US-ATLAS FAST SIMULATION WORKING GROUP PROGRESS REPORT

March 21, 2019

Hasib Ahmed, Zhihua Dong, Heather Gray, Meifeng Lin, Tadej Novak, Kwangmin Yu

February 27, 2019

Introduction

With the planned high-luminosity LHC (HL-LHC), the data rates and therefore data processing demands will be 10 times of the current LHC experiments. However, the anticipated growth of the availalble CPU cycles will remain largely flat. There is a clear need for the modernization and optimization of the ATLAS software stack.

Optimization Challenges

ATLAS software as many other HEP codes has been optimized several times during its life cycle and no obvious performance bottlenecks remain to be addressed. However, the code has been designed for an older generation of CPU only systems and does not work well overall on modern heterogeneous architecture systems, and it does not make use of any type of accelerators (e.g. GPUs) or handle deep memory architectures well.

Besides its large code size, ATLAS software also has several external dependencies, making it challenging to port the code to modern accelerators such as GPUs. In addition, the data being processed are usually fairly large, on the order of tens to hundreds of GBs, so care needs to be taken in terms of data movement for GPUs.

With these challenges in mind, we start with the fast simulation software packages, which tend to have fewer external dependencies. Data requirements in the fast simulations are also smaller, on the order of several GBs. In the current ATLAS workflow, simulation typically takes up more than 50% of the CPU time. In addition to full simulations that are based on `Geant4`, different fast simulation packages have also been developed to reduce the CPU requirement while giving reasonable accuracy of the simulations. `FastCaloSim` [1] is one of these fast simulation packages that was developed to provide fast and relatively accurate simulation of the ATLAS calorimeter system.

- Our goal is to see if we can optimize the `FastCaloSim` package for modern CPUs and/or GPUs, and use it as a prototype for future code optimizations for other ATLAS software packages.
- This would require us to look into the data models in the ATLAS software stack, and make use a programming model that is suitable for `FastCaloSim` in particular and ATLAS in general.

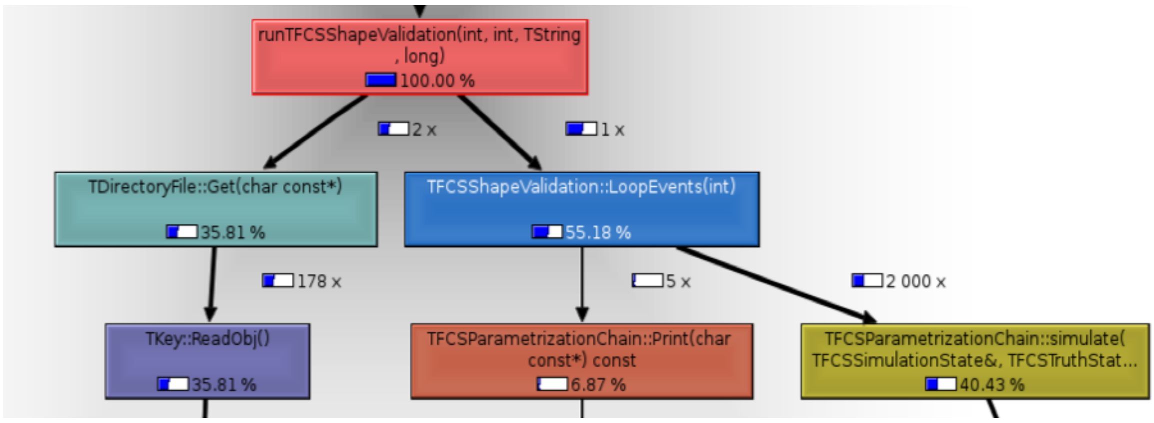
runTFCSShapeValidation(int, int, TString , long)
100.00 %
2 x
1 x
TDirectoryFile::Get(char const*)
35.81 %
TFCSShapeValidation::LoopEvents(int)
55.18 %
178 x
5 x
2 000 x
TKey::ReadObj()
35.81 %
TFCSParametrizationChain::Print(char const*) const
6.87 %
TFCSParametrizationChain::simulate(TFCSSimulationState&, TFCSTruthStat...
40.43 %

Figure 1: Percentage of CPU time in relevant subroutines for `FastCaloSim` with a photon energy of 65 GeV and 50,000 Monte Carlo hits. The run was performed on the Institutional Cluster at Brookhaven National Laboratory, with Intel Xeon "Broadwell" CPUs.

FastCaloSim Performance Analysis

Typically `FastCaloSim` is executed within the Athena framework. To facilitate the code optimization effort, we first isolated `FastCaloSim` from Athena to make it a standalone code. However, the first standalone version only runs under the ROOT interpreter, and is difficult to profile using standard performance profiling tools. The interpreted version would also make it difficult to port the code to GPUs. Considerable effort went into developing a compiled version of `FastCaloSim` code, which has been done successfully recently. We first did some preliminary performance analysis of `FastCaloSim` , to see if there is an obvious bottleneck for optimization. Figure 1 shows the percentage of CPU time spent in some of the main subroutines in `FastCaloSim` for a sample run with photon energy of 65 GeV and 50,000 Monte Carlo hits. Roughly 40% of time is spent on the actual simulation, while about another 40% is spent on data I/O.

We also profiled `FastCaloSim` within the Athena framework. When running the full workload, the CPU consumption of FastCaloSim is only about 3%, while the highest CPU consumer also only takes up about 10%. As mentioned earlier, this is a well-known problem with the ATLAS workloads, with each task taking up a small fraction of the total execution time.

Next Steps

Our next step is to see how we can optimize the simulation part of `FastCaloSim` while also looking into improving the data I/O performance. Given that `FastCaloSim` , and many other ATLAS software packages, depend on the external library ROOT, making the code performance portable will be a challenge. One possibility is to use OpenACC, which will involve less code change but may be difficult to use with complicated C++ codes such as `FastCaloSim` . Another possibility is to look into performance portable frameworks such as Kokkos, which may require major code reorganization and rewrite. Given the C++ nature of the `FastCaloSim` , Kokkos may be the way to go. The suitability of these performance portable programming models is what we will investigate next.

Bibliography

[1] ATLAS Collaboration. The atlas calorimeter simulation fastcalosim. In *IEEE Nuclear Science Symposuim Medical Imaging Conference*, pages 1–5, Oct 2010.