Preparing the track reconstruction in ATLAS for a high multiplicity future

Robert Langenberg[1,2], Anthony Morley[3], Andreas Salzburger[2], Markus Elsing[2], Niels van Eldik[2] and Ruslan Mashinistov[4] on behalf of the ATLAS Collaboration

[1] Technische Universität München, Institut für Informatik, 85748 Garching, Germany
[2] CERN, CH-1211 Genève 23, Switzerland
[3] KTH Royal Institute of Technology, Department of Physics, SE-10691 Stockholm, Sweden
[4] Russian Academy of Sciences, Lebedev Institute of Physics, 53 Leninskij Prospekt, 119991, Moscow, Russia

E-mail: robert.langenberg@cern.ch

Abstract. The track reconstruction algorithms of the ATLAS experiment have demonstrated excellent performance in all of the data delivered so far by the LHC. The expected large increase in the number of interactions per bunch crossing introduces new challenges both in the computational aspects and physics performance of the algorithms. With the aim of taking advantage of modern CPU design and optimizing memory and CPU usage in the reconstruction algorithms a number of projects are being pursued. These include rationalization of the event data model, vectorization of the core components of the algorithms, and removing algorithm bottlenecks by using modern code analysis tools. Recent results of the advances made in these ongoing projects indicate up to three-fold speedup in the optimized modules while in some modules the code size could be reduced by up to 97% leading to higher readability, maintainability and decreased interface complexity.

1. Introduction

During run 1, the Large Hadron Collider (LHC) at CERN exceeded all expectations in performance delivering a total integrated luminosity of about 25 fb^{-1}. This would not have been possible without increasing the number of instantaneous collisions per bunch crossing to a peak value of 40, exceeding the design specifications of the LHC by a significant amount; see figure 1. The resulting time required for reconstruction that delivers the physics requirements approaches the limits given by the data rate. The current LHC long shutdown 1 (LS1) gives a unique chance to prepare the tracking for the next years of data taking. Current production software doesn't efficiently exploit computational resources, leading to computation and memory usage becoming bottlenecks. Data from run 1 is being processed on the WLCG grid consisting of machines all supporting some standard of vectorization and parallel computation on multiple cores. The track reconstruction software currently used in production has been written with non-parallel hardware architectures in mind, sometimes years ago with different approaches to software architecture, some of which nowadays are considered outdated. Current resource usage estimates suggest that a speedup factor of 2 in reconstruction time is needed to support future data taking and physics analyses adequately. The timeframe given for changes is set by LS1, after which the latest software will be frozen for production. To meet this goal, several parallel actions have been taken:

- Review, optimization and cleanup of the existing software,
- Simplification where possible and flattening inheritance hierarchies,
- Optimization of memory usage,
- Exploitation of vector registers and multithreading capabilities.

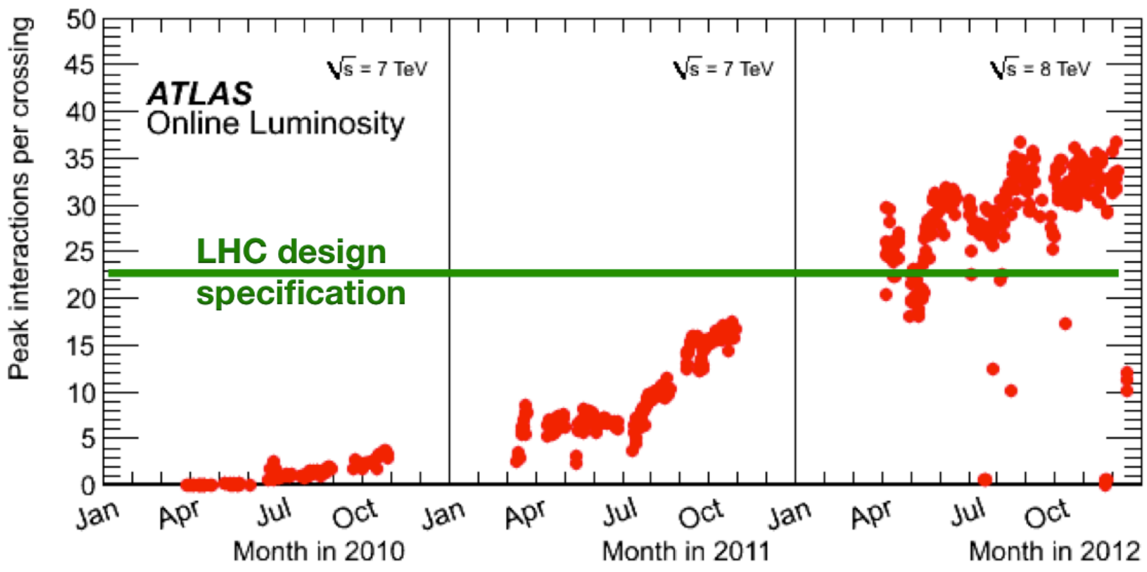

Figure 1. Luminosity during Run 1, with peak interactions clearly exceeding the design specifications indicated by the green line.

Track reconstruction in ATLAS is conducted using the ATHENA [1] framework. This software consists of algorithms, tools and services. These are connected with each other in the following way: events are processed by a series of algorithms. Each of these algorithms may call one or more tools. Algorithms and tools can access common services such as a common persistency service. Track reconstruction makes up a big part of the total computation budget and is performed in a series of many algorithms such that each algorithm at best uses up a few percent of the total time. Taking this into consideration, optimization cannot meaningfully be aiming at improving few selected algorithms but must affect many different packages. As the ATLAS reconstruction runs more than 600 algorithms and many more tools, measures taken must not require rewriting large parts of algorithms affected. Instead, they must be easy enough to rapidly change many different packages. Changes that involve touching hundreds of packages also require thorough analysis to assure goals are met after spending considerable time for implementation. Projects chosen for optimization were selected following a careful analysis of hot-spots and the potential gains and effort. Promising projects that exceed the time scale are conducted separately; for example, efforts to process multiple events in parallel require reworking the framework and making the algorithms thread-safe, which is unrealistic within the timescale given to achieve the goals set.

2. Methods and Measures Taken

In this section applied changes are being introduced and discussed.

2.1. *Preconditions*

While increased event throughput is the goal, tests have to be performed to assure physics performance is not negatively affected, possibly making undiscovered bugs hard to find if they show up only later in the call chain. Therefore, accompanying the development, either equivalence has to be proven or differences have to be shown not to negatively impact result accuracy, as could be the case for e.g. differences in numerical errors.

The projects discussed here can be divided into increasing code maintainability and reusability by simplification and increasing speed. Both do not always allow maintaining interfaces such that dependent packages have to be changed as well, causing a chain of required changes. Modifications have also been introduced such that future changes can be performed with minimum effort.

2.2. *Analysis Tools*

The following tools have been used to identify hot spots: perfmon [2], gperftools [3] and Gooda [4] to analyze CPU usage on algorithm-, function- and instruction level.

Perfmon is part of default ATHENA configuration and allows finding hot spots at the algorithm level, showing the amount of time and memory spent per algorithm. Perfmon results show a small number of algorithms to be responsible for a few percent of the whole runtime, exceeding other algorithms by far. Still, few percent do not justify time-consuming optimization as Amdahl's law limits the maximum gain. Nonetheless, results have been used to apply some less work-intensive optimizations. Gooda allows very detailed analysis on an instruction level. This can prove very helpful when trying to find out why parts of the code are responsible for a certain time spent. Gooda can show this as fine grained as per assembly operations assigned to lines of code.

Gperftools, originally developed by Google, is a sampler, using the stacktrace to allow analyzing call graphs and time spent per function, independent of from where the function is called. This showed a few libraries used in many different algorithms to be responsible for a considerable amount of time spent. The biggest consumers were the trigonometric functions from libm, allocation of memory with tcmalloc, and geometry functions from CLHEP.

Memory usage analysis will be conducted after application of optimizations.

2.3. *Applied and Ongoing Changes*

Several parts of the code have been, currently are or will in the near future be rewritten, restructured or optimized. These include the magnetic field service, replacing the currently used algebra library (CLHEP [5]) with a newer product (Eigen, [6]) that exploits modern CPUs, rewriting the TrackParameters class, moving away from libm, and replacing the outdated version of Google's tcmalloc [7] by a faster allocator.

2.3.1. Magnetic Field. The magnetic field service is frequently accessed in both simulation and reconstruction, as it is required to determine the curvature of a track. The field service used to be distributed in several files each containing thousands of lines of Fortran77 code with complex execution paths, making a thorough reworking difficult. The ATLAS experiment moved from FORTRAN to C++ as the main framework language already more than ten years ago, while a few components stayed using FORTRAN and being called with a wrapper function. The service was completely rewritten including several changes to the algorithms. Adding caching of the most recently used field values retrieved from the map resulted in potentially large speedups, strongly depending on its use. Event simulation gained most from the improvements not only because up to 18% of the simulation time was spent in the magnetic field routines, but also because of its usage frequently requiring field values from physically closely neighboured points. Other improvements include minimizing unit conversions between e.g. kilo Tesla and Gauss or between cm, m and mm, which took up measurable time. Some hot parts of the code were rewritten to autovectorize by code reordering and loop unrolling. All optimizations combined lead to a speedup of 20% in mock up tests conducted for reconstruction; investigation why deployed code doesn't see the expected speedups in full reconstruction is ongoing. Now only 2% of the simulation time is spent in the magnetic field routines, though this includes optimizations requiring less field access.

2.3.2. Replacing CLHEP by the Eigen Library. The CLHEP library is being used widely distributed through all parts of the software, providing geometry and linear algebra functions. CLHEP conveniently provides frequently required functionality but has several disadvantages, the most grave of which is it not being further maintained except for bug fixing. This led to more recent libraries outperforming CLHEP by factors of up to ten for matrix-matrix multiplication [8]. Other disadvantages of CLHEP are inconsistencies with Fortran- and C-style accessors to data structure elements and incompatibilities when mixing data types in a calculation. Comparisons of CLHEP, MKL [9], Eigen and SMatrix [10] showed Eigen to be faster or equally fast for matrix and vector operations on recent architectures. Eigen is a template library meaning it is only available in source code allowing inlining which greatly affects the speed of the provided functions since context switches are avoided. Another advantage of Eigen being a template library is the extensibility, further facilitated by provided macros [6]. As Eigen doesn't offer symmetric matrix operations, dedicated methods have been implemented allowing efficient calculation without reverting to different data types. Many convenience methods provided by CLHEP are not available in Eigen. By extending Eigen with these functions an interface very similar to CLHEP without losing performance by avoiding an additional call through inlining can be achieved. CLHEP types in ATHENA are accessed directly or using a multitude of wrappers, resulting in a confusing mix of different types. Trying to avoid this, a new namespace "Atlas Math and Geometry" (Amg) has been defined as a unified interface. This allows masking changes in the interface, for example, requiring changes only in a single point rather than distributed in the code. Another advantage is easy exchangeability with other algebra libraries, which usually exhibit a similar API. The migration from CLHEP to Eigen is currently advancing; the Inner Detector code is only days from running independently of CLHEP except for few persistified types, which are accessed from many not yet migrated places. To test the migrated parts of the code, a test bed has been set up allowing comparison of algorithms before and after the migration. As the API of Eigen has been adapted to match CLHEP as far as possible, the same test code can be used for both, ifdef statements exclude the library-specific parts from being compiled with an unsuitable release. Tests have shown the performance for the heavily used conversions from global coordinates in the detector to local coordinates on a surface and vice versa to be three times faster with Eigen, showing the potential gains.

2.3.3. Track Parameters. Tracks are extrapolated from surface to surface, where surfaces can be of different types. Current production software has track parameters class per surface type, resulting in dynamic casts to determine its type and unnecessarily complex persistification. All track parameters were unified, using only one templated type and adding an enum field for the type, allowing retrieval of the type without cast. Persistification is now necessary only for this one type, facilitating further changes also reflected by the number of code lines being reduced to only three percent of its original size.

2.3.4. GNU libm. The GNU libm provides many frequently used functions such as sqrt and trigonometric functions. As libm is outdated, libraries optimized for recent architectures outperform libm. Additionally, libm is IEEE compliant and therefore has to provide maximum accuracy, exceeding ATLAS' requirements but possbily wasteful with regard to resources. Performance was therefore compared to other libraries. Vdt [11], a library developed by CMS, and Intel's libimf [12] (also IEEE compliant) show performance exceeding libm by a factor of more than ten for trigonometric operations with accuracy within an acceptable range. Which of the two libraries will replace libm will be decided after more conclusive tests have been conducted.

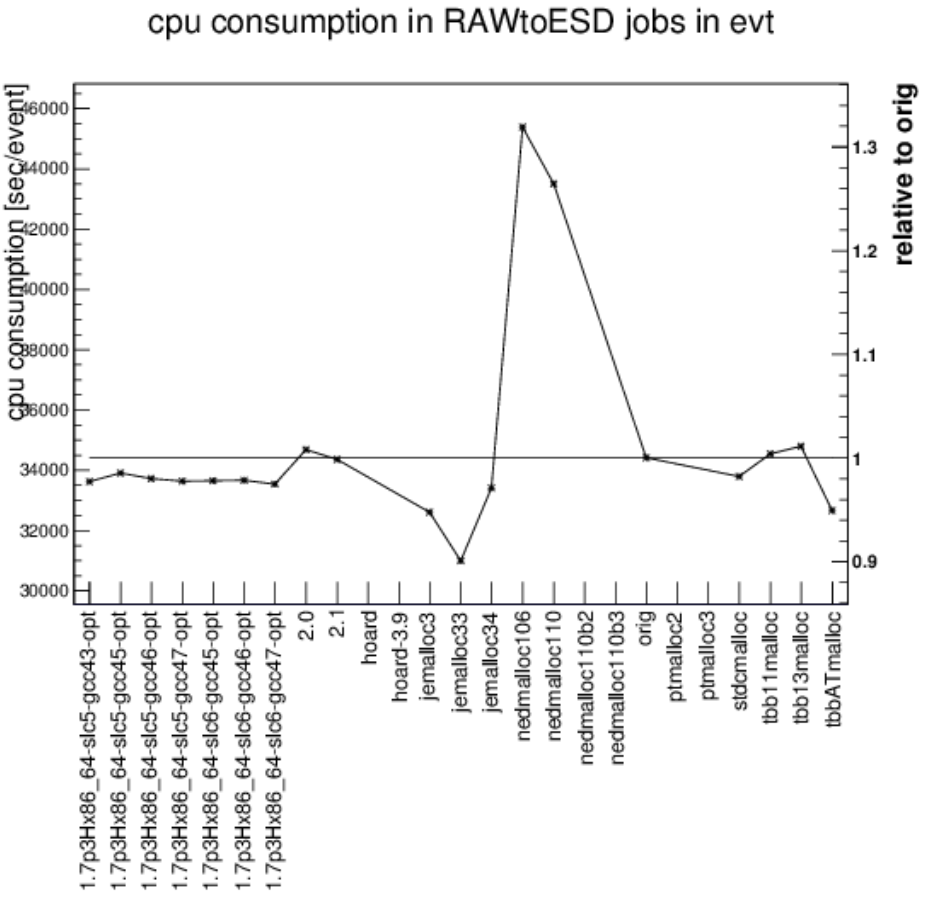

Figure 2. Comparison of CPU consumption of different memory allocators with current allocator tcmalloc as baseline during reconstruction.

2.3.5. tcmalloc. Tcmalloc 0.99 is the memory allocator currently in use. It has originally been selected because it uses the least amount of memory and at the time was faster than or comparable to other allocators. Recent tests have shown that tcmalloc 0.99 does not allocate aligned memory (as also stated in the changelog of tcmalloc 1.00), effectively preventing any vectorization, leading to crashes when using explicit vectorization. Recent tests show several other allocators to be faster. Figure 2 shows up to 10% less CPU consumption for jemalloc [13], an allocator largely developed by Facebook and also used by CMS. Slightly lower memory efficiency is attributed to the allocation of aligned memory. Conclusive tests are still outstanding to decide to which allocator will be used. A comparison of different allocators can be seen in figure 2.

3. Results and Findings

A simplified version of the magnetic field service is in use for track identification. To test whether the new magnetic field service allows better physics performance, the unoptimized coarse-grained magnetic field service was replaced with the speed up version. The new magnetic field allowed finding 0.01% more tracks, which was considered marginal compared to the 10% performance penalty. Preference was given to the development of an improved version of the coarse-grained magnetic field to gain performance. Comparing the magnetic field values retrieved from the new field with those of the old field, differences are <1% or smaller than 1 Gauss, which is lower than the uncertainty of the field values. A similar issue has been investigated with Eigen, where global to local coordinate conversions (and vice versa) show slightly different results than the CLHEP version of these calculations. Comparisons show the results calculated with Eigen to be closer or equally close to the original coordinates than CLHEP. These promising physics performance findings complement the performance gain by the measures taken.

4. Future Projects

Even with optimistic assumptions, the goal will not yet be met with the conducted efforts and ongoing projects. On the other hand, there are many more opportunities to improve speed. Notably, most of the code has not been optimized for recent architectures. Enabling and allowing autovectorization is favored to make use of SIMD registers offered by most CPUs, as other options have been found to make the source code difficult to read. Massaging code to ease autovectorization is being supported by compilers advancing in this field, but will remain a manual task for many cases where vectorization is applicable because of certain assumptions the compilers cannot make. For future optimization projects, applying simple changes such as defining arrays as independent or loop unrolling following simple rules to allow vectorization might be a viable option applicable to a limited number of packages. Few selected hot spots may be hand-vectorized using intrinsics or replaced by other algorithms more suitable for multithreading or vectorization. Other projects might be decreasing calls to malloc/free using data pools to reduce the impact of memory allocation on performance as one of the most frequently used functions.

5. Conclusions

Even though projects of this size pose considerable challenges, not the least of which being the logistic effort to conduct changes on hundreds of packages, the conducted efforts have been rewarded with great results. Some projects demonstrate how easily significant speedup can be achieved, emphasizing the need for continuous testing and evaluation of recent developments. Overall, optimizing many different aspects of the code lead to a huge step towards the goal of a two-fold speedup.

References

[1] ATLAS collaboration – Atlas Computing: technical design report, CERN, 2005, ATL-SOFT-PROC-2013-040, CERN-LHCC-2005-022
[2] PerfMon - http://acode-browser.usatlas.bnl.gov/lxr/source/atlas/Control/PerformanceMonitoring (version from 01/16/2014)
[3] Gperftools - https://code.google.com/p/gperftools/ (version from 01/16/2014)
[4] Gooda - https://code.google.com/p/gooda/ (version from 01/16/2014)
[5] CLHEP - http://proj-clhep.web.cern.ch/proj-clhep (version from 01/16/2014)
[6] Eigen library: http://eigen.tuxfamily.org (version from 01/16/2014)
[7] Tcmalloc - http://goog-perftools.sourceforge.net/doc/tcmalloc.html (version from 01/16/2014)
[8] N.Chauhan et al. - ATLAS Offline Software Performance Monitoring and Optimization, CERN, 2013, ATL-SOFT-PROC-2013-040
[9] Intel Math Kernel Library - http://software.intel.com/en-us/intel-mkl (version from 01/16/2014)
[10] Root - http://root.cern.ch/ (version from 01/16/2014)
[11] Vdt library - https://svnweb.cern.ch/trac/vdt (version from 01/16/2014)
[12] Libimf - http://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/GUID-D7A8B891-FD72-440B-A00D-22F5AC0EF841.htm (version from 01/16/2014)
[13] Jemalloc - http://www.canonware.com/jemalloc (version from 01/16/2014)