Recent performance improvements in ALICE simulation/digitization

Sandro Wenzel / CERN-ALICE

WLCG meeting, San Francisco, 9.10.2016

Outline

- Motivation
- Initial profile situation
- Developments to increase software performance
- Performance increase
- Outlook

Motivation

- Improve CPU performance of complete algorithmic pipeline in ALICE
  - in the current AliRoot framework
  - in ALICE specific code (independent of improvements in external simulation packages)
  - as preparation for ALICE-O2 and higher luminosity requirements

- Start with analysis of simulation/digitization as the most important CPU consumer

Initial profile

- Status March 2016: Use valgrind/igprof on typical MC simulation scenarios using Geant3 or Geant4
  - p-p benchmark
  - Pb-Pb benchmark
- Main CPU users:
  - TPC digitization   (expected)
  - simulation physics routines (expected)
  - TGeo geometry routines (expected)

  - dynamic_casts on ~6% level (unexpected)
  - accessing thread local storage variables on ~2% level (unexpected)
  - accessing ROOT containers on ~10% level (unexpected)
  - lots of calls to very small functions on the 1% level (unexpected)

Performance optimization campaign

- A dedicated campaign was started to remove the “unexpected” hotspots: “grab” the low-hanging fruits !!
- ROOT related issues:
  - often not offering fast container access
  - slow/non-optimal access to TLorentzVector
  - no type-strictness of ROOT containers
    - dynamic_cast oriented type checking …
    - generic sorting algorithm are based on virtual functions, …
- Other typical problems:
  - overuse of virtual function paradigm (when not strictly necessary)
  - cache access problems due to wrong loop order etc.
  - campaign to remove unnecessary virtual functions + inline campaign in AliROOT + Geant3 + VMC
    - help the compiler doing optimizations

Overall Results

- Improvements measured on running a Pb-Pb simulation (typical event) - Geant3
- **~20% gain** in total runtime achieved (from ~2359s to ~1966s)

	Original	**Tuned compiler flags**	**Code optimizations in AliRoot/ROOT**
RunSimulation	**1462s**	**1367s**	**1182s**
RunSDigitization	**683s**	**692s**	**585s**
Total simulation + all digitization + other parts	**2359s**	**2274s**	**1966s**

Details: Dynamic cast problem

- Overuse of dynamic casting in AliRoot:
  - ROOT does not offer strongly typed containers, preventing compile-time checks on objects put into them
  - users are then inclined to perform type checks at runtime
- Example:
  - typically we write
    - TObjArray *fDetectorModules // supposed to store objects of type DetectorModule and derived
  - retrieving objects … one is inclined to say
    - if (dynamic_cast<DetectorModule>(fDetectorModules->At(i))) …
  - This is an expensive operation at every read from fDetectorModule which happens every step
    - although fDetectorModule never changes
  - sums up to almost ~6ish %
- Action taken:
  - perform type checks only at initial write to containers
  - use static_casts to read from (const) containers
  - achieved complete elimination of dynamic_cast problem from profile

Details on some ROOT issues

- Non-optimal ROOT Container access
  - AliROOT relies heavily on ROOT containers
  - example 1: no fast way to get element from TArrayI fast; both TArrayI::At( ) and TArrayI::operator [](int) both perform bounding checks
  - example II: retrieving element TMatrixT<>::operator(row, col) always performs assert checks + boundary checks
  - ticket ROOT-5472 opened a while ago
  - problems have been fixed in AliROOT with custom "fast-access" functions
- TLorentzVector
  - heavily used in parts of simulation/digitisation
  - access operators/constructors non-inline and convoluted (leading to 1% to 2% overall cost)
  - an "optimizing" patch has been submitted to ROOT and was accepted
- ROOT Sorting
  - sorting a TClonesArray is slower than sorting a "std::vector<T>" because the type of element is not known —> preventing inlining of "sort/compare" functor
  - A template version of TClonesArray::Sort<T> has been implemented, speeding up sorting by factor ~2
  - plan to submit patch to ROOT team

Avoid access to thread-local variables

- Another major problem turned out to be accessing “thread-local storage” variables
  - noticeable by excessive appearance of _tls_get_addr_ in profiler outputs
  - strange since AliRoot not using threading for moment
  - major reason was found to be accessing the singleton Virtual Monte Carlo object with TVMC::GetMC()
  - problem of this function was: “virtual” + “non-inline” + “thread local local storage object”
- Solution taken:
  - cache a reference to MC object in AliROOT simulation base class … not longer need to call GetMC()
  - thread local storage problem almost completely resolved apart from some things remaining deep inside ROOT/TGeo

Miscellaneous points

- Systematic study of build system etc.
- Optimized the build flags for Geant3
  - previously compiled in a “conservative” mode
  - reason: true RELEASE mode compilation for Geant3 leads to numeric instabilities
  - did a systematic scan of compiler flags and identified the cause of numeric instabilities
  - can now build Geant3 almost in RELEASE mode: -O2 + -fno-strict-overflow

Outlook

- After this pass of optimization steps, we get a clearer view on the real important algorithms in AliRoot
    - geometry in simulation
    - TPC in digitization
- Next steps will be a tackling of those parts
- Concrete ideas exist to decrease the time spent in geometry routines via usage of "VecGeom"
    - modern, high-performance geometry package for simulation
    - e.g., plan to using the VecGeom engine in Geant4/Geant3 via the Virtual Monte Carlo interface
- Also take a look at reconstruction algorithms