Framework & Tracking @ IJCLab

Hadrien Grasland

IJCLab – Orsay

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

Advanced Tracking Tools

Belle 2 profiling

- Belle 2 tracking uses genfit with a Geant4 geometry
  - Slow + little maintenance → Desire to move to Acts
  - Few people available → Desire for a **piecewise integration**
  - But what would be a promising candidate?
- I profiled the Belle 2 reco to answer this question
  - 40% of reco time spent in genfit + children
  - Large chain of **non-inlined** hot function calls…
  - ...leading to G4Navigator calls (33% of genfit time)
- Next: Improve hot path inlining + try Acts geometry.

Acts micro-benchmarks

- Acts benchmark infrastructure underwent major rework
  - Multiple timed runs → Output timings now have error bars
  - Robust stats → Filter out OS scheduler timing noise
  - Warmup time → Study steady state, not transients+steady
  - Optimization barriers → No “unfair” code simplification
  - Lambda-friendly API → Very easy to write new benchmarks
- **Usage example:**

```
const auto fixedPos = genPos();
const auto map_cached_result = Acts::Test::microBenchmark(
  [&] { return bFieldMap.getField(fixedPos); },
  iters_map
);
std::cout << map_cached_result << std::endl;
```

- **Output:** 20000 runs of 500 iteration(s), 591.7ms total,
  29.5020+/-0.0363µs per run, 59.004+/-1.624ns per iteration

Boundary checks

- Track-surface intersection is a very common operation
  - Search for candidates in following, material integration
  - Navigate through detector volumes…
- Different kinds of 2D boundary checks may be used
  - No boundary check at all, infinite surface (1 CPU cycle)
  - Inside/outside boundaries (6~10ns on a plane)
  - χ^2-based tolerance (10~90ns on a plane)
- I have been optimizing this part of Acts by
  - Eliminating unnecessary work in “simpler” checks
  - Speeding up χ^2-based checks (ongoing)

Build performance

- Acts build resource consumption is concerning
  - Some compilation units take several minutes to build
  - 3-4GB/process is common, once saw a 10 GB process
  - Often need to tune down concurrency to avoid OOM
- I started investigating this issue
  - Can recommend Clang 9's -ftime-trace for such work
  - Most central issue seems to be Eigen expression templates
    - Possible workaround: wrapper that forces eager evaluation
    - Another argument in favor of moving away from Eigen in the future…
  - Boost libs (program options, MPL…) may be an issue too

Smaller things

- Helping new students working on Acts
  - Xiaocong Ai in her work on a Kalman filter GPU port
  - Georgiana Mania in her work on internal TBB parallelism
- Maintaining Acts spack package
  - New release cadence: 26 releases since last AIDA meeting!
  - …but most don’t touch the build system, thankfully
- Working on the next big Acts-related projects

Future prospects

- Continue optimizing surface intersections
  - Finish optimizing boundary checks
  - Review usage of boundary checks in Acts
  - Optimize surface intersection proper (not dominant yet)
- Optimize Runge-Kutta covariance transport
- Resolve Acts build resource consumption issues

Framework Extensions

Context

- IJCLab's FW contribution is now focused on **MarlinMT***
  - More AIDA-2020 WP3 synergies than with Gaudi
  - Also a more productive work environment
- Area of interest: **Parallel histogramming**
  - Two classic approaches: mutex and per-thread copies
  - Must work around ROOT 6's thread-hostile design
  - Can we do better? Yes… with help from ROOT 7

* See also Rémi's presentation.

ROOT::Experimental::RHist

- Complete redesign/rewrite of THx histograms
  - Templated over bin type, dimension, recorded stats
  - Sane API scope, no crazy global variables
  - Native support for buffered multi-threaded fills
- **Problem:** In a *very* preliminary state right now
  - It's not that its API *may* change, it *will need to*
  - Fill is solid, Add is okay, queries need major work
  - Not something you want to expose to physicists
- **Solution:** Use RHist internally, convert to THx in output

RHist converter

- A tool that…
  - Ingests any RHist that has a THx equivalent
  - Produces the equivalent ROOT 6 histogram
  - Output almost* indistinguishable from filling via ROOT 6
- Current status
  - 98% feature-complete** and used by MarlinMT
  - Open-source, but not advertised yet because…
    - …relies on RHist internals → should live in ROOT

* Modulo some edge cases around overflow bin contributions to statistics.

** Does not yet support some obscure TH3 axis configurations + no compilation failure tests yet.

ROOT developments

- RHist converter development & testing revealed…
  - Many bugs and API shortcomings in RHist
  - Some of which call for major refactorings
- …so I started working with the RHist team
  - Resolved MarlinMT integration issues as they came up
  - Fixed many bugs, extended the test suite
  - Redesigned RAxis for type-erased access ergonomics
  - Now contributing to regular/overflow bin separation
  - Next, will contribute to API redesign to kill pImpl
  - …and after that the RHist → THx converter should go in

So, why RHist?

- Frequently asked question: *What about Boost.Histogram*?
  - Clearly playing in the same field as RHist
  - Both libraries need serious work before becoming usable
    - RHist: Going in the right direction, but very immature.
    - Boost.Histogram: No ROOT integration, no threading support.
  - Significant work already done on RHist
  - Politically, ROOT seems like a safer horse to bet on
  - In the worst case, can change MarlinMT hist backend

Future prospects

- Finish solving various RHist problems
- Integrate RHist converter into ROOT

Questions? Comments?