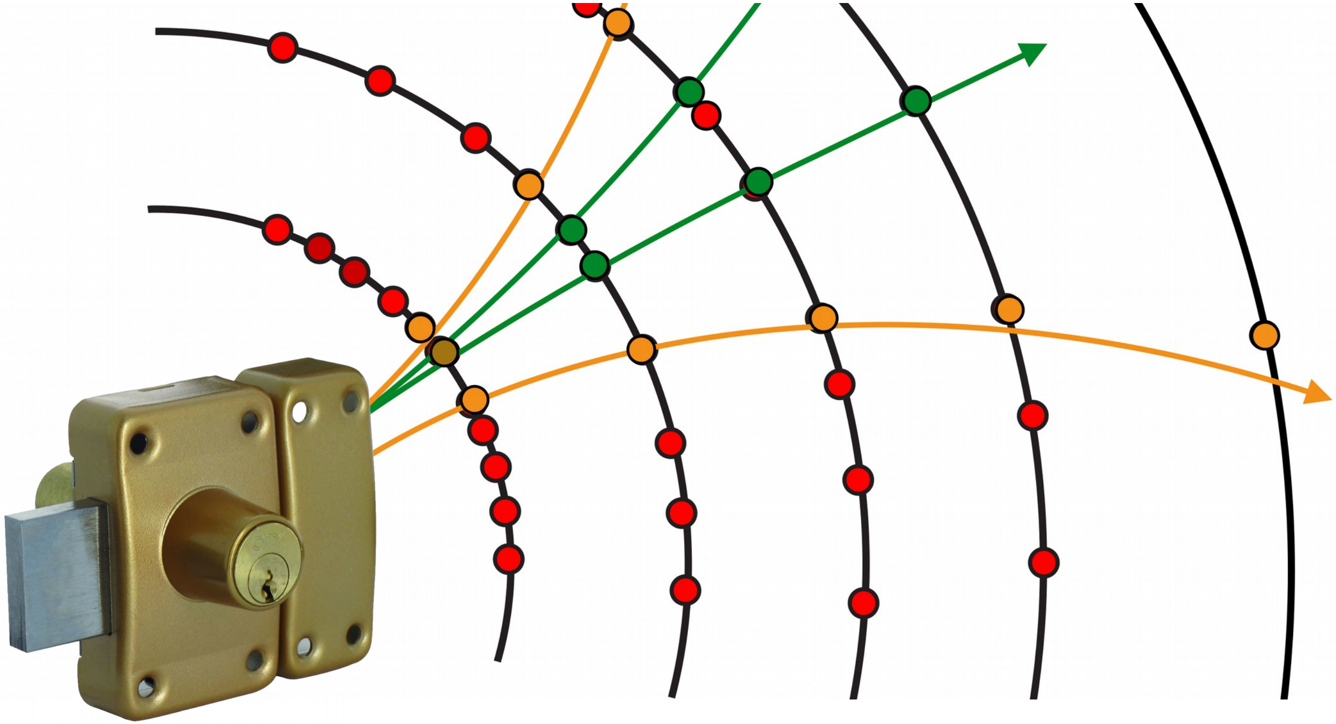


Floating-point profiling of ACTS using Verrou

Hadrien Grasland David Chamont François Févotte Bruno Lathuilière

CNRS – LAL EDF R&D – PERICLES

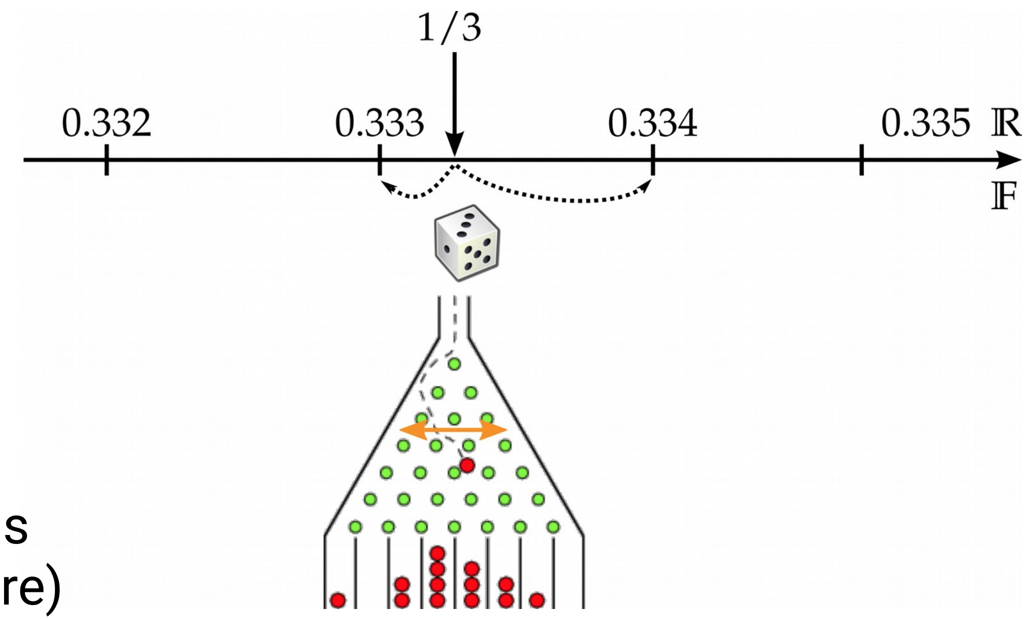
AIDA 2020

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.

LABORATOIRE DE L'ACCÉLÉRATEUR LINÉAIRE

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

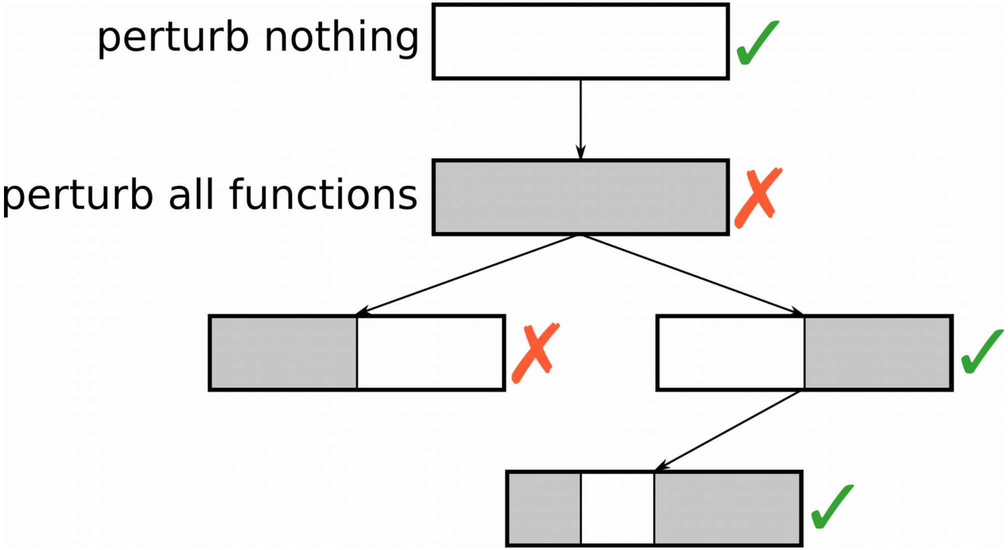
Choices of rounding mode

- **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 verrou_dd



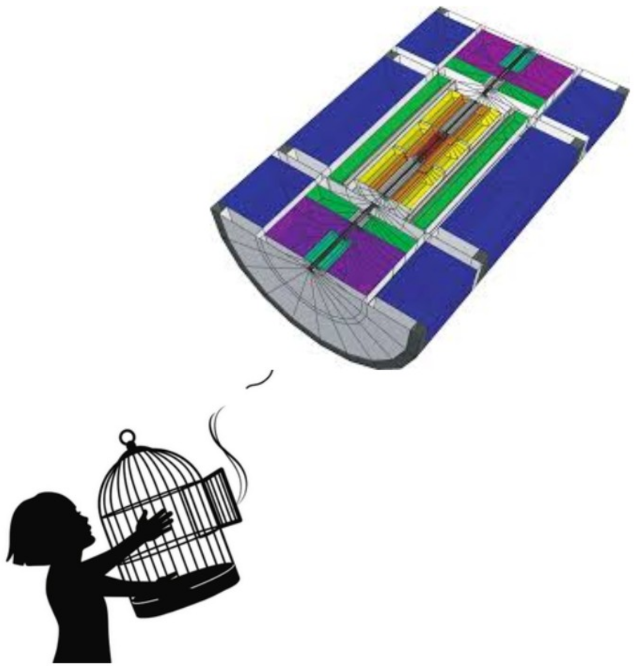

```
  SurfaceArrayCreatorTests.cpp:127 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:135 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:138 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:139 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:140 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:141 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:143 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:144 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:145 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:146 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
/root/acts-core/build/dd.line/d75047bd521d366772ed3d4c7c568d2e  --( run )->  FAIL(0)

dd (run #2): trying 5 + 5
/root/acts-core/build/dd.line/5e10220d10c2190cfcb826fd411ee7bd  --( run )->  PASS

dd: 5 deltas left:
  SurfaceArrayCreatorTests.cpp:127 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:135 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:138 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:139 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:140 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
/root/acts-core/build/dd.line/7a5322bddf57197c54eea333810abae5  --( run )->  FAIL(0)

dd (run #3): trying 2 + 3
/root/acts-core/build/dd.line/4422ef95b34880f35036cb8e3b472dfb  --( run )->  FAIL(0)
/root/acts-core/build/dd.line/847c43ef521eba2787dce6641522b594  --( run )->  PASS

dd: 3 deltas left:
  SurfaceArrayCreatorTests.cpp:138 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:139 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
  SurfaceArrayCreatorTests.cpp:140 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
/root/acts-core/build/dd.line/f93d07fc8297ecb70f789e3b6d61786d  --( run )->  FAIL(0)

dd (run #4): trying 1 + 2
/root/acts-core/build/dd.line/db7db6ec4fc361c18e8d1561f65c164e  --( run )->  PASS

dd: 1 deltas left:
  SurfaceArrayCreatorTests.cpp:138 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
/root/acts-core/build/dd.line/9f73270ae01c6d4b57fa6fab20904546  --( run )->  FAIL(0)
dd: done
ddmax (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd):
  SurfaceArrayCreatorTests.cpp:138 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
/root/acts-core/build/dd.line/9f73270ae01c6d4b57fa6fab20904546  --(cache)->  FAIL
```

```
ddmax (global):
  SurfaceArrayCreatorTests.cpp:221 (_ZN4Acts4Test26SurfaceArrayCreatorFixture17makeBarrelStaggerEiidddd)
  SurfaceArrayCreatorTests.cpp:103 (_ZN4Acts4Test26SurfaceArrayCreatorFixture21fullPhiTestSurfacesECEmdddddd)
  SurfaceArrayCreatorTests.cpp:138 (_ZN4Acts4Test26SurfaceArrayCreatorFixture22fullPhiTestSurfacesBRLEmddddd)
0d0cdb4d9c9d:~/acts-core/build #
```

ACTS (A Common Tracking Software)

- **Project goals:**
  - Major clean-up of ATLAS Run 2 tracking
  - Usable by other experiments, R&D projects
  - See presentations by A. Salzburger*

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



* For example https://indico.cern.ch/event/587955/contributions/3012710/

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

```
The following tests FAILED:
          3 - ParameterSetUnitTest (Failed)
          5 - CurvilinearParametersUnitTests (Failed)
          6 - BoundParametersUnitTests (Failed)
         15 - ProtoLayerUnitTest (Failed)
         24 - PropagatorUnitTests (Failed)
         26 - SeedingUnitTest (Failed)
         27 - SeedingToolsUnitTest (Failed)
         33 - CylinderSurfaceUnitTest (Failed)
         34 - ConeSurfaceUnitTest (Failed)
         35 - DiscSurfaceUnitTest (Failed)
         41 - ConeBoundsUnitTest (Failed)
         48 - DiscTrapezoidalBoundsUnitTest (Failed)
         49 - SurfaceArrayUnitTest (Failed)
         51 - GeometryIDUnitTest (Failed)
         52 - BinningDataUnitTest (Failed)
         59 - InterpolationUnitTest (Failed)
         61 - CylinderVolumeBoundsUnitTest (Failed)
         63 - SurfaceArrayCreatorUnitTest (Failed)
         64 - LayerCreatorUnitTest (Failed)
```

* 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 rounding of ($2\pi/N$) → Not elucidated yet
- **In ACTS itself:**
  - Divisions whose denominators can get arbitrarily close to zero
  - Compute φ 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 ~10^{-6} precision, but $m_P >> 10^6 m_e$…)
  - Choice of precision is undocumented, can't tell if double used on purpose
- **Initial 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

First round of findings

- **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. min <= value < 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…)

Limits of initial approach

- **Single precision dev branch was unmaintainable**
  - Changing every “double” to “float” = merge conflicts with everything
- **Solved by “float” rounding mode in verrou 2.0**
  - Greatly reduced magnitude of single-precision patch
  - Almost as good as real port (but doesn’t like std::numeric_limits & such)
- **Led to more findings**
  - Uninitialized memory used in average with 0 weight (0 x NaN != 0)
  - Broken covariance matrix comparison logic (single relative tolerance)
  - Waiting for $u_N \leq \lim_{N \to \infty} u_N$ where $u_0 > u_N$

Conclusions

- **Verrou is a nice validation tool for numerical code**
  - Easy to get started, catches many classic floating-point issues
  - Helps finding some suspicious (e.g. unnecessarily complex) code
  - No magic bullet: Depends *heavily* on the quality of your test suite
- **Using it was beneficial to ACTS code quality**
  - Comparison and tolerances in test were deeply re-thought
  - Uncovered several classic numerical gotchas in core codebase
- **Single-precision port sadly remained a prototype**
  - Did not find answer to “How much precision do you really need ?”

Perspectives

- **Found areas of future Verrou improvement**
  - Better default configuration (e.g. automatically exclude libm false positives)
  - verrou_dd is slow and serial, needs parallelization + algorithm work
  - Narrowing down rare failures with verrou_dd can be difficult
  - verrou_dd could use backtrace sensitivity (for “dot product failures”)
- **Verrou already improved much during this study**
  - Support for longer symbol names (~mandatory for modern C++)
  - Python 3 compatibility in verrou_dd
  - verrou_dd restricted to symbols with FP ops
  - “Float” rounding mode, backtrace on NaN

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…
- **Not optimized by compilers** → Byproduct of previous properties