The core software framework for the LHCb Upgrade

Concezio Bozzi[1,2,*], *Sébastien* Ponce[1,**] and *Stefan* Roiser[1,***]
on behalf of the LHCb collaboration

[1]CERN, 1211 Genève 23, Switzerland
[2]Sezione INFN di Ferrara, Via Saragat 1, 44122 Ferrara, Italy

Abstract. The LHCb detector will be upgraded for the LHC Run 3. The new, full software trigger must be able to sustain the 30MHz proton-proton inelastic collision rate. The Gaudi framework currently used in LHCb has been re-engineered in order to enable the efficient usage of vector registers and of multi- and many-core architectures. This contribution presents the critical points that had to be tackled, the current status of the core software framework and an outlook of the work program that will address the challenges of the software trigger.

1 Introduction

The LHCb experiment will be upgraded in view of data taking during the LHCb Run 3 and beyond [1]. In addition to sub-detector elements and readout channels, the LHCb software and computing infrastructure [2–4] will undergo substantial modifications. In particular, the removal of the lowest-level hardware trigger implies that the High-Level software Trigger (HLT) must be able to cope with the inelastic collision rate of 30 MHz. Moreover, the instantaneous luminosity will increase by a factor 5 to $2 * 10^{34}\ cm^{-2}\ s^{-1}$, with a corresponding increase in pile-up, thereby increasing the processing time for a single event.

The development of high-throughput software, needed by the LHCb trigger to cope with the above conditions and provide a fast decision, has been the subject of an intense R&D program that ultimately relies on the better utilization of modern computing architectures. This paper reports the work that has been done in this respect. The main directions that have been explored are detailed in Sec. 2. The tools that have been put in place to monitor progress and the current results are reported in Sec. 3. An outlook towards the goal of sustaining software trigger capability at a 30 MHz rate is discussed in Sec. 4. Many of these aspects have been detailed in Ref. [2].

2 Main directions

Even though the number of transistors on integrated circuit chips has continued to follow Moore's law over the last decade, and will probably for the next few years, the trend of ever faster single-threaded CPU performance has been broken about ten years ago. Instead, the

[*]e-mail: concezio.bozzi@cern.ch
[**]e-mail: sebastien.ponce@cern.ch
[***]e-mail: stefan.roiser@cern.ch

additional transistors have been invested into multiple CPU cores on a die, and, within these cores, into wider execution units. Furthermore, the gap between the processing units (PUs) and memory has increased (Figure 1), and feeding the processing units with data is becoming more than ever a bottleneck.

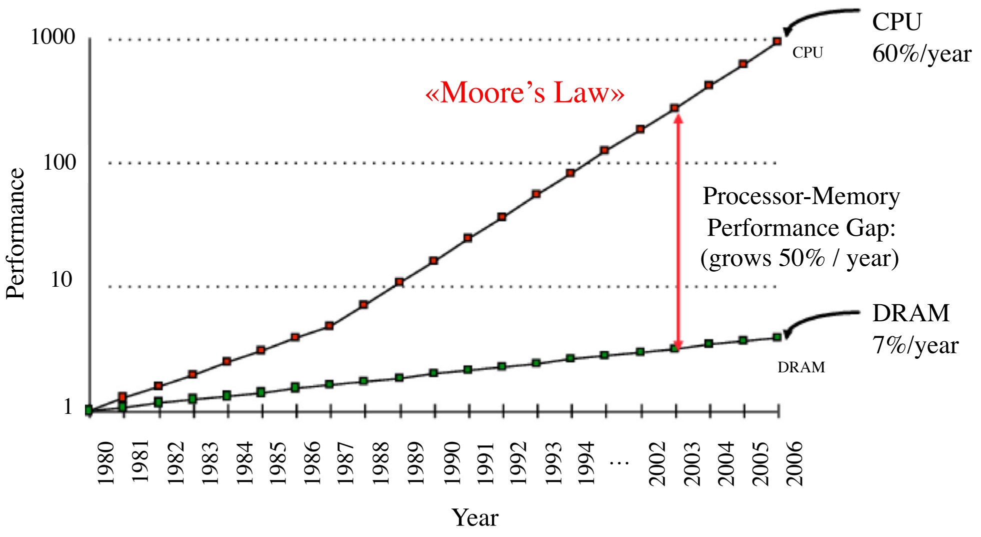


Figure 1. Performance of processors and memory as a function of time [5, 6]. The gap between process and memory performance grows about 50% each year.

Besides, the amount of RAM available per core has been stable, if not going down, recently with typical setups of up to 40 or 64 virtual cores for 64 to 128GB of memory. Therefore the multi-process approach, where multiple instances of the same application are run in parallel, one per processing core, is no longer sustainable due to lack of RAM – a typical LHCb application needs 2-4GB to run.

The LHCb core software framework Gaudi [7] has thus been re-engineered towards multithreading so that the parallelism in the processing cores is exploited. LHCb specifically takes benefit of the inter-event parallelism by running as many events in parallel as cores. This re-engineering process meant the implementation of thread-safety, by further developing pioneering work previously done within Gaudi [8], via the definition of *functional* algorithms [9], the implementation of a new scheduler [10], and the reliance on modern C++ standards.

The LHCb software stack was then gone through step by step, first through small-scale tests in order to learn these new programming techniques and paradigms. Software optimization guidelines include:

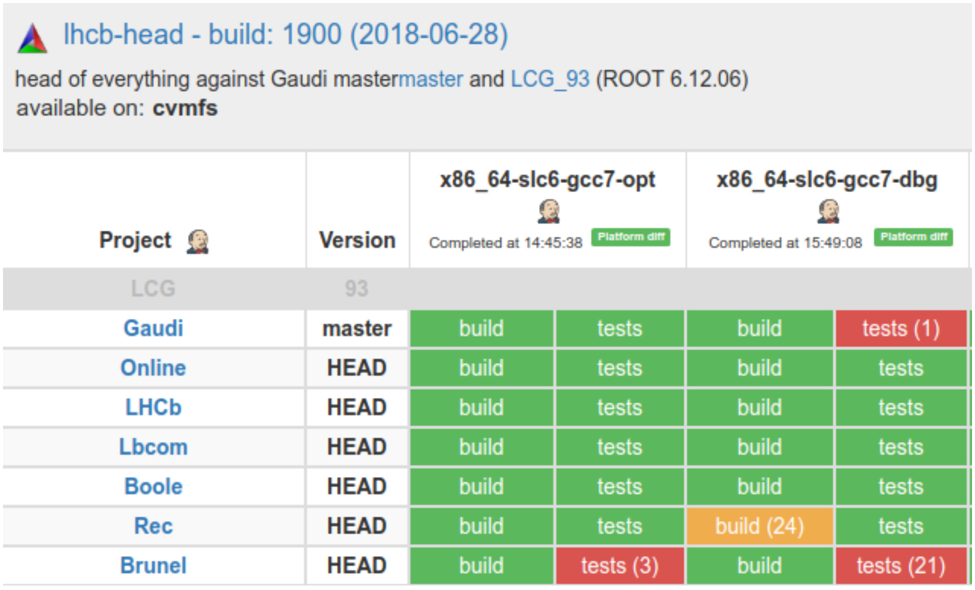
- a careful usage of memory, in order to avoid costly cache misses, which this implies e.g. the redesign and slimming of data classes;
- the utilization of vector processing units, which requires the rethinking of data models and data preparation steps;
- the introduction of new C++ features, in particular those related to optimization;
- the extensive use of constness throughout the code in order to ensure thread safety at compile time as much as possible.

The above guidelines require extensive knowledge of modern programming techniques and thus a large software training effort within the LHCb collaboration. To this end, tutorial sessions are held regularly on topics such as core C++ programming, the LHCb framework, coding practices, optimization tools and collaborative tools. Also, bi-monthly hackathons are held in order to pair code developers with software experts, and make rapid progress.

3 Monitoring the progress

As already mentioned, one of the main objectives of the LHCb upgrade for Run 3 is to process events from LHC collisions at a rate of 30 MHz by using a fully software trigger. In order to meet this objective the evolution of the HLT performance must be followed on a regular basis. In the case of a trigger farm consisting of 1000 nodes, this means that a single node should process about 30k events per second.

The first benchmark tests performed on the HLT1 application, the first HLT stage that runs upfront synchronously with data taking, showed an event processing rate of about 3k events per second only. Efforts were therefore prioritised to port the HLT1 algorithms to the modernized Gaudi framework sketched above, in order to better exploit the previously unused dimensions of computing resources. The HLT1 algorithms that perform the event reconstruction were reviewed and converted in order to be run in a fully multi-threaded approach. Then, profiling tools such as valgrind, callgind, and vtune were used to identify hotspots in the code and algorithmic improvements were subsequently implemented. The LHCb build infrastructure and continuous integration platform [2] is used (Figure 2) to nightly-build and -check code validity on all computing platforms, to check the overall code performance, and to record all performance indicators, including cache misses and vectorization levels. Trends can be extracted at the single algorithm level, an example being shown in Figure 3.


lhcb-head - build: 1900 (2018-06-28)

head of everything against Gaudi mastermaster and LCG_93 (ROOT 6.12.06)
available on: **cvmfs**

Project	Version	x86_64-slc6-gcc7-opt Completed at 14:45:38 Platform diff		x86_64-slc6-gcc7-dbg Completed at 15:49:08 Platform diff	
LCG	93				
Gaudi	master	build	tests	build	tests (1)
Online	HEAD	build	tests	build	tests
LHCb	HEAD	build	tests	build	tests
Lbcom	HEAD	build	tests	build	tests
Boole	HEAD	build	tests	build	tests
Rec	HEAD	build	tests	build (24)	tests
Brunel	HEAD	build	tests (3)	build	tests (21)

Figure 2. Snapshot of the outcome of a nightly build from the LHCb continuous integration platform. Software is organized in projects (rows), computing platforms correspond to columns.

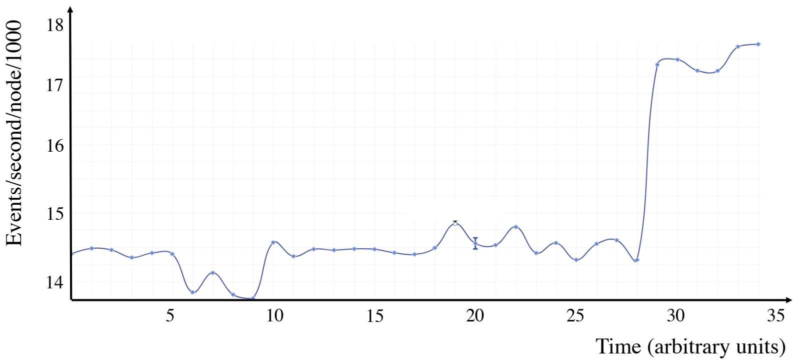


Figure 3. Trend plot of the number of events processed per second by a given HLT1 algorithm.

Figure 4, taken from [2], shows the throughput of the HLT1 sequence as measured on a dedicated machine with 20 physical cores, on which the sequence is run multiple times, each time by varying the number of reconstruction processes and the number of threads in each process. At the time of this Conference, a throughput of 15k events per second on a

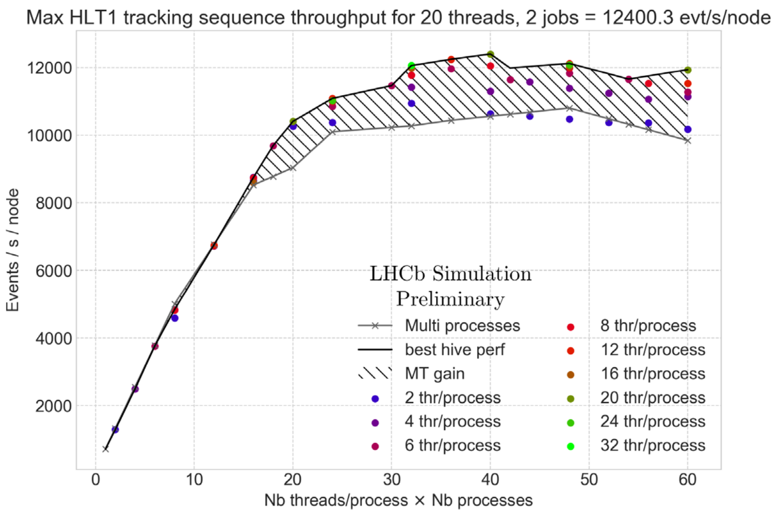


Figure 4. (Taken from [2]) Throughput of an HLT1 reconstruction sequence, measured on a dedicated machine with 20 physical cores, as a function of the product of the number of processes and number of threads, for different number of threads per process, as indicated in the legend. The throughput peak performance is 12400 evt/s/node for 2 processes and 20 threads per process. The "Multi processes" line indicates the performance that is achieved without multithreading.

single computing node was obtained. A snapshot of the fraction of time spent in each HLT1 algorithm is shown in Figure 5.

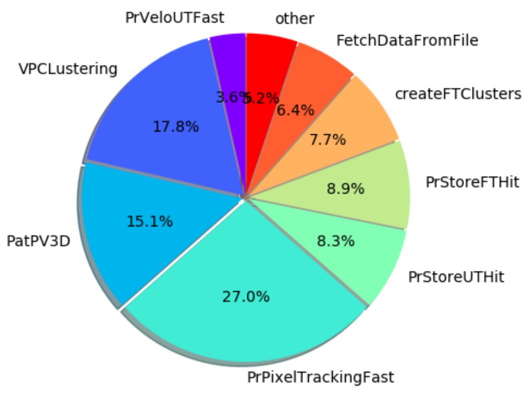


Figure 5. Fraction (in percent) of time spent in each algorithm of an HLT1 application.

4 Conclusion and outlook

This paper has shown the infrastructure that has been put in place and that will allow the LHCb experiment to meet the ambitious target of building and operating a 30MHz software trigger. This includes in particular a substantial training effort in order to spread the knowledge of new software engineering techniques and speed-up the work plan.

The efforts start to pay off: the performance of the HLT1 sequence, the first trigger stage that has to cope with an input rate of 30MHz, has improved a lot from the original 3k to 15k events per node in the latest measurements. However the effort needs to continue and many items have to be tackled, among which the trigger selections, the second trigger stage (HLT2) and a deep review of the internal event model.

References

[1] LHCb collaboration, *Framework TDR for the LHCb Upgrade: Technical Design Report*. LHCb-TDR-012

[2] The LHCb Collaboration, *Upgrade Software and Computing Technical Design Report*. CERN-LHCC-2018-007. LHCB-TDR-017

[3] The LHCb Collaboration, *Computing Model of the Upgrade LHCb experiment Technical Design Report*. CERN-LHCC-2018-014, LHCb-TDR-018

[4] The LHCb Collaboration, *LHCb Trigger and Online Technical Design Report*. CERN-LHCC-2014-016, LHCb-TDR-016

[5] J.L. Hennessy, D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th edn. (Morgan Kaufmann, Amsterdam, 2012), ISBN 978-0-12-383872-8

[6] J.D. McCalpin, *Memory Bandwidth and System Balance in HPC Systems*, Invited Talk, International Conference for High Performance Computing, Networking, Storage, and Analysis (2016)

[7] LHCb collaboration, *LHCb computing: Technical Design Report.* LHCb-TDR-011

[8] *The Concurrent Framework Project (CF4Hep)*, http://concurrency.web.cern.ch/GaudiHive (2012)

[9] M. Clemencic, B. Hegner, C. Leggett, Journal of Physics: Conference Series **898**, 042044 (2017)

[10] R. Matev et al., *A 30 MHz software trigger for the LHCb upgrade*, in *the procedings of this Conference* (2018)