Optimization of the trigger software of the LHCb experiment

Thesis requirements specification

Péter Kardos

2018-2019

1 Background

CERN[1] is a nuclear research organization that hosts the world's largest particle accelerator[2]. The accelerator is a circular tube, with a circumference of about 27km. Batches of protons are circulating in both directions, with the opposite directions intersecting at a few points, giving possibility for the proton beams to collide. This thesis project is focused on the software that processes the data collected during the collisions for the LHCb[3] experiment.

The amount of data generated for collisions is huge, however, only a small fraction of events are worth storing for further investigation. The purpose of the so-called *trigger software* is to process the data in real-time to filter for interesting events, which are further processed offline.

Fast dedicated hardware electronics collect the data from the detector and build so called raw events. These are then decoded and analyzed in multiple stages on a large farm of standard servers. Particle trajectories are reconstructed and analyzed in order to decide whether the event is worth recording based on characteristics such as momentum and type of tracks. This process will happen at a rate of 30MHz from 2021 on, leaving only around 30us of node time for each event reconstruction on average. The size of the server farm will be in the order of 1000 nodes, each node having 40-100 CPU cores, which puts the scale of the challenge into perspective.

As the main code is dating from early 2000s, the underlying framework itself (named Gaudi) was not prepared for such workload and environment. A lot of effort has been put into modernizing it in the past years but the performance is not satisfactory yet. The goal is to achieve a 3 times speedup of the code compared to its current state. (This means a 6 times speedup compared to the code before modernization.)

In order to achieve this ambitious goal, the current software of the LHCb experiment needs to be revamped to take full advantage of modern processor features. Many core systems, low level parallelization, SIMD vectorization and efficiently utilizing superscalar architectures are of particular interest.

Failure to deliver required performance improvements would mean that the number of interesting events reaching the offline analysis stage would be reduced, limiting the physics potential of the overall experiment.

2 Description of the task

The thesis project is focused on optimizing the code running on an individual node of the compute cluster to fully utilize resources available on the node. Nodes host 40-100 modern AMD/Intel CPUs, run a recent Linux distribution, and operate independently of each other.

The work consists of the following main points:

- Benchmark existing code to measure efficiency of the current code base.
- Discover limitations using tools such as Cachegrind, perf and Intel VTune Amplifier.
- Make proposals to improve efficiency: a proposal can be rewriting subparts, reorganizing data structures, new processing algorithms, among others.
- Implement some of these optimizations.
- Measure improvements achieved.
- Share the acquired knowledge with colleagues so that they can make use of the same optimization techniques.

3 Methods

Programming environment:

- C++, latest specification
- Python
- Multi-core systems

Methods:

- AVX2 and/or AVX512 vectorization
- Multi-threading
- Cache and data layout optimization
- Vectorization using external libraries like VC or VCL

Evaluation:

- Cachegrind
- Intel Parallel Studio XE
- perf
- Manual timing measurements

4 Relevant courses

- High Performance Programming
- Parallel and Distributed Programming

5 Delimitations

The work is solely focused on delivering CPU computing optimizations inside a single node.
As such

- GPU processing,
- developing new physics algorithms,
- and distributing the workload across nodes

is out of scope of this project.

6 Time plan

- Learning the environment and the tools (**2 months**)
- Benchmarking and drawing conclusions on the pieces to improve (**2 months**)
- Implementing an improved version of a given piece of code (**3 months**)
- Validating results and fine tuning optimization (**1 month**)

7 References

[1] About CERN:
https://home.cern/about

[2] About the Large Hadron Collider:
https://home.cern/topics/large-hadron-collider

[3] About the Large Hadron Collider beauty experiment:
https://home.cern/about/experiments/lhcb