Performance benchmark of LHCb code on state-of-the-art x86 architectures

D H Campora Perez¹, N Neufeld¹ and R Schwemmer¹

¹ CERN, CH-1211 Geneva 23, Switzerland
E-mail: rainer.schwemmer@cern.ch

Abstract. For Run 2 of the LHC, LHCb is replacing a significant part of its event filter farm with new compute nodes. For the evaluation of the best performing solution, we have developed a method to convert our high level trigger application into a stand-alone, bootable benchmark image. With additional instrumentation we turned it into a self-optimising benchmark which explores techniques such as late forking, NUMA balancing and optimal number of threads, i.e. it automatically optimises box-level performance. We have run this procedure on a wide range of Haswell-E CPUs and numerous other architectures from both Intel and AMD, including also the latest Intel micro-blade servers. We present results in terms of performance, power consumption, overheads and relative cost.

1. Introduction
The LHCb High Level Trigger (HLT) farm [1] is a data centre which is located at P8 of the LHC accelerator. The farm consists of about 1800 industry standard computing servers. These servers are typically used in High Performance Computing (HPC) and other compute intensive applications and are optimised to give a maximum of raw CPU performance per price. Since the applications we are running are not sensitive to having a homogeneous compute cluster we are continuously adding new machines when a sufficient amount of funds become available. As a result approximately 80% of the machines are currently based on several generations of Intel’s Xeon processor family, the other 20% being AMD Opteron machines.

The detector is producing data at a rate between 50 to 60 GB/s and an event rate of 1 MHz. Since it was not cost effective at the time the experiment was designed to transfer this amount of data from the 100 m underground cavern to the surface, the data centre was built right next to the detector, protected by a radiation shielding wall. The downside of this is that power and cooling is limited and the cluster can only grow up to those limits.

Due to additional physics requirements and the luminosity upgrade of the LHC in the upcoming Run 2, we will have to add a significant amount of compute capability. We expect to have to add an additional 70% of our current compute power to the cluster in order to keep up with the increased requirements. At the current event complexity this would correspond to an increase in trigger rate of 700 kHz. At the same time this compute capability will have to fit into a power and cooling envelope of 200 kW.

In order to spend our available funds as effective as possible we have turned our trigger software into a stand alone benchmark application. This benchmark can be installed on a Live DVD and distributed to platform integrators to test various computing platforms for their efficiency and optimise trigger decision throughput vs. price and power demands. As a positive side effect this benchmark also serves as a basis for optimising machine settings to get the best out of our current machines.

It should be noted here that we have decided against using HEP-SPEC since we have found discrepancies on the order of 10% between what HEP-SPEC reported and our application achieved on various platforms.

In the following sections we will show how we turned our application into a benchmark, the results we obtained for running the software on different platforms and improvements we made to the machines in order to maximise trigger throughput.
2. Benchmark Software
In order to get the best estimate of the machine performance running our HLT, we have turned the application itself into a benchmark. A one touch control program was written in python, which launches all the necessary processing, monitoring and control jobs. The program then monitors the amount of trigger decisions the machine processes over a period of time. In order to get well reproducible values a specific test is typically run for an hour which results in a statistical error of less than 2% and mostly below 1%. This high resolution is necessary because these values are used in a public tender to judge which company offers the best price per performance. Since the quoted prices are usually very close together we need to be able to tell the difference at least at this level.

The control part of the benchmark also automatically takes care of running various configurations of processes, warm-up periods and error handling depending on the machine configuration. Once the benchmark is shipped to the integrator company, they only need to run this program on their compute platforms and send back a result file.

2.1. Trigger Application
The LHCb trigger software itself is heavily based on a producer-consumer pattern, illustrated in fig.1. At the very beginning is a process which receives data fragments from the detector read-out boards and assembles them into work units called events. A part of these events is written out to a local hard disk on the machine. It is used later to keep the machine busy while the accelerator is recharging [2].

The next step is the trigger process itself which makes up the bulk of the CPU time consumed on the machine. It is a single threaded program which is supplied with the stream of events. These events have sizes on the order of 50 to 60 kB. The application partially analyses the physics content of the event and makes a decision on the usefulness of the data. Parallelism is achieved by running multiple instances instead of using multi-threading. The instances are fed with data from a shared memory buffer. One key observable of the benchmark is to find out what the optimal number of instances is for a particular machine.

If the event is deemed interesting by the application it is sent to a writer program which then forwards the event to a central storage system where it is written to disk and later to tape. For our benchmark we had to replace the receiver process, which normally receives data from a network card. We used the process which is run when the accelerator is offline and reads data from the disk instead. Since this process is rather lightweight compared to the trigger itself (less than 1% of the total processing time on the machine), we don’t expect this to bias our performance measurements too much.

Care had to be taken however to make sure that the sample files we used as input where pre-loaded into cache. The benchmark is also intended to be run from a bootable DVD in which case waiting for I/O can become an issue. To preload the sample input data into cache the benchmark runs for a configurable warm up time without taking the results into consideration. During this time all input files are read and put into the Operating System’s (OS) file system cache. We also dropped the sending process and replaced it with a process which discards the accepted events. Again this is a lightweight process which has no significant impact on the measurements.

2.2. Application Profile
The trigger application itself is a typical program designed at a time where multi-core CPUs where just about to enter the market [3]. It uses multi-threading only to logically separate control and processing and not for performance reasons. At the time SIMD was also not yet as prevalent as today and there is no provisioning for Vectorization in the code either.
As a result the application makes very bad use of modern CPUs. A particular event needs to be processed from start to finish before the next event can be loaded into memory and processed. This linear processing prevents the arrangement of events in a way that would allow to exploit locality of code and data. It also strongly discourages Vectorization due to the Array of Structure paradigm used at the time.

The lack of SIMD creates two particular problems. At first, the use of vector units usually forces programmers into organising data structures in a way that also helps with how data is loaded and stored from main memory. Without this organisation constant cache misses are the result and memory latency becomes a very important factor in process speed.

The second problem is the use of branching. At the time the program was designed, floating point throughput was limited and programs followed an abort early paradigm. Performance is achieved by constantly checking if a certain calculation is necessary. If it is not, it is skipped. With vector units giving ample amount of floating point throughput this is much less of an issue anymore. The constant and random branching upsetting the branch prediction units of the CPU has reached a point where it would be better to just do the calculation anyway and throw the result away.

On the memory side the application has a footprint of approximately 1.1 GB. About 600 MB of this are static data structures which are read only, while 400 to 500 MB are dynamic and are constantly read and written. To reduce the amount of installed memory in the machine, the trigger processes are created by spawning child processes of a main process via the fork() system call [4]. This cloning is done after most of the static data structures have been loaded. With this method the static memory can be shared between all instances, significantly reducing the overall memory consumption and cost for installing memory into the machine.

2.3. Benchmark Image Creation

![Figure 1. Architecture of the LHCb HLT application](image1.png)

![Figure 2. Sample plots showing machine throughput vs. number of running instances.](image2.png)

In order to fit the whole benchmark suite onto a live DVD it was necessary to significantly shrink down the on-disk footprint of the application. A DVD has about 4 GB of space. Approximately 1 GB is used for the OS. We used the default SLC 6.5 live DVD from CERN and removed unnecessary packages, man pages, localisation files and other clutter. Another 2.2 GB are consumed by input files which contain sample events gathered from LHC Run 1. This leaves a little under 1 GB of space for the trigger application, conditions database snapshots, magnetic field maps and other relevant files.
Since the application shares a lot of code and libraries with our offline software, it is not obvious which pieces are used for the trigger and what is used for offline processing. The distribution also contains trigger lines and other code that is not used by the current configuration of the program but still in the repository weighing in at about 15 GB in total.

In order to only select what is absolutely necessary we used `strace`, a program which allows to intercept system calls in a running process. While the application is starting up and running, we are intercepting all open(), stat() and execv() calls to find out exactly which files are needed. A python script parses the output and copies all touched files and symlinks into a parallel file tree on the DVD image. The result is a DVD which has the same path environment as one of our farm nodes but only contains the files that are absolutely necessary to run the HLT application.

### 3. Benchmark Results

All platforms shown here except for the Avoton C2750 are dual socket machines. Given the pricing of quad socket capable CPUs, we did not expect them to give us any advantage on our software so we excluded them from the start.

#### 3.1. Application and Platform tuning

The creation of this benchmark was not only useful for deciding which compute platform to buy, but also helped solve and illustrate other issues on our current farm. Fig. 2 shows a typical set of graphs produced by the benchmark. The plot shows the throughput of trigger decisions vs. the number of running HLT instances on five different machines. The X5650 and Opteron 6272 machines make up the bulk of our current farm. The E5-2650 v3 and E5-2630 v3 are potential new candidates we tested. These plots illustrate very well the effect of Hyper Threading (HT) on Intel CPUs and the lack of it on AMD CPUs.

The Intel CPUs have 5 distinct areas explained at the example of the E5-2650 v3 machine. For the first 10 instances machine throughput rises very quickly. The derivative on the bottom half shows the effect of Intel’s Turbo Boost technology, which increases core frequency when only very few CPU cores are used. From 10 to 20 we can see the OS starting to put jobs onto the HT cores of socket 1. Throughput increase diminishes and the statistical error of each measurement increases due to fluctuations caused by HT.

From 20 to 30 instances the OS starts scheduling jobs on socket 2 and there is again a strong increase visible in the derivative plot on the bottom. However, performance does not increase as much as it did on socket 1. From 30 to 40 instances performance gain tapers off again due to HT until we reach a plateau at 40. This is the maximum number of virtual cores the CPU has to offer and improvements are only marginal from here on. Nevertheless a few more percent can be gained by over-committing the number of running instances.
Two interesting effects can be seen here. First, our application gains almost 50% more performance from HT alone which is due to the effects described in the Application Profile section. Branch mispredictions and bad memory layouts are stalling the instruction pipeline which is where the second execution unit comes in and picks up the work.

The second interesting effect is that performance on socket 2 does not scale as well as on socket 1. The explanation to this is due to Non Uniform Memory Access (NUMA) and illustrated in fig. 3. We are starting one master process which allocates a large amount of static data structures. These data structures end up in the memory of the socket that spawns the master. All child processes that are running on the other socket will incur a certain latency penalty by having to go through the inter processor link. The OS detects this circumstance and decides to schedule processes on the HT cores of socket 1 first.

To alleviate this problem we are now spawning one master process per CPU socket and use numactrl to lock them and all their children to that socket. Table 1 shows examples of how much performance can be gained on a select sample of platforms with up to an impressive 14% on the new 8 core Haswell CPU. This also works on AMD platforms with more than one socket giving an 8% improvement.

It should be mentioned here that many Haswell with more than 10 cores have a feature called Cluster On Die (COD) which separates the cores on a single socket into two memory domains with independent memory controllers. This is usually disabled in the BIOS of the machine and needs to be switched on. Another improvement of 3% was achieved on the 10 core machines we tested when running NUMA aware to COD.

| CPU                  | Decisions/s No NUMA | Decisions/s NUMA | Gain  |
|----------------------|---------------------|------------------|-------|
| Intel X5650 (8 cores)| 599.6               | 648.8            | 1.08  |
| Opteron X6272        | 623.4               | 682.0            | 1.08  |
| E2630 v3 (8 cores)   | 865                 | 986              | 1.14  |
| E2650 v3 (10 cores)  | 1129                | 1210             | 1.07  |

Table 1. Improvements in machine throughput by running the HLT application NUMA aware.

3.2. Memory related Issues

Another issue related to memory accesses is illustrated in fig. 4, which shows the scaling of performance within a family of processors with a fixed number of cores but different core frequencies. Each data series consists of two plots. The solid lines show the measured performance while the dashed lines show what the performance should look like if the first point in the series was scaled up linearly with CPU frequency. On our current Westmere machines the scaling is almost exactly as expected. On the new Haswell platform however there is almost no gain for the 10 core CPUs and only slight improvement for the 12 core CPUs.

We are suspecting that the latency of memory has finally caught up with our application. Fig. 6 was recorded running Intel’s Performance Counter Monitor [5] while running the HLT application. The plots show various cache related observables vs. the number of running instances. Of particular interest here is the rate of L3 Misses and lost clock cycles due to L3 misses (middle row). Even though Intel has managed to improve their cache management slightly between Westmere and Haswell, we still have approximately 50% of loads that need to be fetched from main memory. On top of that the L3 Clock Lost plot shows that clock cycles lost from cache misses have gone up dramatically from 15% for Westmere to almost 50% for Haswell. This measurement shows just how important it is today to really optimise data accesses in applications.
Figure 4. Scaling of performance vs. Core Clock Frequency.

Figure 5. Memory CAS (Column Strobe Access) latency vs. Memory base frequency for DDR3 and DDR4.

Figure 6. Number of instances vs. memory subsystem values. While the number of cache misses have gone down in newer generation CPUs, the penalty in clock cycles for every miss has gone up dramatically.

A plot with specified latencies vs. memory frequency from one manufacturer [6] can be found in fig. 5. The latency of DDR4 is significantly worse than DDR3 and so every cache miss will incur additional lost clock cycles over DDR3. DDR4 is a new technology and latencies might get better in the future though.
3.3. Performance vs. Price
For LHCb the first most important question of this endeavour is how much is our target computing capability of an additional 700 kHz of trigger decisions going to cost. For this purpose we have sampled a wide range of Haswell CPUs. We have also looked at the new Avoton and AMD’s latest Opteron series of processors. The raw performance values per machine are summarised in fig. 7. Since we could not measure all systems we assumed that for every generation of CPU the performance value per GHz and core is fixed. We then requested access to samples and price quotes for the cores that we thought might be interesting and verified the performance. As expected the non scaling of higher frequency cores causes some discrepancies for the cores with high core clocks between the predicted and actual value.

![Predicted / Measured performance](image)

**Figure 7.** Raw machine performance vs. CPU type.

Since we are interested in the price, we then normalised these values according to the cost for 700 kHz of trigger performance. The result can be seen in fig. 8. Since these are confidential quotes we cannot disclose the actual monetary value on the axis. Suffice it to say, since Intel is right now dominating the market for CPUs the prices are pretty much in line with what can be found on their web site [7].

3.4. Performance vs. Power Consumption

![Cost for fixed throughput](image) ![Power Consumption for fixed throughput](image)

**Figure 8.** Price at fixed throughput (left) and power consumption (right) of compute platforms.

Figure 8 also illustrates the power consumption of a finished installation that can do 700 kHz. Since most of the machines we had access to where hosted by external companies, we did not have access to power measurement equipment for most of them. Again we had to rely on a model which is simply $TDP \times N_{sockets} \times 1.6$, where TDP is the Thermal Design Power rating of the
CPU. The factor of 1.6 is an empirical value we found from measuring the following platforms: Avoton C2750, Opteron 6376, E5-2630 v3, E5-2690 v3

Within these four platforms the error for this formula is less than 5%.

3.5. Avoton

Even though we are strongly constrained by power consumption, in our case the Avoton is not particularly better than the high end Xeon CPUs. The reason for this lies again with the memory sensitivity of our application. The Avoton CPU has no L3 cache and also no Hyper Threading. This leads to a disastrous Instructions per Clock (IPC) value of less than 0.4. Even if the application could be optimised for this sort of architecture it would be difficult for us to deploy this platform.

We need local disk storage on every node. The cost would be driven up significantly by having to buy small form factor HDDs. Another non negligible contribution to the high price comes from the additional networking infrastructure necessary to service many slow machines. Overall the processor seems to be very good though, if these issues can be overcome or are not a concern. An IPC of 1.0 for example would have brought the price down to a workable level.

4. Conclusions

In this paper we have shown how we turned our HLT application into a benchmark. We used this benchmark to optimise our currently installed computing resources and also used it for a public tender for buying an additional 700 kHz LHCb trigger equivalent compute power.

Profiling of the benchmark has shown that our application is very sensitive to memory latencies. These latencies can be caused by either slow memory or NUMA related effects. We managed to improve the performance of our application by 8% on our already installed farm and 14% on the newly acquired farm nodes. Other optimisation efforts towards parallelism and SIMD utilisation are on the way [8] but will not be ready for LHC Run 2.

In the end we decided to purchase 800 dual socket machines based on the E5-2630 v3 CPU, which gave the best performance for money and power consumption for our application. AMDs current processors are still competitive in cost but not in power efficiency. The Avoton based micro server was found not to be suitable because of issues in our memory access pattern.

5. Acknowledgements

We would like to thank our colleagues from Cern IT and our industry partners from E4 Computer Engineering, Intel Swindon Labs, ASUS and Macle GMBH for their help and the early access they provided to the compute platforms we tested.

References

[1] F. Alessio et al. LHCb online event processing and filtering. J. Phys. Conf. Ser., 119:022003, 2008.
[2] Markus Frank, Clara Gaspar, Niko Neufeld, and Beat Jost. Deferred High Level Trigger in LHCb: A Boost to CPU Resource Utilization. In D. Groep, editor, Proceedings of the 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013), 2014.
[3] Barrand G. et al. GAUDI - A software architecture and framework for building LHCb data processing applications. In Proceedings of CHEP 2000, 2000.
[4] Frank M. et al. Optimization of the HLT Resource Consumption in the LHCb Experiment. In Proceedings of CHEP 2012, 2012.
[5] P. Fay T. Willhalm, R. Dementiev. Intel Performance Counter Monitor - A better way to measure CPU utilization. [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-counter-monitor, 8 2012.
[6] Corsair Inc. From DDR3 to DDR4: Bandwidth by the Numbers. [Online]. Available: http://www.corsair.com/en/blog/2014/september/ddr3_vs_ddr4_synthetic, 9 2014.
[7] Intel Inc. Intel Processor Database. [Online]. Available: http://ark.intel.com, 5 2015.
[8] D. H. Campora Perez et al. SIMD studies in the LHCb reconstruction software. This Conference, 5 2015.