministic modes:**
  - Upwards, downwards, towards 0, farthest
  - Can be convenient for failure analysis, especially delta-debugging
Delta-debugging

• Locates the origin of a verrou-induced test failure
  – Combines an include/exclude mechanism with binary search
  – Can go down to the granularity of individual lines of code
  – Requires debug information (“-g” compiler flag, “-debuginfo” packages...)

• Very powerful, but takes a while to master
  – Prefer deterministic rounding modes if they reproduce your instability
  – Otherwise, must tune number of executions before declaring success
  – If your test uses random input, force a specific seed that reproduces failure
  – Even with binary search, can take a while to converge
The joy of verrouddf
ACTS (A Common Tracking Software)

- **Project goals:**
  - Major clean-up of ATLAS Run 2 tracking
  - Usable by other experiments, R&D projects
  - Dedicated plenary by A. Salzburger at 6pm

- **My main areas of interest:**
  - Performance (algorithms, trigonometry, vectorization, memory accesses...)
  - Quality (thread-safety, maintainability, numerical accuracy...)

Stress-testing ACTS using Verrou

- **Build recommendations:**
  - CMAKE_BUILD_TYPE=Debug
  - ACTS_BUILD_TESTS=ON
  - ACTS_BUILD_INTEGRATION_TESTS=ON
  - As many plug-ins as your patience allows!

- **Usage on unit tests:**
  - `valgrind --tool=verrou \`
    --rounding-mode=random \`
    --trace-children=yes* ctest -j8

* By default, Valgrind does not attach to the extra processes spawned by ctest
Issues in the original code

- **In the tests:**
  - Fragile float comparisons (exact, relative near 0, uncontrolled text dump)
  - Using floating-point pow() to compute powers of 2
  - Some tests gratuitously injected NaNs in input, obscuring actual FP errors :-/
  - One test is extremely sensitive to the rounding of \((2\pi/N)\) → Not elucidated yet

- **In ACTS itself:**
  - Divisions whose denominators can get arbitrarily close to zero
  - Compute \(\varphi\) coordinate difference via two atan2 + subtract + wraparound

- **False positives:**
  - libm's sin/cos/tan algorithms are rounding-sensitive: leave them alone
Step 2: Move to single precision

• The challenge:
  – HEP code tends to use double precision as a safe default
  – Single-precision compute is at least 2x as fast*, more on some hardware
  – Single-precision isn’t always enough (gives $\sim 10^{-6}$ precision, but $m_p >> 10^6 m_e$)
  – Choice of precision is undocumented, cannot know if double is used on purpose

• The plan:
  – Move all current hard-coded doubles to single-precision, see what breaks
  – Tune tolerance up a bit & use delta-debugging to locate where things break
  – Selectively bring back double precision (or compensated algorithms) as needed

* Uses 2x less cache space & memory bandwidth, enables 2x wider vectorization
Things found so far

- **More test suite woes**
  - Even more exact float equality / uncontrolled text dump comparisons
  - Some very low relative tolerances ($10^{-11}$) → Arbitrary or intentional?
  - Edge effects (e.g. $\text{min} \leq \text{value} < \text{max}$) → Probably a false positive *in this case*
  - Some tests help more than others (detailed comparisons >> success flag)

- **But also...**
  - Incorrect call to Eigen::Transform constructor which only worked by luck (!)
  - ACTS inverts a matrix on *every* global → surface-local coordinate conversion
  - Footguns in boost::test’s handling of tolerances (percentages, float != double...)
Conclusions

- **Verrou is a very nice validation tool for numerical code**
  - Very easy to get started, catches many classic floating-point issues
  - Also helps finding some suspicious (e.g. unnecessarily complex) code
  - No magic bullet: Depends *heavily* on the quality of your test suite

- **Could use some improvements, under discussion**
  - Better default configuration (e.g. automatically exclude classic false positives)
  - double → float porting use case could use dedicated support in verrou
  - verrou_dd is slow and serial, needs more parallelization love
  - Narrowing down rare failures with verrou_dd can be difficult
  - verrou_dd could use a backtrace-like report (e.g. for failures deep inside Eigen)
Questions? Comments?

https://github.com/edf-hpc/verrou
IEEE-754 floating-point is hard

- Internally uses base 2 → Most decimals numbers are not stored exactly
- Not associative → \[ (1 + 10^{30}) - 10^{30} \neq [1 + (10^{30} - 10^{30})] \]
- Not totally ordered → Think before you sort a list of floats...
- Javascript-style error handling → Trivial mistakes easily get ignored
- List accumulation can saturate → Addition is dangerous
- Catastrophic cancellation → Subtraction is dangerous
- Limited exponent range → Multiplication and division are dangerous
- Full of correctness edge cases → +/-0, multiple NaNs, denormals, +/-inf...
- Full of performance pitfalls → Trigonometry, sqrt, div, NaNs, subnormals...