First experiences with a parallel architecture testbed in the LHCb trigger system

S Gallorini\textsuperscript{1}, D Lucchesi\textsuperscript{1}, A Gianelle\textsuperscript{1}, S Amerio\textsuperscript{1} and M Corvo\textsuperscript{2}

\textsuperscript{1} INFN and Università di Padova, Via Marzolo 8, 35131, Padova, Italy
\textsuperscript{2} INFN and Università di Ferrara, Via Saragat, 1, 44122 Ferrara, Italy

E-mail: stefano.gallorini@pd.infn.it

Abstract. In view of Run3 (2020) the LHCb experiment is planning a major upgrade to fully readout events at 40 MHz collision rate. This in order to highly increase the statistic of the collected samples and go further in precision beyond Run2. An unprecedented amount of data will be produced, which will be fully reconstructed real-time to perform fast selection and categorization of interesting events. The collaboration has decided to go for a fully software trigger which will have a total time budget of 13 ms to take a decision. This calls for faster hardware and software. In this talk we will present our efforts on the application of new technologies, such as GPU cards, for the future LHCb trigger system. During Run2 a node equipped with a GPU has been inserted in the LHCb online monitoring system; during normal data taking, a subset of real events is sent to the node and processed in parallel by GPU-based and CPU-based track reconstruction algorithms. This gives us the unique opportunity to test the new hardware and the new algorithms in a realistic environment. We will present the setup of the testbed, the algorithms developed for parallel architectures and discuss the performance compared to the current LHCb track reconstruction algorithms.

1. Introduction
The LHCb experiment is starting the phase of upgrade of its detector to allow the collection of data at luminosity of $2 \cdot 10^{33}$ cm$^{-2}$s$^{-1}$ at a centre-of-mass energy of 14 TeV. For this upgrade, the tracking system and the Ring Imaging CHerenkov (RICH) detector, used for particle identification, will be replaced. The DAQ system will be redesigned around a trigger-less readout, which allows the full inelastic collision rate of 30 MHz to be processed in the Event Filter Farm (EFF). One of the main limitations of the current trigger system is the L0 hardware trigger which limits the input rate to the HLT (High Level Trigger) to 1.1MHz. It is this initial reduction which causes the largest inefficiencies, especially for purely hadronic decays. Therefore, the main purpose of the LHCb upgrade is to remove this bottleneck by implementing a full software trigger able to process the full collision rate. We might expect improvements in execution times coming from a more efficient use of the multicore structure and parallelization. Therefore, sensibility advises to build up a program to study and exploit the possibilities of parallelization of the algorithms involved in the trigger. Among the candidate architectures to support these algorithms we find General Purpose Graphics Processing Units (GPGPUs), specialized for compute-intensive, highly parallel computation. GPGPUs may offer a solution for reducing the cost of the HLT farm for the LHCb upgrade and R&D studies have started to evaluate the possible role of this architecture in the new trigger system.
In the following sections we will discuss our attempt to port the reconstruction algorithm of the LHCb silicon vertex detector on GPU and the effort to integrate accelerators in the LHCb online system. During Run2, a desktop PC, equipped with an NVidia Titan X GPU, has been installed in the LHCb monitoring farm to assess the performances of the new architecture on a real-time environment.

2. FastVelo

2.1. Description of the algorithm

The VELO [1] is a silicon strip detector that provides precise tracking very close to the interaction point. It is used to locate the position of any primary vertex within LHCb, as well as secondary vertices due to decay of any long lived particles produced in the collisions. The VELO detector is formed by 21 stations, each consisting of two halves of silicon-strip sensors, which measure $R$ and $\phi$ coordinates. A sketch of the VELO detector is shown in figure 1.

“FastVelo” [2] is the algorithm developed for tracking of the current VELO and was written to run online in the HLT tracking sequence. For this reason, the code was optimized to be extremely fast and efficient in order to cope with the high rate and hit occupancy present during Run1-Run2 data taking. FastVelo is highly sequential, with several conditions and checks introduced throughout the code to speed up execution and reduce clone and ghost rates.

![Figure 1. A sketch of the current VELO detector.](image)

The algorithm can be divided into two well-defined parts. In the first part (RZ tracking), all tracks in the RZ plane are found by looking at four neighbouring R-hits along the $z$-axis (“quadruplet”). The quadruplets are searched for starting from the last four sensors, where tracks are most separated. Then the quadruplets are extended towards the lower $z$ region as much as possible, allowing for some inefficiency. In the second part of the algorithm, the full tracks are built by adding the information of the $\phi$ hits to the RZ track. The final 3D track is re-fitted using the information of $R$ and $\phi$ hits, while hits with the worst $\chi^2$ are removed from the track. Hits already used in a track are marked as used and not further considered for following iterations (“hit tagging”); this is done to reduce the number of clones produced by the algorithm, avoiding encountering the same track several times.
2.2. GPU implementation

The strategy used for porting FastVelo to GPU architectures takes advantage of the small size of the LHCb events (≈ 60kB per event, ≈ 100 kB after the upgrade) implementing two level of parallelization: “of the algorithm” and “on the events”. In principle, with many events running concurrently, it can be possible to obtain additional speed-up with respect to the only parallelization of the algorithm. The GPU algorithm was adapted to run on GPU using the NVIDIA Compute Unified Device Architecture (CUDA) framework [3].

One of the main problems encountered in the parallelization of FastVelo concerns hit tagging, which explicitly spoils data independence between different concurrent tasks (or “threads” in CUDA language). In this respect, any implementation of a parallel version of a tracking algorithm relying on hit tagging implies a departure from the sequential code, so that the removal of tagging on used hits is almost unavoidable. The main drawback of this choice is that the number of combinations of hits to be processed diverges and additional “clone killing” algorithms (intrinsically sequential and not easy to parallelize) have to be introduced to mitigate the increase of ghost and clone rates.

2.3. Physics performances on simulated events

The GPU model used for these tests is an NVidia GTX Titan (14 SMX, each equipped with 192 single-precision CUDA cores), while the CPU is an Intel(R) Core(TM) i7-3770 3.40 GHz. A simulated sample of $B_s \rightarrow \phi \phi$ events generated with 2012 Run1 conditions (with pile-up of $\nu = 2.5$) has been used to evaluate the tracking and timing performances. The efficiencies obtained by FastVelo on GPU are quite in agreement with the sequential FastVelo; in particular, clones and ghosts are at the same level of the original code. Figure 2 shows the tracking efficiency as a function of the true track momentum as obtained by the two algorithms; the overall agreement is good, showing that the GPU implementation does not introduce any distortion on the resolution of the track parameters. The speed-up obtained by the GPU algorithm with respect to FastVelo running on a single CPU core as a function of the number of processed events is also shown in figure 2. The maximum speed-up obtained by the GPU algorithm with respect to the sequential FastVelo is $\approx 3 \times$ for the 2012 datasets. The speedup as a function of the number of events can be explained by the fact that the GPU computing resources are more efficiently used as the number of events increases (there are more threads running at the same time). The GPU performance has been compared to a full CPU using our testbed where we measured the throughput during the data acquisition in a more realistic environment.

3. Testbed architecture

3.1. The Monitoring Farm

The Monitoring Farm (MF) works with a random sample of raw events which passed a loose selection imposed by the Level0 hardware trigger. The events sent to the MF are then processed by the HLT software and the results of the reconstruction made available to the relevant monitoring tasks to produce monitoring histograms for an online validation of the incoming data. The average rate of events feeding the monitoring nodes is $O(10 \text{ Hz})$. The overall scheme of the HLT and MF is shown in figure 3.

3.2. Testbed hardware

The testbed installed in the MF consists of a standard desktop PC equipped with an Intel i7-4790 CPU @ 3.60GHz (12 physical cores with hyper-threading), hosting a GPU NVidia GTX Titan X (3072 cores, 12 GB ram, 250/300W). The installation of the GPU testbed in the MF prevents possible interference with the data taking of the experiment. To make the testbed able to run in the MF, some further configuration was needed, in particular we installed some specific packages needed to communicate with the monitoring infrastructure of the online environment.
Figure 2. CPU-GPU performance comparison using a Run1 simulated sample. (Left) Tracking efficiency as a function of the true track momentum. (Right) Tracking execution time and speedup versus number of events.

Figure 3. Scheme of the HLT and Monitoring farms. The data coming from the readout boards ("Tell1") are distributed via switches to all computing nodes of the farm.

3.3. Integration with the framework
One of the critical issues for the use of the new hardware is the integration with the software framework of LHCb ("Gaudi"). The Coprocessor Manager [4] is the framework (developed in LHCb) that enables Gaudi algorithms to exploit the power of massively parallel algorithm accelerators. It uses a client/server architecture, where a process called cpserver runs on each coprocessor-equipped machine, while multiple Gaudi instances on the same machine or on the same network connect to it as clients (figure 4). The cpserver process hosts all of the GPU algorithms. When a client sends data for processing, it specifies which algorithm is to be used for the task. As multiple clients send data to cpserver concurrently, it schedules them in a way that maintains high throughput. The server receives data from multiple concurrent clients, schedules algorithm execution, combines data into batches, runs algorithms, and distributes the results back to clients. To handle multiple concurrent clients, the server opens a socket and accepts each connection on a new thread. This is currently a local Unix socket, but an option to use a network socket is available.

In the testbed, client and server live in the same machine, the client being the HLT application.
sending data to the server and waiting for the GPU Velo tracks. After the Velo tracking on GPU, the tracks are sent back to the client which continues the standard HLT sequence on CPU.

![Coprocessor Manager architecture.](image)

**Figure 4.** Coprocessor Manager architecture.

### 4. Results

The testbed has been operational in the MF since 2016 and it allowed to collect both $pp$ and $p$-Lead collision events. The timing and physics performances of the “GPU-assisted” data-taking have been compared with the official reconstruction running on CPU. The comparison has been done by re-running the HLT sequence with the official CPU version of FastVelo on the same raw data and on the same machine. The timing performances has been measured as a function of the number of clients which ran in parallel on the testbed. The total elapsed time seen by a client, from sending data to the server to receiving back the output tracks, is roughly a factor 2 larger than the GPU tracking time alone (figure 5). This is due to the latency introduced by the data transfer (host-to-device, device-to-host copies) and the Coprocessor manager. The throughput of processed events vs the number of clients is also shown in figure 5: as expected, the GPU performance increases with the number of events but the number of available clients is not enough to fully exploit all the processing power of the GPU. More clients would be needed to get better performances and stress the system in a more realistic scenario.

Physics performances have been studied by comparing the signal and background yields for several types of particles reconstructed by the HLT monitoring (detached and prompt $D^0$, $J/\psi$, $\phi$). Yields have been extracted by fitting the invariant masses with a single Gaussian for the signal plus an exponential for the combinatorial background. Figures 6 and 7 show the fitted yields for detached $D^0$ and $J/\psi$ candidates for a 2016 sample of $pp$ events at the energy of 13 TeV (1M events). The number of signal candidates reconstructed by FastVelo GPU is $\approx 10\%$ lower than the yield obtained by CPU, while the signal to background ratio is slightly better for the GPU. In addition, the fitted mass resolutions are very close between CPU and GPU. A possible explanation of the discrepancy in the efficiency can be due to tighter cuts applied in the GPU algorithm to remove clone tracks; since these cuts have been tuned using Run1 MonteCarlo events at 7 TeV, some additional efficiency can be recovered by re-tuning the algorithm on a MonteCarlo sample generated with Run2 conditions.

1 The maximum number of clients which can be ran concurrently is limited to 24, the logical cores of the machine.
Figure 5. (Left) Execution time on GPU. (Right) Event rate in the case of FastVelo running on GPU (green line) and CPU (blue line). GPU rate includes latencies due to data transfer.

Figure 6. $D^0 \rightarrow K\pi$ invariant mass for detached $D^0$ (MeV/$c^2$). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.
Figure 7. $J/\psi$ invariant mass (MeV/$c^2$). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.

5. Conclusions and future plans

A first attempt to use accelerators in the HLT system of LHCb has been done during the Run2 using a parasitic GPU test-bed installed in the monitoring farm. The test-bed has been fully operational during 2016 and the physics performances have been found to be acceptable, even though some tuning of the GPU algorithm is still required. The throughput is currently limited by the number of clients communicating with the server, and more HLT nodes are required to obtain the best performances from the GPU and test the new system at higher rate.

There is an ongoing effort to add support for parallelism inside the Gaudi framework. The new framework is called GaudiHive and it uses the Intel Thread Building Blocks to divide work into tasks and its own scheduler to run them. GaudiHive could become a more efficient way of interfacing with the new hardware. In addition, a novel tracking algorithm based on cellular automata has been developed for GPU, its performances will be studied on the test-bed and applied to the upgraded detector in the near future.
6. References

[1] Barbosa-Marinho P R et al, *LHCb VELO: Technical Design Report*, CERN-LHCC-2001-011 (2001).

[2] Callot O, *FastVelo, a fast and efficient pattern recognition package for the Velo*, LHCb-PUB-2011-001 (2011)

[3] Nvidia C, *CUDA C programming guide* PG-02829-001 v6.0, February 2014

[4] Badalov A et al, *LHCb GPU Acceleration Project* LHCb-PUB-2015-021, October 2015