Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN FISICA

XXX CICLO

Coordinatore: Prof. Vincenzo Guidi

Study of feasibility of a GPU driven software for the LHCb High Level Trigger

Settore Scientifico Disciplinare FIS/01

Dottorando
Marco Corvo

Tutore
Prof. Eleonora Luppi

Anni 2013/2017

Abstract

Computing in High Enery Physics, as well as in other scientific contexts, has reached the limit in computational power provided by “serial” software, that is software which runs tasks sequentially. The concomitant improvement in luminosity and efficiency of the LHC, on one hand, and of the LHCb detector on the other, will require during Run3 a consistent speedup in every software layer, including the reconstruction one. Modern CPUs come with many distinct cores, autonomous processing entities which are able to run tasks concurrently, giving software applications the possibility to run faster.
In this thesis we will discuss the application of hardware devices, such as GPU cards, in the current LHCb trigger system. During Run2, a node equipped with a GPU has been inserted in the LHCb online monitoring system. During normal data taking, real events have been sent to the node and processed by GPU-based and CPU-based tracking algorithms. This gave us the unique opportunity to test the new hardware and the new algorithms in the real-time environment of the experiment. In the following sections, we will describe the algorithm developed for parallel architectures, the setup of the testbed and the results compared to the LHCb offcial reconstruction.

This work is organized as follows:

- Chapter 1 contains a brief description of the LHCb detector and of its subsystems, including the trigger which is the focus of these studies
- Chapter 2 contains the description of the motivations that lie beneath this work, along with some previous efforts from other HEP experiments. Some initial considerations on the underlying software and hardware needs are then described
- Chapter 3 gives a full description of the “LHCb GPU Acceleration Project”, its goals, implementation and peculiarities
- Chapter 4 shows the results of the tests performed both on simulated and real data during the normal LHC operations

- Chapter 5 finally is a wrap up of the project with some considerations that we could state following these experiences

Chapter 1

Introduction

1.1 LHC and the LHCb experiment

The Large Hadron Collider (LHC) [1] is the largest collider in the world, both because of its size and the energy of the accelerated particles. It was constructed by the European Organization for Nuclear Research (CERN) from 1998 to 2008, with the aim of testing the Standard Model and physics beyond it. Its physics program is heterogeneous and span many different studies like, among others, to search for the existence of the Higgs boson, to provide precise measurements of CP violation, to study the state of matter called quark-gluon plasma and to search for monopoles. In addition, with the pp collisions whose centre-of-mass energy can go up to 14 TeV, one has the opportunity to look for new physics, like Supersymmetry, dark matter and dark energy.

The LHC is built on the French and Swiss border, in a circular tunnel of 27 km in circumference and around 50 m to 175 m underground to avoid a high flux of cosmic rays and radiation hazard. It is designed to collide two counter-rotating beams of protons or heavy ions. Each beam consists of 2808 bunches (n_b), each bunch has $\approx 10^{11}$ protons (N_b) and the spacing between two consecutive bunches is 25 ns, giving an overall frequency of collisions of 40 MHz. Particles in the LHC are accelerated using radio-frequency (RF) cavities. 1232 dipole magnets are used to keep the beams on circular paths while 392 quadrupole magnets prevent the beams from spreading on the plane perpendicular to the beam line, with the aim of maximizing the number of collisions at the interaction points. The magnets are kept at an operating temperature of 1.9 K using roughly 96 tonnes of superfluid helium-4.

The nominal luminosity of LHC is given by:

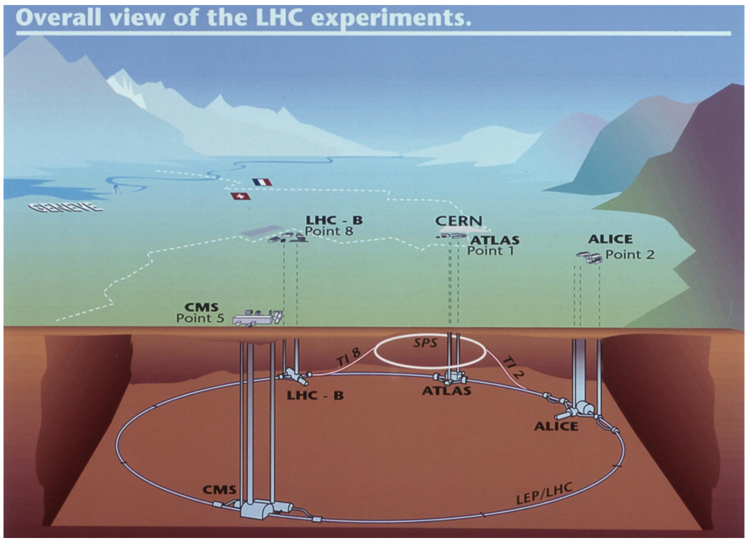


Figure 1.1: A sketch of the LHC tunnel with the four experiments

$$\mathcal{L} = \frac{N_b^2 n_b \nu}{F} \tag{1.1}$$

where ν is the frequency of revolution and F is a factor that groups the terms describing the beam geometry. The maximum instantaneous luminosity that the LHC can deliver is $\mathcal{L} = 2 \cdot 10^{34} cm^{-2} s^{-1}$.

The beams are brought together at four interaction points each one surrounded by one of the four main detectors. Two of these, ATLAS [2] and CMS [3], are multipurpose detectors designed to search for, among other aims, New Physics such as supersymmetric particles and to measure the Higgs boson. The other two detectors are more specialized. ALICE [4] primarily studies lead ion collisions while LHCb [5] was designed to carry out precise measurements in the heavy flavour sector, as briefly described in the following paragraph.

Figure 1.1 shows a sketch of the LHC tunnel along with the four experiments.

1.2 The LHCb Detector

The LHCb detector [6,7], whose side view is visible in figure 1.2, is a single-arm forward spectrometer covering the pseudorapidity range $2 < \eta < 5$, designed for the study of particles containing b or c quarks. It is composed of a magnet, a tracking system and a particle identification system. The tracking system includes the VErtex LOcator (VELO), two silicon trackers, the Tracker Tauricensis (TT) and the Inner Tracker (IT) and the Outer Trackers (OT).

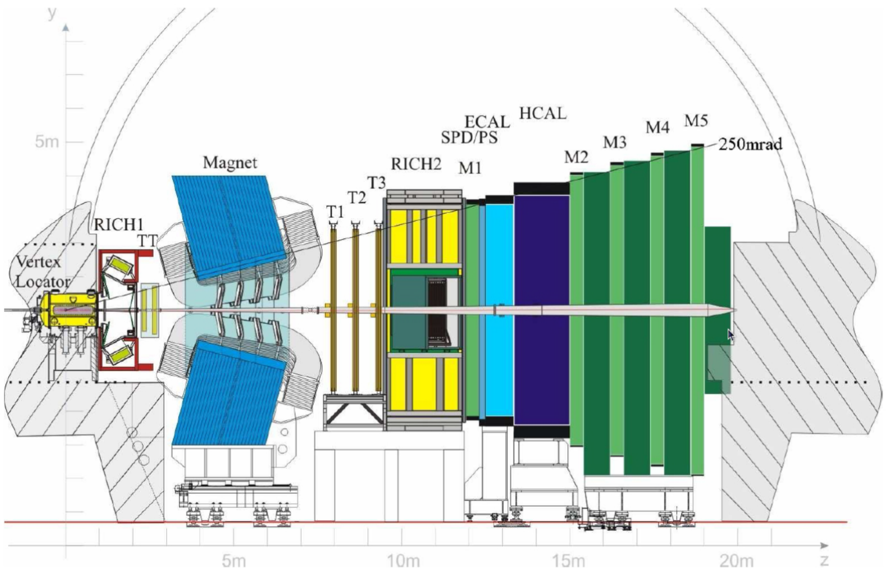


Figure 1.2: A side view of the LHCb detector

The particle identification system includes two Ring-imaging Cherenkov detectors (RICH), the calorimeters and the muon chambers. The different types of charged hadrons, like pions, kaons and protons, are distinguished using information from the two RICH detectors [8], while photons, electrons and again hadrons are identified by a calorimeter system [9] consisting of scintillating-pad and preshower detectors, an electromagnetic calorimeter and a hadronic calorimeter.

Muons are identified by a system composed of alternating layers of iron and multiwire proportional chambers [10].

1.2.1 Tracking System

The high-precision tracking system consists of a silicon-strip vertex detector surrounding the *pp* interaction region [11], a large-area silicon-strip detector [12] located upstream of a dipole magnet with a bending power of about 4 Tm, and three stations of silicon-strip detectors and straw drift tubes [13] placed downstream of the magnet.

The task of the tracking system is to reconstruct the particle trajectory whose curvature allows to deduce the momentum and the sign of the curvature with respect to the LHCb detector magnetic field, thus providing the charge of particle. The LHCb tracking system consists of three sub-detectors: the Vertex Locator (VELO) covering the collision point, the Tracker Turicensis

(TT) before the magnet and the three tracking stations T1-T3 behind the magnet. The VELO and TT are silicon pad and micro-strip detectors. In the T1-T3 stations, two different techniques are employed: silicon micro-strips for the region close to the beam pipe (Inner Tracker [14]) and straw-tubes for the outer region of the stations (Outer Tracker [15]) where the occupancy is less severe.

The tracking system provides a measurement of momentum, p, of charged particles with a relative uncertainty that varies from 0.5% at low momentum to 1.0% at 200 GeV/c. The minimum distance of a track to a primary vertex (PV), the impact parameter (IP), is measured with a resolution of $(15/p_{\mathrm{T}})\,\mu\mathrm{m}$ for particles with $p_T \approx 80\,\mathrm{GeV}/c$, where p_{T} is the component of the momentum transverse to the beam, in GeV/c.

1.2.2 The VELO sub-detector

The VErtex LOcator (VELO) is placed close to the beam pipe ($\sim$ 8 mm) and surrounds the collision point. It provides a precise measurement of the track coordinates in the region close to the interaction point in order to form the primary and decay vertexes. The resolution of the vertexes depends on the number of tracks. As an example, for a primary vertex of 25 tracks the resolutions are of 13, 12 and 69 μm in x, y, z dimensions respectively (this result is from 2011 data and for events that have only one reconstructed primary vertex). The VELO also allows to measure the deposited energies by the tracks in the silicon sensors that may be used to identify the particles [16]. The VELO is composed of 42 half-disc shaped modules that are spaced and perpendicular to the beam pipe over a length of $\sim$ 1 m. Upstream of the VELO sensors, there are two pile-up veto stations. Each VELO module has two silicon sensors: one measures the radial distance r and another one measures the azimuthal angle ϕ in the cylindrical geometry. Each silicon sensor is 300 μm thick and has 2048 strips which are read out by 16 Beetle FE chip, providing analogue data. The modules are divided into two halves (21 modules for each half) that are retractable during the beam injection to avoid high radiation damage to the sensors and to increase the aperture around the beam as required by LHC machine. These two halves will be moved to the closed position during data taking with the stable beams. Figure 1.3 shows the arrangement of the silicon sensors in the xz plane, and the relative positions of the sensors when the VELO is fully closed or opened.

To enable the sensors to be close to the beams ($\sim$ 8 mm), the VELO vessel

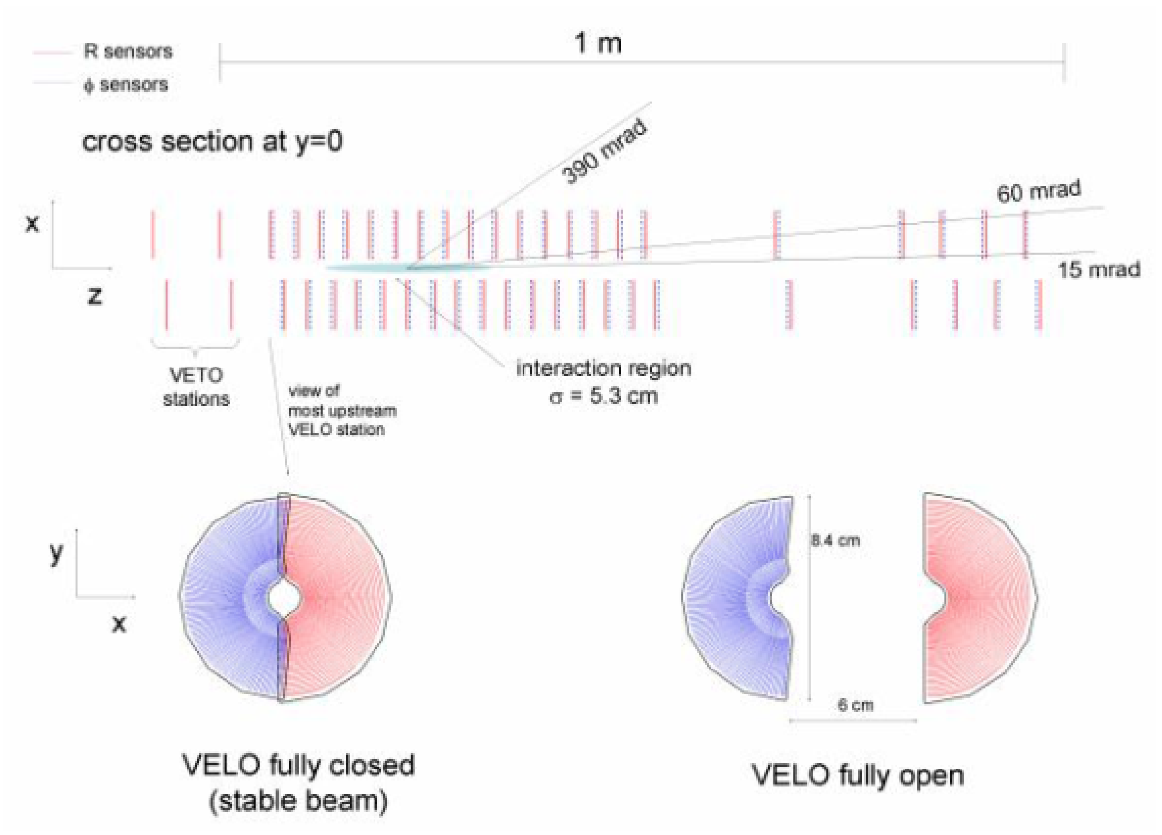


Figure 1.3: The VELO geometry

has to be integrated directly into the beam pipe. However, in order to prevent any accident from outgassing of the VELO modules which can jeopardize the LHC vacuum, the VELO vessel is maintained with a secondary vacuum which is separated from the machine vacuum by a thin wall of corrugated aluminium (RF foils). These RF foils form two boxes that enclose the VELO modules. The RF foils face to the beams are 0.3 mm thick, while the side walls of these boxes are 0.5 mm thick. A half of modules and a RF box are shown in figure 1.4 (left). The two halves of detector can overlap each other when they are in the closed position, as in figure 1.4 (right). These RF boxes are also intended to provide a shield against radio-frequency noises from the LHCb beams that affect to the VELO electronics.

1.2.3 Particle identification system - RICH

The RICH system has the task of identifying charged particles over the momentum range $1 - 150\,\mathrm{GeV}/c$, within an angular acceptance of $10 - 300\,\mathrm{mrad}$.

Particle identification is crucial to reduce background in selected final states

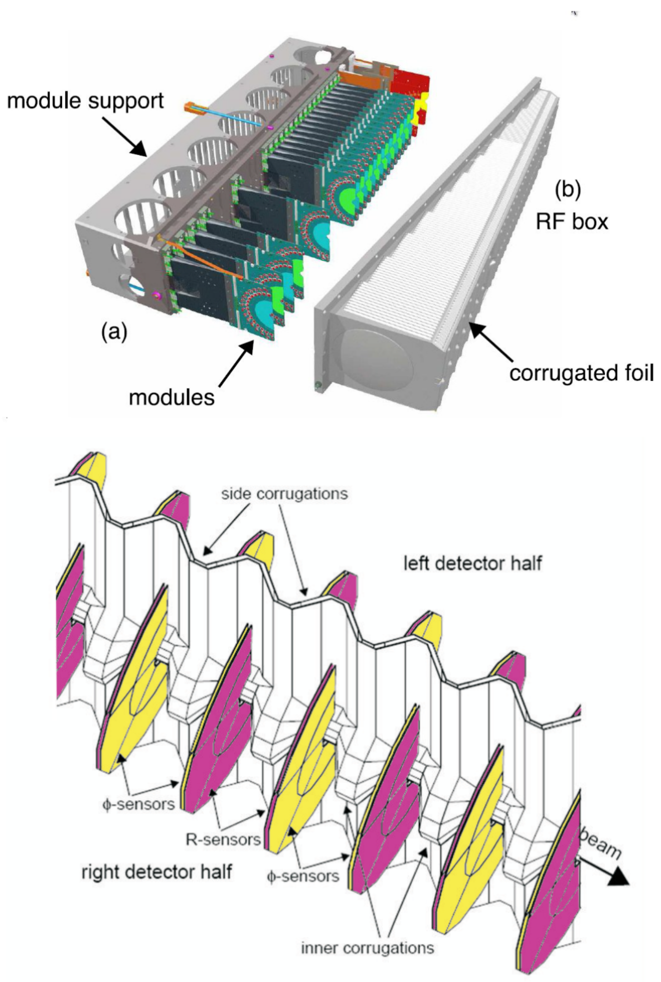


Figure 1.4: The VELO module

and to provide an efficient tag of the b-quark flavour using the kaon from the other b-quark produced in an event.

The LHCb RICH system consists of two distinct detectors: RICH 1 located upstream of the LHCb magnet and close to the interaction point and RICH 2 located downstream of the tracking system and before the calorimeter. Both detectors have gas radiators (C_4F_{10} and CF_4 respectively) while RICH 1 also used aerogel in Run I. The RICH detectors were re-optimised following Run I taking into account the new requirements for the event reconstruction and the need to run the same algorithms online and offline. The aerogel radiator was removed as its ability to provide particle ID for particles below the C_4F_{10} Cherenkov threshold for kaons is compromised by the total number of photons in RICH 1 in such a high track multiplicity environment. At the same time its removal allowed the full use of the gas radiator that is located between the RICH 1 entrance window and the aerogel (see figure 1.5. Removing it

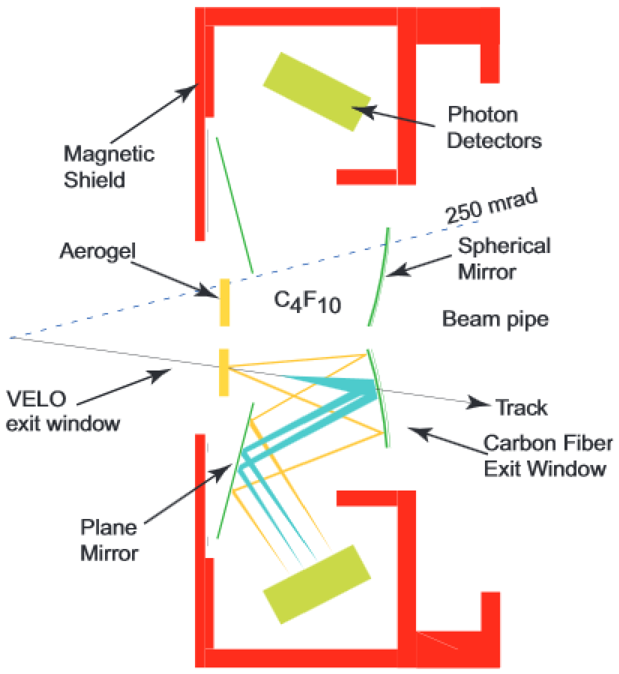


Figure 1.5: A section of the RICH 1 schematic showing the gas volume from the side. While the aerogel radiator was in place the Cherenkov photons generated in the shaded blue area filled with C_4F_{10} would be blocked. With the removal of the box these photons are free to reach the photon detectors.

also contributed significantly to the speed of the RICH reconstruction as it reduced by more than half the number of photon candidates (combinations of photon-detector hits with tracks) for which a Cherenkov angle is calculated [17].

Particles produced in the collisions in LHCb will travel through the mirrors of RICH 1 prior to reaching measurement components further downstream. To reduce the amount of scattering, RICH 1 uses special lightweight spherical mirrors constructed from a carbon-fibre reinforced polymer (CFRP), rather than glass. There are four of these mirrors, each made from two CFRP sheets moulded into a spherical surface with a radius of 2700 mm and separated by a reinforcing matrix of CFRP cylinders.

As RICH 2 is located downstream of the tracking system and magnet, glass could be used for its spherical mirrors, which in this case are composed of hexagonal elements.

Both RICH detectors use Hybrid Photon Detectors (HPDs) to measure the positions of the emitted Cherenkov photons. The HPD is a vacuum photon detector in which a photoelectron, released when an incident photon converts within a photocathode, is accelerated by a high voltage of typically 1020 kV

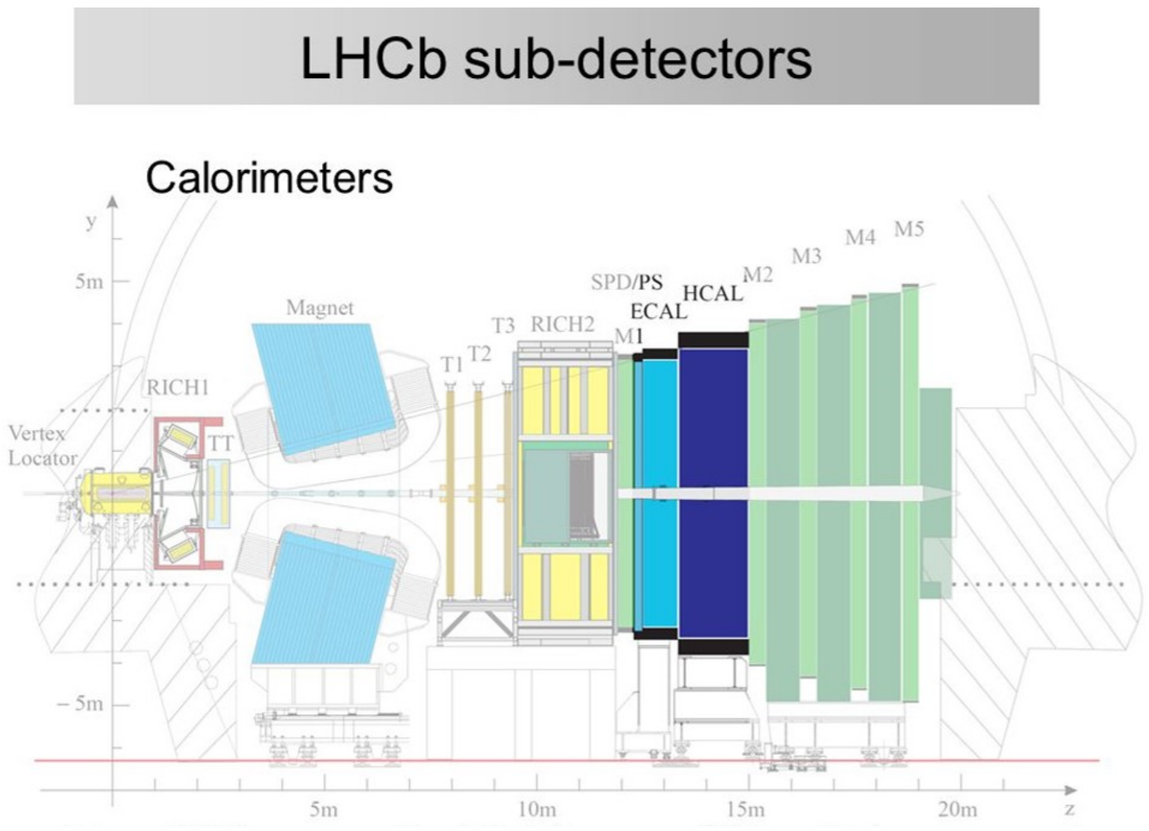


Figure 1.6: The LHCb detector with a highlight of the calorimeter system.

onto a reverse-biased silicon detector. The tube focuses the photoelectron electrostatically, with a demagnification factor of around five, onto a small silicon detector array.

1.2.4 Particle identification system - Calorimeters

The main purpose of the LHCb calorimeter system is to trigger on electrons, photons and hadrons [9]. It will also provide energy and position measurements of the particles produced in their angular acceptance, which are used in offline event analysis. Furthermore, the electromagnetic calorimeter will measure photons and neutral pions in association with the hadronic calorimeter. The calorimeter system (figure 1.6) consists of several layers: the Scintillating Pad Detector (SPD), the Pre-Shower Detector (PS), the Electromagnetic Calorimeter (ECAL), and the scintillating tile iron plate Hadron Calorimeter (HCAL).

The SPD determines whether particles hitting the calorimeter system are charged or neutral, while the PS indicates the electromagnetic character of the particle (i.e. whether it is an electron, if charged, or a photon, if neutral).

They are used at the trigger level in association with the ECAL to indicate the presence of electrons, photons, and neutral pions. The SPD and PS consist of scintillating pads with a thickness of 15 mm, interspaced with a lead converter with a characteristic radiation length X_0 of 2.5 $g \cdot cm^{-2}$. Light is collected using wavelength-shifting (WLS) fibers. Almost four turns of fiber are inserted and glued in the round groove made in the square pad, and both ends of the WLS fiber are used to transmit the light to multi-anode photomultipliers (MAPMTs) located at the periphery of the detector.

The ECAL employs "shashlik" technology of alternating scintillating tiles and lead plates. The cell size varies from $4 \times 4\,\mathrm{cm}$ in the inner part of the detector, to $6 \times 6\,\mathrm{cm}$ and $12 \times 12\,\mathrm{cm}$ in the middle and outer parts. The cell granularity corresponds to that of the SPD/PS, aiming at a combined use in γ/e separation. The overall detector dimensions are $7.76 \times 6.30\,\mathrm{m}$, covering an acceptance of $25\,\mathrm{mrad} < \theta_x < 300\,\mathrm{mrad}$ in the horizontal plane and $25\,\mathrm{mrad} < \theta_y < 250\,\mathrm{mrad}$ in the vertical. Light is detected by 10-stage photomultipliers.

The HCAL is positioned behind the ECAL. Its internal structure consists of thin iron plates interspaced with scintillating tiles arranged parallel to the beam pipe. Like ECAL, the inner and outer parts of the calorimeter have different cell dimensions ($13 \times 13\,\mathrm{cm}$ and $26 \times 26\,\mathrm{cm}$, respectively). The structure of the HCAL has the following features: in the lateral direction tiles are interspaced with 1 cm of iron (matching with the radiation length X_0); while in the longitudinal direction the length of tiles and iron spacers corresponds to the hadron interaction length λ in iron. Light is collected by wavelength-shifting fibers running along the detector towards the back side where the PMTs are housed. Three tiles arranged in depth are in optical contact with each 1.2 mm diameter Kuraray Y-11(250)MSJ fibre.

1.2.5 Particle identification system - Muon system

Muon triggering and offline muon identification are fundamental requirements of the LHCb experiment. Muons are present in the final states of many *CP*-sensitive B decays and play a major role in *CP* asymmetry and oscillation measurements, as muons from semi-leptonic b decays provide a tag of the initial state flavour of the accompanying neutral B mesons.

The muon system (figure 1.7) provides fast information for the high p_T muon trigger at the earliest level (L0) and muon identification for the High Hevel Trigger (HLT) and offline analysis.

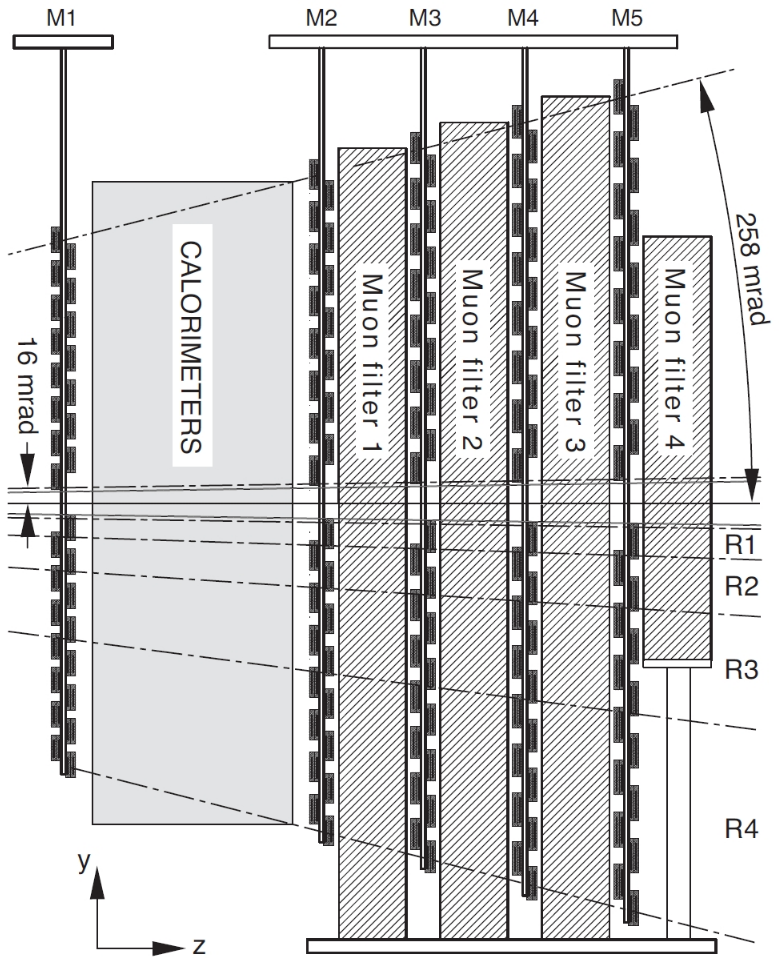


Figure 1.7: Sketch of the muon system sub-detector

The system is composed of five stations (M1-M5) of rectangular shape, covering an acceptance of ±300 mrad (horizontally) and ±250 mrad (vertically). M1 is placed in front of the scintillating pad detector/pre-shower. M2-M5 follow the hadron calorimeter (HCAL) and are separated by iron filters. The stations cover an area of 435 m^2.

Each station is divided into four regions, R1 to R4, with increasing distance from the beam axis. All the regions have approximately the same acceptance, and their granularity is shaped according to the particle density in that region in order to keep occupancy roughly constant over the detector. The granularity of the readout is higher in the horizontal plane, in order to give an accurate measurement of the track momentum and p_{T}.

Information must be gathered within 20 ns, so the detectors are optimized for speed. The system is therefore equipped with Multi Wire Proportional

Chambers (MWPC) with 2 mm wire spacing and a small gas gap (5 mm). Triple-GEM detectors are used in the innermost region (R1) of Station M1, where the rate is highest. This choice was dictated by the better ageing properties of this type of detector. There are 1380 chambers in the muon system, of 20 different sizes.

The detectors provide space point measurements of the tracks, giving a binary (yes/no) information. In order to provide this information, different readout methods are employed in the various parts of the detector: anode wire readout, cathode pad readout, or both. A total of 126'000 front-end readout channels are used. The electronics is based on custom radiation-hard chips developed for the muon system.

Chapter 2

The LHCb Trigger System

2.1 Introduction

The online event selection is performed by a trigger [18, 19] which consists of a hardware stage level zero (L0), based on information from the calorimeter and muon systems, followed by a software stage, in which, for data from 2012 on, all charged particles with $p_{\mathrm{T}} > 300\,\mathrm{MeV}$ are reconstructed.

The purpose of L0 is to reduce the LHC beam crossing rate of 40 MHz to the rate at which all sub-systems could be used for deriving a trigger decision. Due to their large mass b-hadrons decay to give a large E_{T} lepton, hadron or photon, hence L0 reconstructs:

- the highest E_{T} hadron, electron and photon clusters in the Calorimeter
- the two highest p_{T} muons in the Muon Chambers

whose information is collected by the L0 Decision Unit to select events.

Events can be rejected based on global event variables such as charged track multiplicities and the number of interactions, as reconstructed by the Pile-Up system, to assure that the selection is based on b signatures rather than large combinatorics, and that these events will not occupy a disproportional fraction of the data-flow bandwidth or available processing power in subsequent trigger levels. All L0 triggers are fully synchronous, i.e. their latency does not depend upon occupancy nor on history, and its electronics is implemented in full custom boards. The implementation of the calorimeter trigger is based on forming clusters by adding the E_{T} of 2×2 cells on the frontend boards, and selecting the clusters with the largest E_{T}. Clusters are identified as e, γ or hadron depending on the information from the Scintillating Pad Detector (SPD), Preshower (PS), Electromagnetic (ECAL) and Hadronic (HCAL) Calorimeter. The E_{T} of all

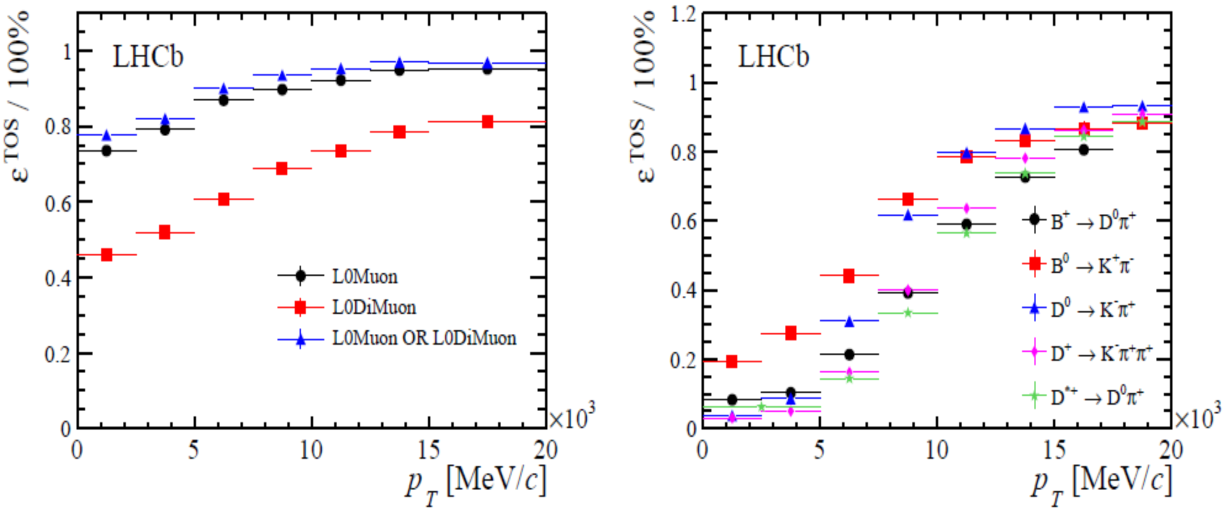


Figure 2.1: Trigger efficiencies for L0 muon (left) and hadron (right)

HCAL cells is summed to reject crossings without visible interactions. The total number of SPD cells with a hit are counted to provide a measure of the charged track multiplicity in the crossing. The muon chambers allow stand-alone muon reconstruction with a p_{T} resolution of 20%. Track finding is performed by processing units, which combine the strip and pad data from the five muon stations to form towers pointing towards the interaction region.

The Pile-Up system aims at distinguishing between crossings with single and multiple visible interactions. It uses four silicon sensors of the same type as those used in the VELO to measure the radial position of tracks, covering $4.2 < \eta < 2.9$. The Pile-Up system provides the position of the primary vertex candidates along the beam-line and a measure of the total backward charged track multiplicity. The Pile-Up information allows a relative luminosity measurement which is not affected by system deadtime, and monitors the beam conditions. The L0 Decision Unit (L0DU) collects all information from L0 components to form the L0 Trigger. The L0DU is able to perform simple arithmetic to combine all signatures into one decision per crossing. This decision is passed to the Readout Supervisor which transmits its Level-0 decision to the FrontEnd.

2.1.1 L0 Trigger and trigger efficiency

The bunch crossing frequency of the LHC is 40 MHz. This rate is reduced by the L0 trigger to a rate of 1 MHz, at which the full detector is read out. The L0 trigger is implemented in custom hardware and has a latency of 4 µs. It triggers on high p_{T} muons and on large E_{T} deposition in the calorimeters [20]. A relative momentum resolution of about 20% can be reached in the L0 muon reconstruction.

The trigger efficiencies are measured on offline selected events using some specific physics channels. For each channel the signal component, that is the set of tracks that forms the offline reconstructed and selected b- or c-hadron candidate, is determined by fitting the invariant mass distributions to avoid background contamination.

To determine the trigger efficiency, trigger objects are associated to signal tracks. The trigger records all the information needed for such an association. All measurements of the sub-detectors have a unique identifier, and these identifiers are written in a trigger report in the data stream for every trigger line that accepts an event. The criteria used to associate a trigger object with a signal track are as follows:

- An event is classified as TOS (Trigger on Signal) if the trigger objects that are associated with the signal are sufficient to trigger the event
- An event is classified as TIS (Trigger Independent of Signal) if it could have been triggered by those trigger objects that are not associated to the signal
- A number of events can be classified as TIS and TOS simultaneously ($N^{TIS\&TOS}$), which allows the extraction of the trigger efficiency relative to the offline reconstructed events from data alone

The efficiency to trigger an event independently of the signal, ϵ^{TIS}, is given by $\epsilon^{TIS} = \frac{N^{TIS\&TOS}}{N^{TOS}}$, where N^{TOS} is the number of events classified as TOS. The efficiency to trigger an event on the signal alone, TOS, is given by $\epsilon^{TOS} = \frac{N^{TIS\&TOS}}{N^{TIS}}$, where N^{TIS} is the number of events classified as TIS.
An example of L0 efficiencies are shown in figure 2.1. For low p_T tracks, the hadron efficiency is low as well and this represents an important limitation for analysis that rely on these class of events, as explained with more details later on.

2.1.2 High Level Trigger

The software trigger, named High Level Trigger or HLT [21], consists of software algorithms implemented in the same framework, named GAUDI, as the software used for the offline reconstruction. The HLT runs on an Event Filter Farm (EFF) of $\sim 52'000$ logical CPU cores ($\sim 29'000$ in Run 1) situated in the cavern close to the detector. Events selected by HLT are sent to the storage system for later offline analysis.

The HLT is divided in two levels (see figure 2.2 left). Each level works with a set of reconstruction algorithms which are grouped in so called *trigger lines*. A trigger line is composed of a sequence of reconstruction algorithms and selections. The trigger line returns an accept or reject decision.
In the first level, HLT 1, a partial event reconstruction is performed. HLT 1 reconstructs charged particle trajectories using information from the VELO and tracking stations. If at least one track is found that satisfies strict quality and transverse momentum criteria, then the event is passed to the second level of the software trigger (dimuon combinations may also trigger the first software level). HLT 1 thresholds can be tuned for an optimal output rate (In 2012 it was $\sim 150\,\text{kHz}$).

In the second level, HLT 2, the complete event is reconstructed. To satisfy the time constraints, the HLT event reconstruction in Run 1 was simpler than that used offline. Also it couldn't use the latest alignment and calibration constants that were calculated later and used in the subsequent offline reconstruction of the data.

An event will be accepted by L0, HLT 1 or HLT 2 if it is accepted by at least one of its trigger lines at the relevant stage. Combinations of trigger lines, together with a L0 configuration, form a unique trigger with its associated Trigger Configuration Key (TCK). The TCK is a 32 bit label pointing to a database that contains the parameters that configure the trigger lines. The TCK is stored for every event in the raw data, together with information on which trigger lines accepted the event.

During Run 1 running period, LHC delivered stable colliding beams for about 30% of the time. HLT operation was synchronous with the delivery of the particle collisions when the front-end boards send to the local disk of each HLT node massive amounts of data. These have to be quickly processed by the HLT 1 to reduce the amount of data sent to the slower HLT 2 level. To profit of the inter-fill inactivity of the farm, in 2012 $\sim 25\%$ of the L0-passed events were buffered to local disks and the HLT selection applied later to these events

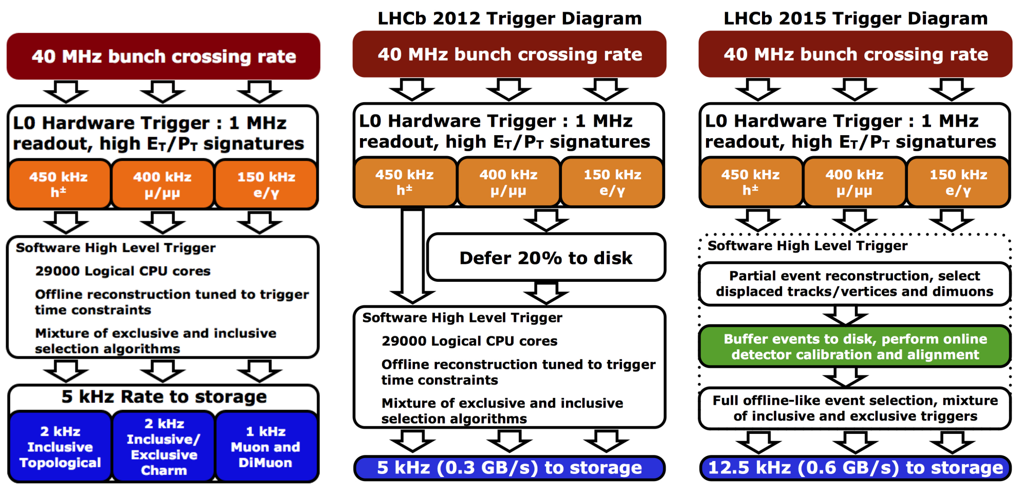

LHCb 2012 Trigger Diagram
LHCb 2015 Trigger Diagram
40 MHz bunch crossing rate
L0 Hardware Trigger : 1 MHz readout, high E_T/P_T signatures
450 kHz h±
400 kHz μ/μμ
150 kHz e/γ
Software High Level Trigger
29000 Logical CPU cores
Offline reconstruction tuned to trigger time constraints
Mixture of exclusive and inclusive selection algorithms
5 kHz Rate to storage
2 kHz Inclusive Topological
2 kHz Inclusive/ Exclusive Charm
1 kHz Muon and DiMuon
Defer 20% to disk
5 kHz (0.3 GB/s) to storage
Partial event reconstruction, select displaced tracks/vertices and dimuons
Buffer events to disk, perform online detector calibration and alignment
Full offline-like event selection, mixture of inclusive and exclusive triggers
12.5 kHz (0.6 GB/s) to storage

Figure 2.2: Three different trigger configuration in 2011 (left), 2012 (center) and 2015 (right)

(see figure 2.2, center). The result was a sizable improvement of $\sim 25\%$ of the HLT process rate. Nevertheless for large periods without beam, the HLT farm was still idle.

In Run 2, HLT 1 and HLT 2 become two independent asynchronous processes running on the same node. The implementation of this separation required a substantial change of the flow of event data in the experiment. All the events passing the HLT 1 selection are buffered in the local disk and later processed by HLT 2 (see figure 2.2, right). The allocated resources allow for a maximal execution time of $\sim 35\,\mathrm{ms}/event$ in HLT 1 and of $\sim 650\,\mathrm{ms}/event$ in HLT 2. Since events not accepted by HLT 1 are lost for later offline physics analysis, while in collision the HLT 1 processes run with high priority. At the same time HLT 2 runs on each node sending the finally accepted events to the long term storage, however during data taking it has a lower priority to not negatively interfere with the HLT 1 processing.

The separation of the two HLT levels allows to execute the high quality alignment and calibration before running the HLT 2 on the events selected by HLT 1. Using the best calibration constants, allow to have offline quality reconstruction already in the trigger. Thanks also to the additional computing resources and the optimized code, in HLT 1 tracks are reconstructed down to $p_{\mathrm{T}} = 500\,\mathrm{MeV}$. Main features of HLT 1 are single and two tracks MVA algorithms which provide inclusive charm and beauty selection with improved performance with respect to Run 1 (e.g. the 2-body charm trigger is factor ~ 2 more efficient), inclusive single and dimuon triggers, and exclusive lines for the “lifetime unbiased” beauty and charm selection (selections don’t cut on quantities correlated with the signal particles decay-time).

HLT 2 does the full event reconstruction starting from the information (vertices and tracks) calculated in HLT 1. It reconstructs all tracks, while in Run 1 a $p_{\mathrm{T}} > 300\,\mathrm{MeV}$ cut was applied upfront, and builds the full particle identification information for long tracks. Reconstruction is followed by physics selections, including MVA lines based on 2-, 3-, and 4-body detached vertices to inclusively select beauty and charm decays, exclusive beauty decays, like $B \to \phi\phi$, $B \to \gamma\gamma$, charmed barions and electroweak bosons.

A further improvement on the global HLT quality comes from the management of the storage disks. During the first year of Run 2, the storage of the HLT farm was 5 PB. As a figure of merit, this allows for 160 hours of data taking with an HLT 1 output rate of $\sim 150\,\mathrm{kHz}(60\,\mathrm{kbytes}/event)$. For redundancy, in 2015 the whole storage was mirrored. From the 2012 experience of disk failure, the risk of data loss due to the unrecoverable errors has been evaluated to be $\sim 0.1\%$. The loss expected without mirroring the HLT disks, has to be compared with a potential $\sim 15\%$ increase of triggered charm physics or with an improved quality of the online reconstruction. For this starting from 2016 data taking, the HLT disks are unmirrored and the whole 10 PB is used in the HLT running. The data loss rate is monitored and agrees with the expectations

2.2 Trigger limits

The trigger system is central for the matters covered in this thesis, as its hardware level will be a considerable bottleneck when the LHC luminosity will increase after the long shutdown in 2018.

The LHCb experiment is in fact 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 main tracking detectors [22] and the RICH [23] 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. One of the main limitations of the current system is indeed the hardware trigger which limits the input rate to the HLT to 1 MHz. This initial reduction causes the largest inefficiencies, especially for purely hadronic decays. This effect is shown in figure 2.3. The L0 is a good trigger for muonic signatures because there is very little punch-through and hadronic misidentification. For hadrons, on the other hand, the background

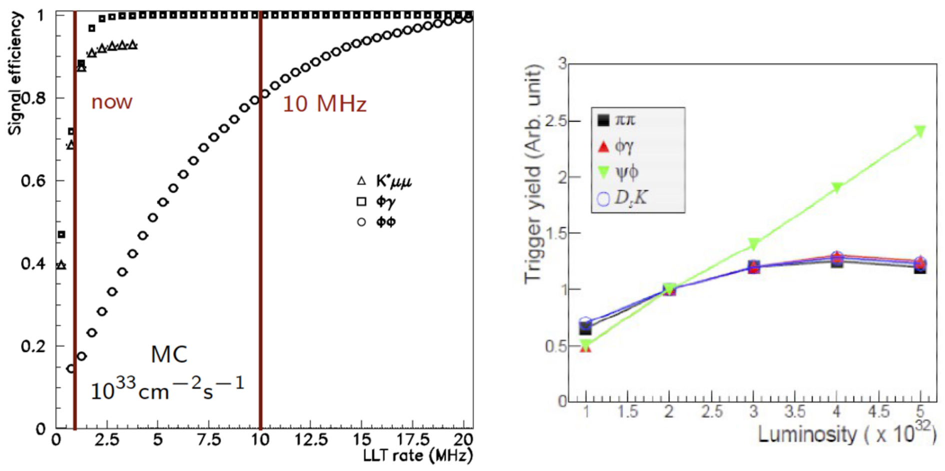


Figure 2.3: The bottleneck of the hardware trigger. For hadronic channels the yield is limited by the available bandwidth

from lightquark QCD production in the *pp* collision is inherently much higher, so the efficiency is lower (see figure 2.1).
Without the L0 hardware level, reading the whole detector at full *pp* collision rate will allow to increase the number of TOS hadron by loosing the thresholds, thus improving the efficiency.

The enhanced luminosity implies signal rates of about 300 kHz of beauty hadrons and 800 kHz for charm hadrons that are reconstructible inside the detector [24] which, if compared with the current rate of nearly 500 kHz of the L0 hadron, is nearly twice its capacity. This represents a fundamental change in trigger requirements of LHCb. It is no longer the case that the trigger must reject background from signal, it must now categorize signal according to physics requirements. Such a strategy requires significantly more information in the trigger than can be provided by low-latency, hardware based solutions.

With the removal of the L0 hardware trigger, LHCb will be the first hadron collider experiment to deploy a trigger exclusively in software, using off-the-shelf hardware. The upgraded LHCb trigger will select and categorise events at the full collision rate. The advantages of such a system are clear. Software is easily modified, allowing an unprecedented flexibility as and when the LHCb physics programme changes. It also has the advantage that computing power is readily upgradeable and benefits from the ability to purchase more CPU power for the same price at a later stage. The deferral technique used in Run 1 and Run 2 will be leveraged also in Run 3. Subdetector alignment and calibration will be performed while events are buffered to disks, permitting the use of offline-quality reconstruction, and reducing the need for reprocessing of data at a later stage.

Therefore, one of the main goal of the LHCb upgrade is to remove this bottleneck by implementing a full software trigger able to process the full collision rate. Given the available resources for the upgrade, this translates into a time budget for the HLT of $\sim 13\,\mathrm{ms}/event$ to take a decision on wheather to accept or reject a given event [25]. It should be noted that this time budget is considerably lower than the current time budget of the HLT 2 which amounts to $\sim 650\,\mathrm{ms}$. Furthermore the 13 ms limit has been calculated taking into account a foreseen growth in computing power of a factor 1.364 per year up to 2020, while the actual growth is only 1.093, as measured according to the observed growth in the throughput, in Hz, of a typical box for the HLT farm.

This discrepancy leads to a missing factor of ~ 6 (see e.g. [26]) in computing power for the upgrade phase, according to this formula:

$$\frac{30\,\mathrm{MHz}}{1.0934^4 \cdot 3364\,\mathrm{Hz} \cdot 1000 nodes} = 6.2 \tag{2.1}$$

where 3364 Hz is the current event throughput of a machine equipped with an *Intel*® Xeon E5-2630 v4 and the current budget for the online farm allows to buy only 1000 nodes.
This gap must be tackled to guarantee the physics performances of the detector and the project described in this thesis goes in this precise direction.

Chapter 3

The quest for speed-up

3.1 Introduction

As discussed in Chapter 2, the increase in luminosity will limit the efficiency of the trigger especially in hadronic channels, thus forcing LHCb to get rid of the bottleneck represented by the hardware level. This means that the whole detector will be read at an unprecedented rate of 30 MHz putting under big strain the HLT and the online facility. The LHCb collaboration started to investigate innovative software solutions to improve the efficiency of the algorithms which make up the HLT, bearing in mind that this is a problem of High Throughput Computing. In fact the goal of the collaboration is to improve the number of events that are fully reconstructed by the HLT in the unit time. To achieve it, the problem must be tackled on two sides: on the first one, a new hardware solution which is able to digest more events per second must be found, on the other one a software solution which is able to process more events concurrently.

The combination of these two lines of research led the LHCb collaboration to start an internal project called "LHCb GPU acceleration project" [27] aiming at studying the potentiality of GPUs applied to HEP software.

3.2 Motivations and background

Other physics experiments have investigated the idea of moving some of their computation to hardware accelerators.

The SuperB experiment [28] was one of the first HEP experiments who started to investigate the possibilities provided by new CPU architectures. At the early stages of its development, people in SuperB started a re-engineering effort on the software framework of the BaBar experiment, introducing the concept of data consumer and producer algorithms, in order to detect potentially independent data processing functions which could be performed in parallel [29]. The idea behind these studies was to factorize sets of algorithms which were not sharing the same chunks of data and execute them concurrently with a concurrent tool by *Intel*® named *Threading Building Blocks*. The prototype developed by the collaboration aimed at being then exploited with hardware accelerators like the *Intel*® Xeon Phi and GPU.

Researchers at the NA62 experiment have investigated the problem of adapting specific algorithms to massively parallel architectures [30]. Using *Nvidia*® CUDA GPU programming platform [31], they make the important observation that the overhead cost of setting up the computation is high compared to the cost of processing a single particle collision event, thus making very desirable to process events in batches, amortizing the setup cost.

Researchers at ALICE (A Large Ion Collider Experiment) at LHC took the next step and investigated the issues that arise from GPU algorithm integration with existing software infrastructure [32]. They have successfully added GPU track reconstruction to ALICE s HLT and the offline reconstruction framework. This turned out to require adjustments in the HLT framework, as well as AliRoot, ALICE offline software framework for data analysis, event reconstruction, and simulation. Because CUDA is only available on a fraction of cluster nodes, all of the GPU code had to be packaged in a separate library and loaded dynamically as needed. The authors also encountered a mismatch in the threading models used by the HLT framework and CUDA. The mismatch was that HLT processes events over multiple asynchronous concurrent threads, while at the time CUDA was not thread-safe and restricted to single-threaded use. It should be noted that later versions of CUDA allow access from multiple threads.

ATLAS (A Toroidal LHC Apparatus) has mounted a serious effort to integrate GPU computation with its software infrastructure [33]. ATLAS has a special relationship with LHCb as the two experiments share the same software framework called GAUDI. ATLAS's implementation of GAUDI is called Athena and spawns multiple concurrent processes for multiple events, using a technique known as COW, that is Copy on Write, where each new process shares the same memory page frames. ATLAS initial design was to make remote procedure calls to the GPU. In this design, a separate GPU Server process hosts a number of GPU kernels. Athena processes connect to this process, choose a kernel, send parameters to the kernel, and wait to receive the results. The server accepts requests and calls whichever kernel is requested in the order it receives the requests. This approach is in practice very similar to ALICE method of packaging CUDA code into dynamically loaded libraries; in both cases GPU kernel calls are made available to regular Athena algorithms as external procedures. Having a separate server offers cleaner separation between regular Athena pipeline code and CUDA dependent algorithms; it also has the added benefit of allowing remote calls to CUDA kernels to be made across a network. Finally, hosting all the kernels in a separate long-lived process could potentially save on the costs of GPU device setup. There are significant costs to hosting GPU code in a separate service this way. Every kernel has custom parameters and returns custom data types. Adding new kernels also requires somewhat complicated encoding and decoding changes to the server request processing loop and to the Athena client. This design also necessitates expensively copying potentially large amounts of data between different processes for sequences of kernel calls when it would be preferable to simply keep this data in GPU memory. A related issue is that the design makes it difficult to reuse resources when calling the same kernel for different events.

ALICE and NA62 experience with GPU shows the difficulty in maintaining high utilization of the processing units. Detectors like LHCb, where individual events require relatively little computation, need massively parallel event processing systems to combine and simultaneously process multiple events. This difficulty will increase as long as the growth of GPU performance outpaces growth in information volume produced by particle detectors. Another issue that has to be addressed is the system ease of use by the algorithm developer. If it happens to be difficult to transfer an algorithms implementation from a standalone version to the detector production system, our choice of massively parallel algorithms may be restricted.

3.3 The Gaudi Software framework

The studies shown in this thesis move from the assumption that, to solve the LHCb trigger revolution, the collaboration must tackle the problem in a twofold manner: the first one is improving the software performances and the second is exploiting hardware which can run this software faster.
To contextualize the problem, we start describing the software environment which lies beneath every modern HEP experiment and which relies, generally, on commodity hardware to run.

The simulation and analisys software rest in fact on a framework, which is re-

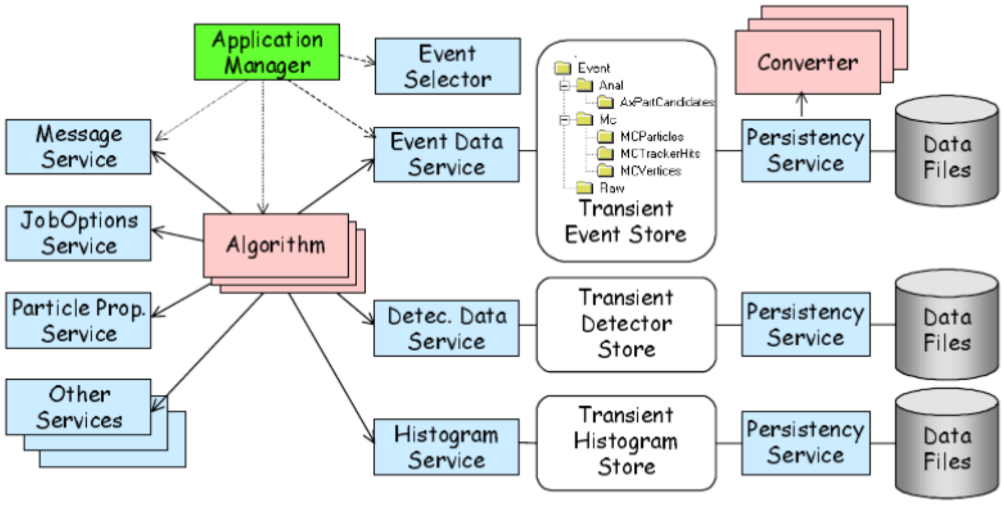


Figure 3.1: Architecture of the GAUDI framework along with its main components

sponsible for many different tasks, e.g. reading the data from the main memory, read the geometry of the detector and the calibration and aligmnent constants, sort and orchestrate the various algorithms to perform the computations in the right order, write back the results to files that can be afterwards read by the user for further analysis etc. Such framework for the LHCb experiment is named GAUDI [34], an object-oriented software, implemented in C++ and Python, that provides a common infrastructure and environment for building HEP event data processing applications. GAUDI is designed on the principles of composability, providing a way to construct applications of any complexity by combining general-purpose and specialized components. The main components of the GAUDI software architecture can be seen in the object diagram shown in figure 3.1. All applications, based on GAUDI, are written in the form of specializations of standard framework components, complying to a well defined set of abstract interfaces. Strict interfacing rules of the GAUDI components were a key to a highly flexible framework.

An important underlying principle of GAUDI is the separation of the concepts of data and procedures used for its processing. Broadly speaking, event reconstruction and physics analysis consist of manipulation of mathematical or physical quantities (such as points, vectors, matrices, hits, momenta, etc.) by algorithms which are generally implemented in terms of equations and natural language. In the context of the framework, such procedures are realized as GAUDI Algorithms. The GAUDI architecture foresees either explicit unconditional invocation of algorithms by the framework, or by other algorithms. This latter possibility is very important as it allows to address a common use case of

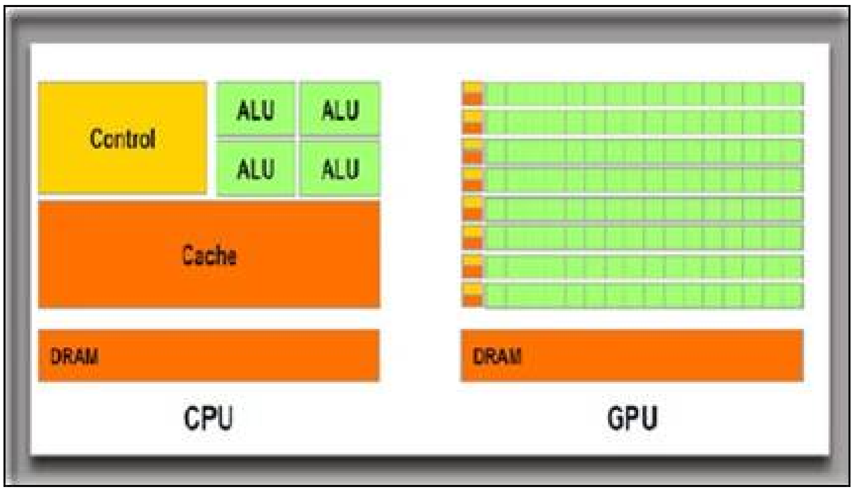


Figure 3.2: A CPU compared to a GPU. The former has only few ALUs, whilst the latter has many

a physics application to execute different algorithms depending on the physics signature of each event, which might be determined at run time as a result of some reconstruction. This capability is supported in GAUDI through sequences, branches and filters. A sequence is a list of Algorithms. Each Algorithm may make a filter decision, based on some characteristics of the event, which can either allow or bypass processing of the downstream Algorithms in the sequence. The filter decision may also cause a branch whereby a different downstream sequence of Algorithms will be executed for events that pass the filter decision relative to those that fail it.

GAUDI, whose architecture and concept date back to the late '90, was originally designed as a sequential framework. It means that it is only able to processes events sequentially, one after the other. Furthermore, it means that the sequence hierarchies, as well as all Algorithms within a sequence, are also executed sequentially for each event. Lastly, all GAUDI components including Algorithms were designed and implemented for single-threaded execution only.

We will see that this kind of paradigm used to fit the architecture of commercial CPUs up to the early XXI century, when Intel and AMD released their first dual-core processors [35,36]. In fact commodity CPUs up to this period were designed to execute instructions sequentially, and so were the software running on these architectures. With the multi- and many-core era, software unable to exploit all the processing units concurrently is no longer effective and looses performances and scalability.
In the following we will review how hardware changed and how LHCb software

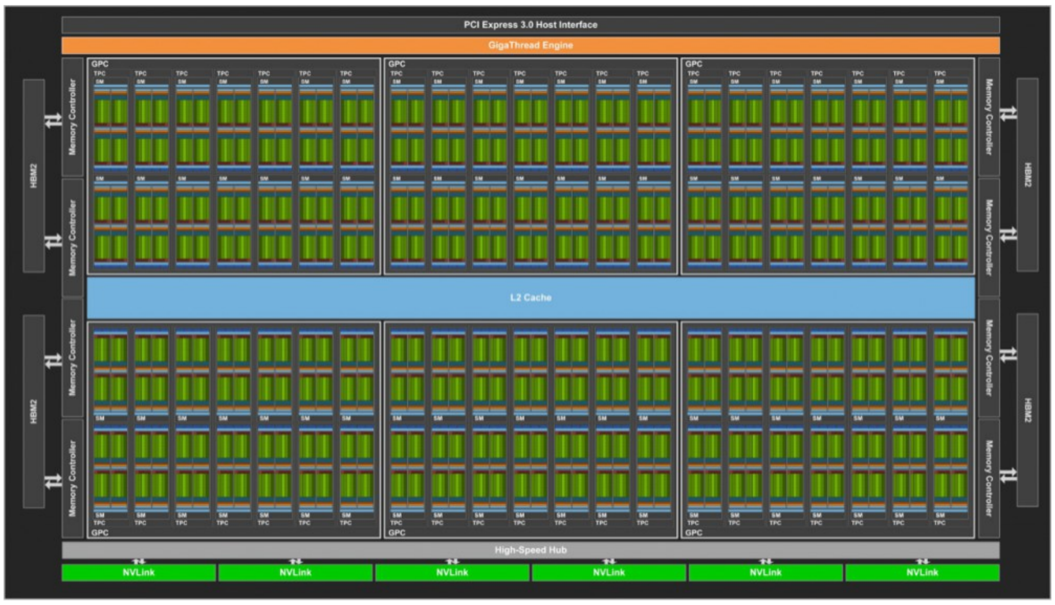


Figure 3.3: A modern GPU from Nvidia. It contains 5120 Cuda core FP32 and 2560 Cuda core FP64

started to change accordingly to adapt to these new architectural transformations.

3.4 Concurrency in hardware and software for HEP

In the past, the increasing demands for HEP processing resources could be fulfilled by the ever increasing clock-frequencies and by distributing the work to more and more physical machines. Two different aspects then showed up and curbed this paradigm and led to exploit concurrency. The first one is a physical limitation which sets an edge on the dimension of transistors and consequently their number on a single die. The second is again a physical limitation on the frequency driven by power consumption of both CPUs and entire data centers. These brought to an end this era of easy scalability.

To overcome these restrictions, hardware producers started to build devices which are characterized by thinner, more specialized and more numerous cores per die, and rather heterogeneous resources.

Early attempts to exploit a sort of concurrency in computing date back to the mid '80s when designers begun to use a technique named "instruction pipelining", a kind of instruction level parallelism. In this architecture, the

processor works on multiple instructions in different stages of completion. For example, the processor can retrieve the operands for the next instruction while calculating the result of the current one, saving clock cycles and thus computing time. A similar idea, introduced only a few years later, was to execute multiple instructions in parallel on separate arithmetic and logic units (ALUs). Instead of operating on only one instruction at a time, the CPU looks for several similar instructions that do not depend on each other, and execute them in parallel. This approach is called superscalar processor design.
A conceptually different approach is the base of modern Graphic Processing Units. In these devices the system takes a single instruction and executes it on many different chunks of data, performing what is called a SIMD operation, that is *Single Instruction on Multiple Data*. These devices are also know as vector processors, because they operate on vectors of data. An example of this kind of modern device is shown in figure 3.3. At the same time, it became clear that software supposed to run on those devices, had to evolve to fully exploit the potential of the many cores and undertake a major conceptual redesign in its architecture: move to a native parallelization of the code. HEP data processing frameworks needed to evolve too to allow for parallelization which can be achieved at three different levels:

- parallel processing of multiple events at the same time
- parallel execution of algorithms within a single event
- parallelization within the algorithms themselves

The first level is the easiest to fulfill and is in fact known as *embarrassing parallelism* [37]. In its conjugation, the procedure is to spawn as many processes as there are events to process, for example exploiting Grid or Cloud Computing facilities spread all over the world. This mechanism proved to be efficient during the first years of data taking of the LHC experiments.

The second level is more subtle and expects a pure multithreading approach, where every algorithm is mapped to a task, which we could identify with a thread of execution, that is executed according to specific scheduling rules. These rules are driven by priorities in the algorithm sequences [38]. As an example we can consider priorities driven by data readiness, where the system gives higher priority to algorithms whose data are ready to be processed.

The third level is even more difficult to fulfill, as it requires a deep re-engineering effort to modify the data structures inside the code to exploit the modern AVX (Advanced Vector eXtensions) or SSE (Streaming SIMD

Extensions) registers and implement a pure SIMD paradigm. The technique behind this model is the *vectorization* of data, that is the ability to organize data structures in vectors whose elements can be processed in parallel. The LHCb collaboration is moving also in this direction, for example in the Kalman fit for particle tracking [39].

3.4.1 The LHCb way for a concurrent framework

To leverage the full extent of hardware parallelism suggested by the industry, the LHCb collaboration developed an initial prototype of a concurrent GAUDI, called "Gaudi Hive" [40]. The prototype aimed at demonstrating, in the context of GAUDI, the basic principles of the multithreading paradigm, one of the fundamental paradigm for high-throughput and many-task computing.

An important evolution in the GAUDI architecture to support and facilitate parallelism is the task-based model programming. The main advantage of formulating computations in terms of tasks, and not threads, is that they favour a well organized work partitioning. The task-based approach also allows concentrating on dependencies between tasks, leaving load-balancing issues to a back-end scheduler. Within the Gaudi Hive project, the *Intel*® Threading Building Blocks (TBB) library was chosen as such back-end [41].

Implementing this evolution in the GAUDI framework required its substantial revision. However, thanks to the principle of composability described earlier, it's been possible to avoid any fundamental architectural change that would require a substantial delay in its development. The components necessary for the first minimalistic prototype of the multithreaded GAUDI were developed and used to augment the framework via its standard interfaces in a pluggable fashion [42]. The architectural peculiarities of the GAUDI Hive prototype are outlined in figure 3.4 and best described in [38].

At the time when the studies shown in this thesis were performed, this prototype was not fully exploitable. As a consequence, the usage of the standard software framework proved to be determinant for the limitations in performances that the studies revealed. Nevertheless this constraint showed also, in a unequivocal way, that the evolution of the framework towards a parallel implementation is critical for the goals of the physics program of the experiment.

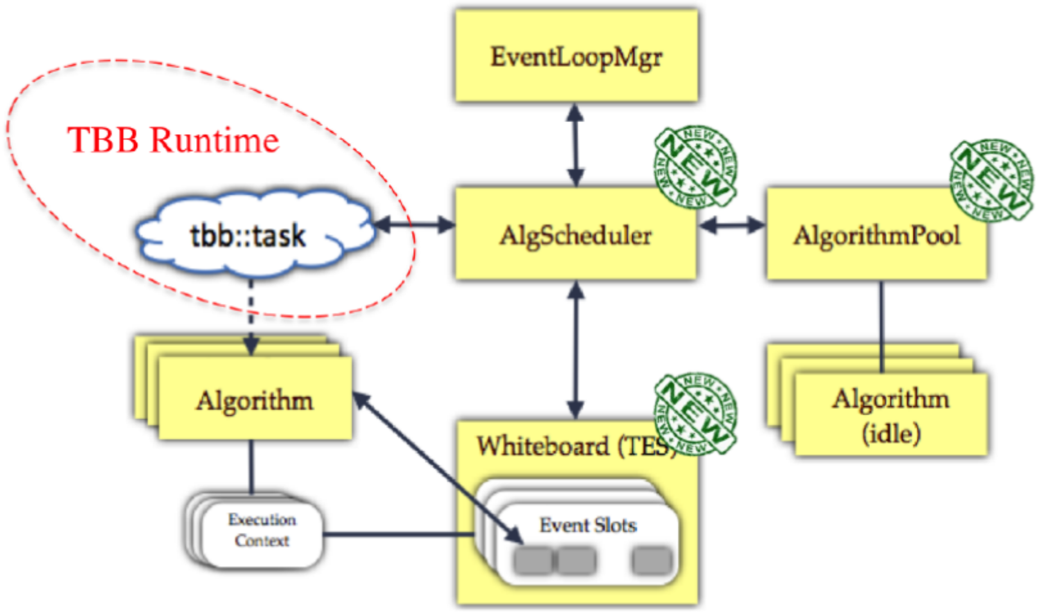


Figure 3.4: The evolution of the GAUDI framework towards a multithreading architecture

3.5 LHCb choice for exploiting concurrency in hardware

It has been shown that parallelism in both hardware and software can be achieved exploiting different approaches. In hardware the LHCb collaboration spotted a promising device to boost the throughput of events through its HLT in the GPGPUs.
These objects push to the edge the concept of the SIMD paradigm in computing, in that they contain a huge, compared to a standard CPU, number of ALU processors (see figure 3.2) which perform the same computation, or *Instruction*, on many different chunks of data. LHCb deemed that this kind of approach fits the intrinsic structure of a HEP experiment, where many different, mutually independent, events can be processed in parallel applying the same algorithms, or instructions, thus accomplishing the first level of parallelism. This is true even inside the event, where charged particles fly towards the different subdetectors leaving “hits” that can be viewed as points forming a straight line. These lines are then reconstructed to determine the parameters of the particles’ tracks. These tracks are independent one from the others, permitting the exploitation of the second level of parallelism.

There are nevertheless some considerations to take into account when

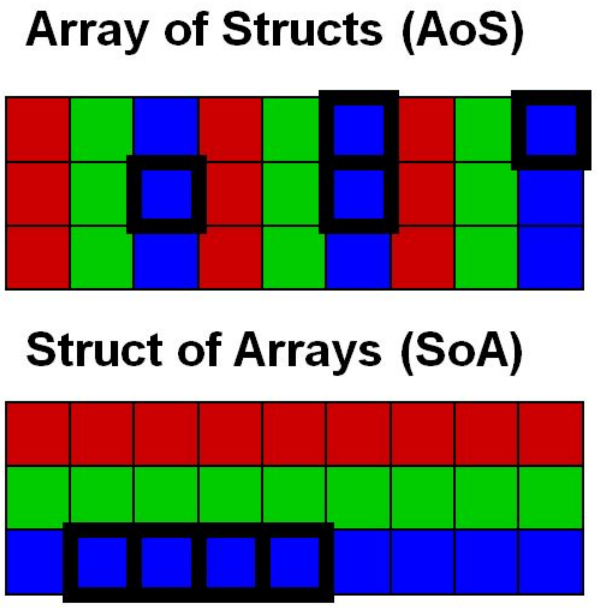


Figure 3.5: AoS vs SoA. Structures of Arrays accomplish what is called *Memory coalescence*, that is data belonging to the same class (e.g. the x component of vectors or the radius r of circles) are stored in contiguous memory locations

approaching these accelerators. The first one is that they "talk" their own programming language and whoever is responsible for a specific physics algorithm, must learn it to leverage to a full extent their computing power. For *Nvidia*® devices, the language is almost a standard ANSI C, with some directives for managing, basically, memory allocation, data offloading and processing.
The second one is that the common "serial" programming patterns are not very useful and a big effort should be put to reorganize data structures into what is called SOA, that is *Structure Of Arrays*, instead of more widespread AOS, that is *Array of Structure*. The technical reason that lies beneath this change in paradigm, is that SIMD computation works more efficiently on regular well-formed vectors to optimally exploit the internal registers (see figure 3.5). This mechanism goes under the name of *Memory Coalescence* because it favours gluing together data of the same class, e.g. the x component of a set of vectors, or the radius r of a set of circles, by storing them in contiguous memory locations. These are then loaded and processed in a single clock cycle.

There are further considerations to take into account and that belong to a more complex scenario where software design hits the hardware architecture of these devices. To make a concrete example, *Nvidia*® GPUs process data towards a bunch of concurrent threads called *warp*. Each *warp* accounts for 32 threads which execute the same operation, or instruction. Consider this simple code:

```
if (x<0.0)
  z = x-2.0;
else
  z = sqrt(x);
```

This kind of “decision making” code is called a *branch*. If data are partitioned so that even only one element takes a negative value, the system must behave differently, that is perform different operations, on the set of 32 threads. This behaviour is called *warp divergence* and can lead to very poor performances due to the extra cost of fetching and performing different operations.

During the time spent to accomplish these studies and measurements, all these issues have been faced, although at different levels and depth according to the priorities and goals of the project, and contributed to build the final considerations on the feasibility of a full software trigger using graphic accelerators.

Chapter 4

The LHCb GPU project

4.1 Goal and work plan of the project

To prove the feasibility of the project, it was decided to spot an algorithm which were both representative of the difficulties in gaining the required scalability with the foreseen LHC luminosity and simple enough to be translated into a CUDA code without too much effort in development: the latter has an intrinsic value to show that the effort to port an existing code to the new language can be kept to a minimum, so that the learning curve of physicists acquiring the new skills can be kept steep.

The *FastVelo* algorithm [43] was deemed suitable for the purposes of the project, as it's been conceived to reconstruct tracks inside the VELO subdetector. The algorithm itself is simpler than other tracking ones as the VELO is not affected by the magnetic field, which bends particles in other tracking stations, thus making the math more complex. A preparatory activity started to analyze the code, to understand it and find the right way to factorize it for the subsequent translation into CUDA code.

Then it was necessary to spot the right environment where to put a GPU-equipped node so to collect real data from the LHCb detector. At teh beginning the idea was to arrange it in the online farm, where the whole HLT lies. This looked the most natural place where to put the node, as it was precisely the place where to test the goodness, affordability and effectiveness of the GPU project. At the end, and after some discussions with the people responsible for the online facility, the choice fell on the Monitoring Farm (see 4.2.1 later on).

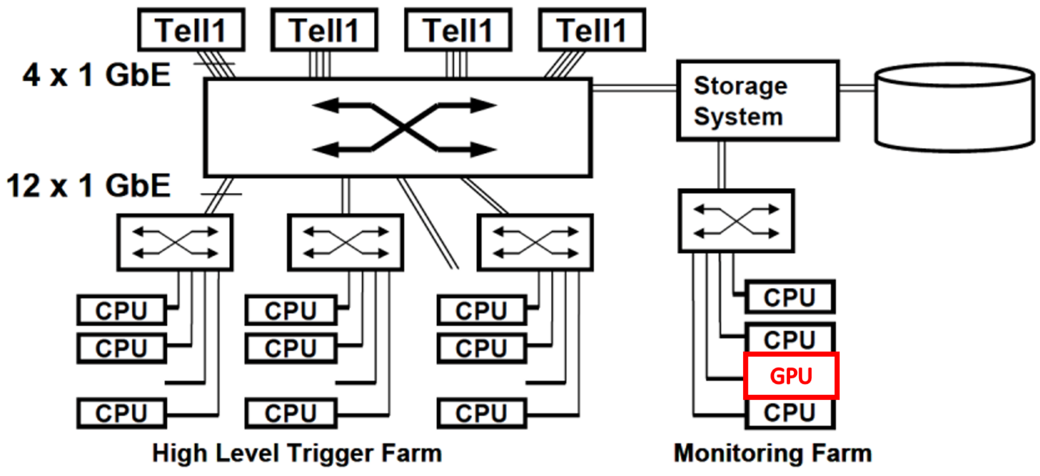


Figure 4.1: The architecture of the HLT farm along with the machine equipped with a GPU that made up the testbed

4.2 Testbed architecture

To perform some measurements directly on data coming from the detector, it has been decided to install a "parasitic" machine in the Monitoring Farm (MF) of LHCb. The MF has a different and less critical role than the Online Farm, which performs the complete reconstruction of the events passing the L0 trigger. In fact its main task is to process a small bunch of events to extract physics quantities, plotted with histograms, which are then used for a real time validation of the incoming data.

This choice permitted on one hand to get real data from *pp* collisions, and on the other to avoid perturbing the regular operations of the online facility. The average rate of events feeding the monitoring nodes is $\mathcal{O}$ (10 Hz). The overall scheme of the HLT and MF is shown in figure 4.1

4.2.1 Hardware and its integration with the online environment of LHCb

The testbed hardware installed in the MF consisted initially of a standard desktop PC equipped with an *Intel*® *Core*™ i7-4790 CPU @3.60 GHz with hyper-threading, while in a second phase a typical HLT server, whose characteristics are shown in table 5.2, has been used. The monitoring farm works at a sustained rate of few tens of Hz, which on one hand is good enough for our testing purposes and on the other hand keeps the system safe. To make the testbed able to run in the MF, a dedicated configuration was needed, in particular some specific packages had to be installed to communicate with the

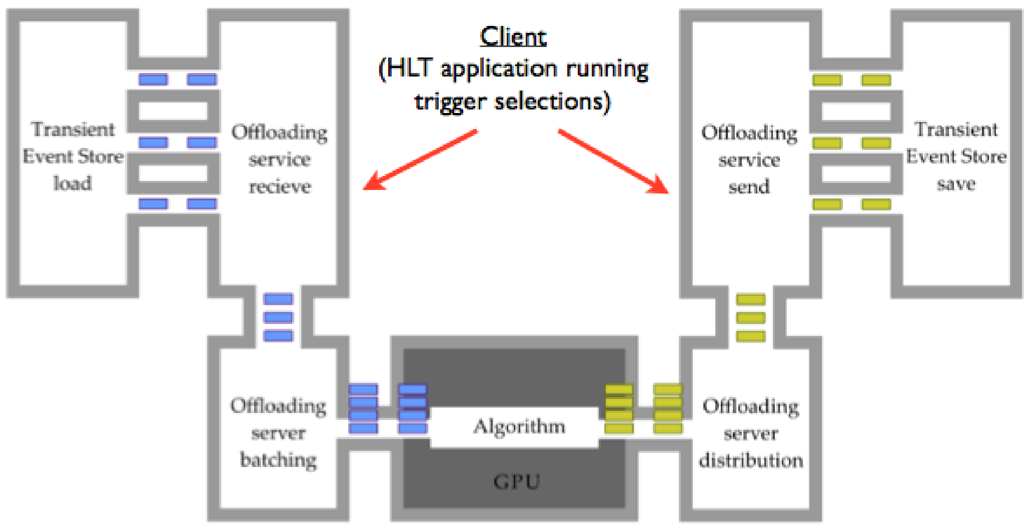

Client
(HLT application running trigger selections)
Transient Event Store load
Offloading service recieve
Offloading service send
Transient Event Store save
Offloading server batching
Algorithm
GPU
Offloading server distribution

Figure 4.2: The architecture of Coprocessor Manager. Each client sends data to the *cpserver* who then sends them back when processed by the accelerator.

monitoring infrastructure of the online environment.

4.2.2 Software integration with the Gaudi framework

As mentioned already in 3.2 one of the critical efforts to accomplish to achieve the goals of the project is to integrate the GPU algorithm with the software framework of the experiment.
The solution adopted by LHCb is similar to the one adopted by ATLAS [44] In fact the collaboration developed the Coprocessor Manager [27], a tool that enables GAUDI algorithms to be run on generic accelerators. It uses a client/server architecture, where a process called *cpserver* runs on each accelerator-equipped machine and listens to connection either on local sockets or on network ones, while multiple GAUDI instances on the same machine or on the same network connect to it as clients.

The server receives data from multiple concurrent clients, schedules algorithm execution, combines data into batches, runs algorithms, and distributes the results back to clients (see figure 4.2). To handle multiple concurrent clients, the server opens a socket and accepts each connection on a new thread. When a client connects, the server receives a data payload and the name of the algorithm to be used to process it. The payload goes into a queue for scheduling and the thread is suspended. Once the payload is processed, the client thread is woken up, and the result of the computation is sent back to the client. The client gets notified when the requested algorithm is not available

or an exception is thrown during the computation. Algorithm scheduling and execution are done on a dedicated thread. This has the benefit that the *cpserver* is not stalled while data is passed back and forth between the server and its clients.
Algorithms are executed one after another whenever data is available. The scheduling algorithm is a variant of the first-come-first-serve (FCFS) approach: the difference is that whenever a payload is removed from the queue, all payloads targeting the same algorithm are removed from the queue with it. In fact these payloads are submitted to the GPU as a batch, that is all the payloads which needs to be processed by the same algorithm are grouped and sent together to the accelerator.
This has the advantage, in the particular configuration that was used to perform the studies and was spotted during our tests, that the server doesn't need an explicit buffering mechanism which would increase the overall processing time adding a latency overhead to fill up the buffer. In principle one could expect that this overhead should be balanced and overcome by a higher throughput because of a higher occupancy of the cores of the GPU, given that more events are feeding it. This overhead, though, is not compensated by this higher throughput and this is (partially) in contrast with the idea of keeping the GPU cores as much busy as possible, queuing as many events as there are cores available. The reason is that the measurements made were performed using a machine which could afford a maximum of 24 (then 32 in the standard HLT machine configuration) client instances, while a real benefit would come with $\mathcal{O}$ (100) clients which can fill up all the GPU cores.

When the hardware cannot cope with the amount of incoming data, new clients are refused connection. FCFS, despite its simplicity, has an important benefit: because there is no prioritization, every payload is guaranteed to be processed in time. Prioritization schemes risk "starving" low-priority clients. However, more complicated approaches may be preferable, if latency guarantees are desired.
A specific GAUDI service, called *CpService*, has been developed to allow GAUDI algorithms to submit data to *cpserver*.

As a further limitation, explained before in 3.4, when these studies were performed, the concurrent GAUDI was still under development and not all the functionality, from which the project could benefit, were operational. Yet the results show that the need for a genuine parallel framework is crucial to exploit thoroughly the architectures of modern CPU and accelerators.

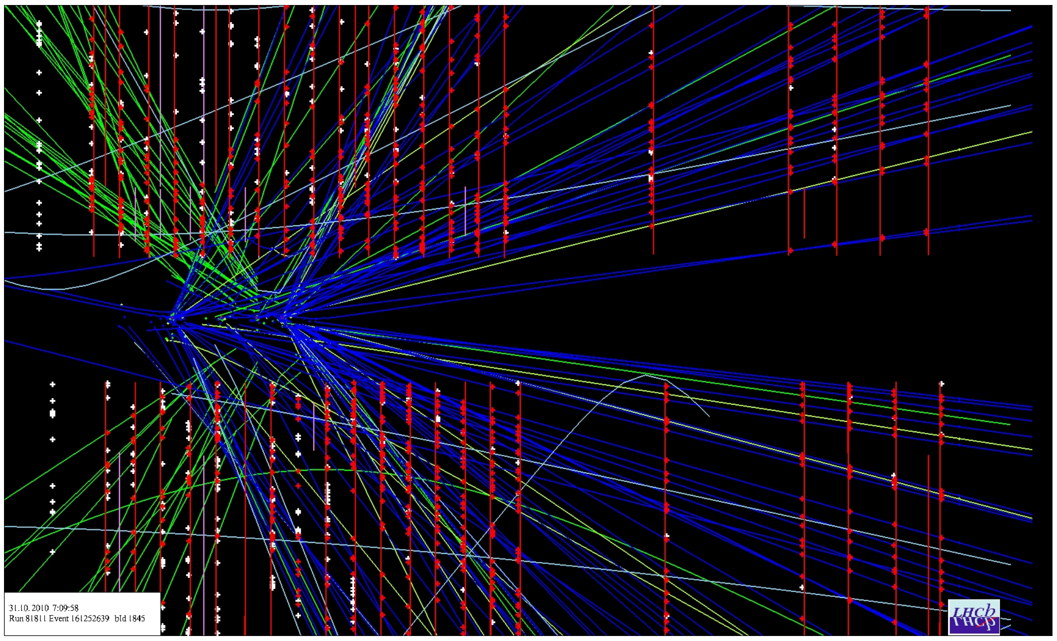


Figure 4.3: The RZ projection of particles' tracks inside the VELO detector. The algorithm looks for "quadruplets" or "triplets" to seed new tracks

4.2.3 Description of the algorithm

As a brief reminder, the VELO [11] 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. It is formed by 21 stations, each consisting of two halves of silicon-strip sensors, which measure R and ϕ coordinates. A sketch of the Velo detector is shown in Figure 1.3.

FastVelo [43] 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.

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")[1]. 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,

[1]R and ϕ hits are sorted at an early stage of data processing.

the full tracks are built by adding the information of the ϕ hits to the RZ track. A first processing step is to define the first and last ϕ sensor to use, then the algorithm starts from the first station with hits and searches for a triplet of nearby ϕ hits. The triplet is then added to the RZ track to form a 3D tracklet, so that the track parameters can be estimated. These 3D segments are then extrapolated towards the interaction region by adding hits in the next stations compatible with the tracklet. The tracks are then re-fitted using the information of the R and ϕ hits, while the hits with the worst χ^2 are removed from the track. As last step, the hits of the track are marked as used and not further considered for the following iterations ("hit tagging"): this is done to reduce the number of clones produced by the algorithm avoiding encountering the same track several times.

4.2.4 GPU implementation

The strategy used for porting FastVelo to GPU architectures ("FastVeloGpu"[2]) takes advantage of the small size of the LHCb events ($\approx$ 60kB per event, $\approx$ 100 kB after the upgrade) implementing two level of parallelization: "of the algorithm" and "on the events". With many events running concurrently, it can be possible, in principle, to gain more in terms of time performance with respect to the only parallelization of the algorithm. The CPU algorithm was adapted to run on GPU using the NVIDIA Compute Unified Device Architecture (CUDA) language [31].

One of the main problems found in the parallelization of FastVelo concerns the hit tagging which brakes data independence among different concurrent threads of execution. The removal of the hit tagging is then necessary to make all threads independent and to minimize the number of atomic operations. However, the main drawback of this choice is that the number of clone and ghost tracks becomes too large 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. It must be noted that the original algorithm running on HLT 2 includes additional parts for searching R-hit triplets and unused ϕ hits. However, the GPU implementation of FastVelo reported in this note refers only to the VELO tracking running on HLT 1 during the RUN 1. Here's a brief description of how the algorithm works.

The algorithm searches for long tracks first, using only the last five sensors downstream the VELO (upstream for backward tracks). Four threads (one for

[2]The code is available on Git: `https://gitlab.cern.ch/gianelle/FastVelo`

each sensor zone) find all possible quadruplets in these sensors. Each quadruplet is then extended independently as much as possible by adding R-hits of other sensors. The R-hits of each RZ track are marked as used; potential race conditions are not an issue in this case, because the aim is to flag an hit as used for the next algorithms. Next, the remaining sensors are processed: each thread works on a set of five contiguous R sensors and find all quadruplets on a zone of these sensors. A check is done on the hits in order to avoid hits already used for the long tracks. In a sense, the algorithm gives more priority to the long tracks with respect to the short ones. At this stage the number of quadruplets belonging to the same tracks is huge and a first "clone killer" algorithm is needed to protect against finding the same track several times. All pairs of quadruplets are checked in parallel: each thread of the clone killer algorithm takes a quadruplet and computes the number of hits in common with the others; if two quadruplets have more than two hits in common, the one with worst χ^2 is discarded (here, the χ^2 is defined as the sum of residual of the position of the R-hits of the RZ track with respect to the predicted position given by fitted track). Each quadruplet is again extended independently as much as possible by adding R-hits of other sensors on both directions. After this step, all possible RZ tracks are built. The number of clones generated by the algorithm is still huge, and another clone killer algorithm similar to the one implemented in the previous step is used to reduce the fraction of clone tracks to a tolerable value. In order to detect clones, a check is made for all possible track pairs: if two tracks shares more than 70% of their R-hits, the shortest track, or the one with worst χ^2, is discarded.

It should be noted that this procedure of cleaning clones follows the same lines of the one implemented in the original FastVelo algorithm ("mergeClones"), the only difference being that in FastVelo the clone killer algorithm is applied only to the full 3D tracks (almost at the end of the tracking procedure), while in the parallel implementation, without hit tagging, we are forced to introduce it well before in the tracking sequence in order to reduce the number of tracks in input to the next steps.

Next step is to perform full 3D tracking by adding ϕ hits information. Each RZ track is processed concurrently by assigning a space-tracking algorithm to each thread. This part is almost a translation in the CUDA programming language of the original space-tracking algorithm, with the notable exception of the removal of tag on the used ϕ hits. A minor modification with respect the original code is that in the parallel version the handling of RZ tracks in the sensors overlap regions was simplified to avoid recursive calls present in the sequential algorithm.

When all 3D tracks have been found, a final cleanup is done on the tracks to kill the remaining clones and ghosts; the clone killer algorithm is the same of the one used in the previous steps, with the exception that now the χ^2 is based on the information of both R and ϕ hits of the track. Finally the survived tracks are converted in the *LhcbTrack*s format and are ready to continue the standard workflow for the complete event reconstruction.

The following chapters describe results and performances on both simulated and real data, collected during two distinct period, one with pp and one with heavy ions collisions.

Chapter 5

Tests, performances and results

5.1 Introduction

This chapter includes the results obtained in two distinct stages of the project: during the first stage, tests were performed only on simulated data and with a standalone version of the algorithm. Standalone is meant to be a version of the FastVelo which run originally as an isolated Cuda executable and only after a preliminary period of testing, as an algorithm integrated in the GAUDI framework.
The idea behind this project is first of all to prove the feasibility of porting an existing algorithm to the Cuda language and to assess its physics performances, in order to certify that the results are exactly the same as those gained with the standard CPU. So the focus of the tests is twofold: on one hand the improvement of the processing time and on the other to match the current physics performances.

The second stage of the tests has been performed using the parasitic testbed and collecting data from real proton-proton and proton-lead collisions. We started taking data with our system in October 30^{th} 2015. The first attempts were useful to set-up and tune the hardware and software components. Several adjustments had to be put in place to heal unexpected behaviour of the whole chain.
Since the software environment of the online and monitoring farm is different from that of the offline, a specific setup had to be put in place in order to get data from the detector and inject them in the trigger sequence. In fact the system needed a particular software switch to route the events either to the CPU, for the standard track reconstruction, or to the GPU.
To get this result, we developed two different *trigger lines*. These are a sort

of wrapper for each algorithm in the trigger sequence: their basic aim is to setup the algorithm, perform some sanity checks on incoming data and then fire the algorithm for the data processing. Every trigger line in the sequence has only one computing target, that is the standard CPU. In our case, we had to develop a separate line with the GPU as a computing target. These two different lines could be switched in order to route data either to the CPU or the GPU algorithm, permitting the comparison of the two software solutions.

One of the main issues during this stage was a hidden clash in the name of the two algorithms which made the software crash when one of the two trigger lines where fired. The second main issue was the tuning of the data structures inside the GPU algorithm, in order to find the right compromise between the memory needed by the process and the dimension of the bunch of events which were processed. The testbed has been operational again with the new server machine in the autumn of 2016 and it allowed to collect both *pp* and *p*-Lead collision events.

5.2 Physics performances on simulated events

Some preliminary measurements have been made on simulated events. In order to obtain a measure of the correctness of a reconstruction, several metrics exist. These performance indicators are used when the reconstructed particles are known, like Montecarlo simulated events.

Reconstruction efficiency is defined as

$$\epsilon = \frac{|F \cap R|}{|R|} \tag{5.1}$$

where F is the set of reconstructed particle, and R the set of reconstructible particles, that is, the set of particles expected to be reconstructed.

The purity of a track is defined as

$$purity = \frac{N_{correct}}{N_{total}} \tag{5.2}$$

where $N_{correct}$ is the number of hits left by the particle that produced the track and N_{total} is the total number of hits associated to that specific track.

Out of all reconstructed particles, some may come from measurements which are produced by noise effects, like multiple scattering on silicon detectors. Others may simply come from a bad selection of hits. Particles reconstructed in this way are referred to as ghosts, since they do not represent real particles. The *ghost fraction* of tracks in a given event is defined as

$$ghost = \frac{N_{ghost}}{N_{total}} \tag{5.3}$$

where N_{ghost} is defined as the set of tracks whose purity is $< 70\%$.

A reconstructed particle is made up by several hits. These data point may be picked up separately to reconstruct several particle tracks, instead of a single one. Particles reconstructed in this way are called *clones*.

The *clone fraction* is then defined as

$$clones = \frac{N_{clones}}{N_{tracks}} \tag{5.4}$$

A track is considered a clone of another one if the number of hits in common is $> 70\%$.

For the first set of measurements two simulated samples have been used to evaluate the tracking and timing performances: $B_s \rightarrow \phi\phi$ events generated with 2012 conditions witha pile-up of $\nu = 2.5$ and a b-inclusive $B_s \rightarrow K^*\mu\mu$ sample produced with an upgraded scenario for 2015 ($\nu = 4.8$). Timing performances have been compared also with real data using a NoBias sample collected during 2012 ($\mu = 1.6$). While ν is the number of total elastic and inelastic proton-proton interactions per bunch crossing, μ represents the number of visible interactions per bunch-crossing. LHCb labels simulated event samples according to ν and real data according to μ. In the Montecarlo sample, the average number of hits per sensor is ~ 17, while the average number of reconstructed VELO tracks per event is ~ 80. The machine adopted for these measurements was an *Intel*® *Core*™ i7-3770@3.4 GHz.

The comparison of tracking efficiencies between the GPU implementation and the original FastVelo algorithm for different categories of tracks is shown in table 5.1. 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 5.1 shows the tracking efficiency as a function of the true track momentum P_{true} and the resolution of the impact parameter as a function of $1/P_{T,true}$ 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 shown in Figures 5.2, 5.3 and 5.4 for the three datasets. The timing performance of the GPU tracking algorithm has been measured first in a standalone way (i.e. outside the LHCb software framework GAUDI) by feeding the GPU with a

Track category	FastVelo on GPU		FastVelo	
	Efficiency	Clones	Efficiency	Clones
VELO, all long	86.6%	0.2%	88.8%	0.5%
VELO, long, p > 5 GeV	89.5%	0.1%	91.5%	0.4%
VELO, all long B daughters	87.2%	0.1%	89.4%	0.7%
VELO, long B daughters, p > 5 GeV	89.3%	0.1%	91.8%	0.6%
VELO, ghosts	7.8%		7.3%	

Table 5.1: Tracking efficiencies obtained with FastVelo on GPU, compared with the results obtained by original FastVelo code. The efficiencies are computed using 1000 $B_s \to \phi\phi$ MC events, generated with 2012 conditions.

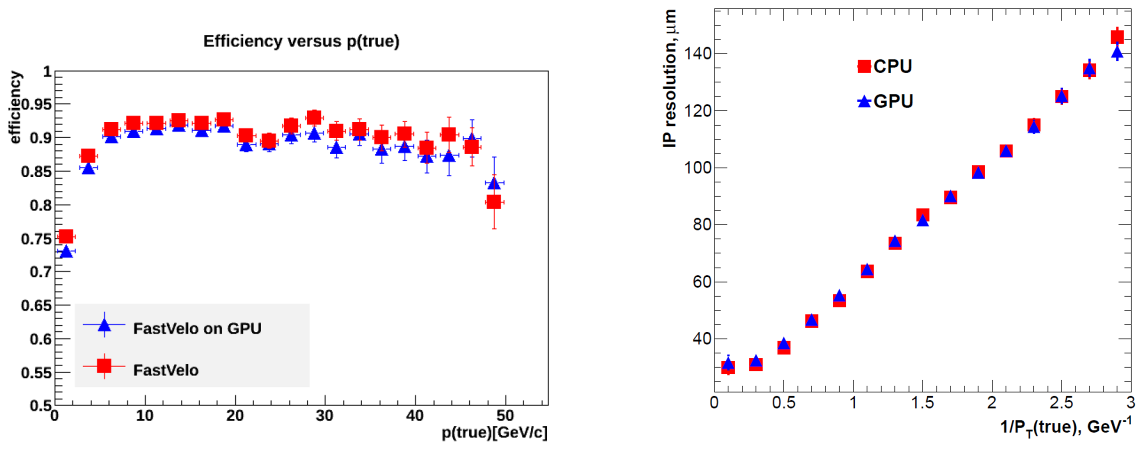


Figure 5.1: Tracking performance comparisons between the sequential FastVelo and FastVelo on GPU. (Left) Tracking efficiency as a function of the true track momentum P_{true}. (Right) Impact parameter resolution as a function of $1/P_{T,true}$

.

fixed amount of events, while the CPU tracking time has been taken from the detailed FastVelo profile given by the LHCb reconstruction software Brunel. The GPU time has been measured using the standard CUDA timer. In these plots, only the GPU and CPU tracking time has been considered (excluding data transfer from host to GPU memory and vice-versa).

The GPU algorithm behaves differently according to the occupancy of hits in the VELO (the average occupancy in the 2015 Montecarlo sample is a factor of two higher than 2012 samples). The maximum speed-up obtained by the GPU algorithm with respect to the sequential FastVelo is $\approx 3x$ for the 2012 datasets, while it decreases to $\approx 2x$ for the 2015 sample. The speedup as a function of the number of events can be explained by the fact that the occupancy of the GPU cores increases with the number of events (level 1 and 2 parallelism).

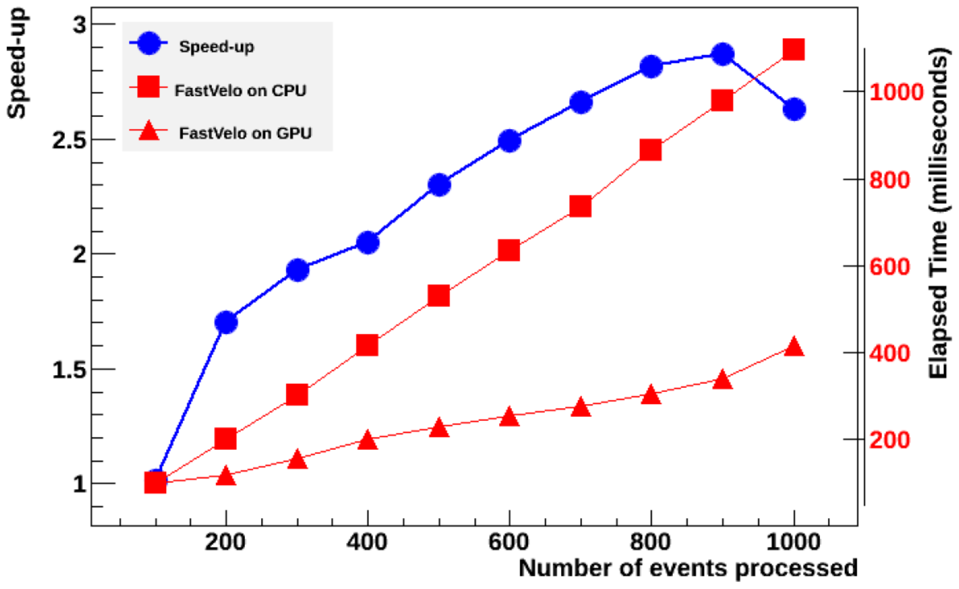


Figure 5.2: Tracking execution time and speedup versus number of events using a 2012 MC sample of $B_s \rightarrow \phi\phi$ decays ($\nu = 2.5$).

A test has been performed also to try to compare the throughput of the GPU and of a multi-core system, an *Intel*® Xeon E5-2600@2.7 GHz with 12 physical and 24 logical cores. The multi-core system can sustain a rate of nearly 5000 ev/sec with an instance of the HLT software per core, while the GPU shows a sustained rate of 2600 ev/sec. It's important to note, though, that it's difficult to compare the performances of heterogeneous architectures which make up the two systems. As this measurement shows, even a 24 logical cores CPU can "beat" a GPU, but the attention should be focused on a more significant parameter as the throughput normalized to operational costs. The price of the high end commodity gaming card used for these tests is currently only a fraction of that of a server class box which equips the HLT farm. So it's more sensible to compare the number of events per second and per unit of currency.

The efficiency and timing measurement for the 10'000 $B_s \rightarrow K^*\mu\mu$ events have been done also with the new hardware configuration of the testbed which is listed in Table 5.2 and replaced the initial desktop class machine.

The tracking performances, provided by the LHCb reconstruction application, are summarized in Table 5.3: the efficiencies of the GPU algorithm are 2-3% lower than the official FastVelo, while clones and ghost rate are at the same level. Figures 5.5 and 5.6 shows the tracking efficiency as a function of the true track momentum and η, respectively, as obtained by the two algorithms; figure

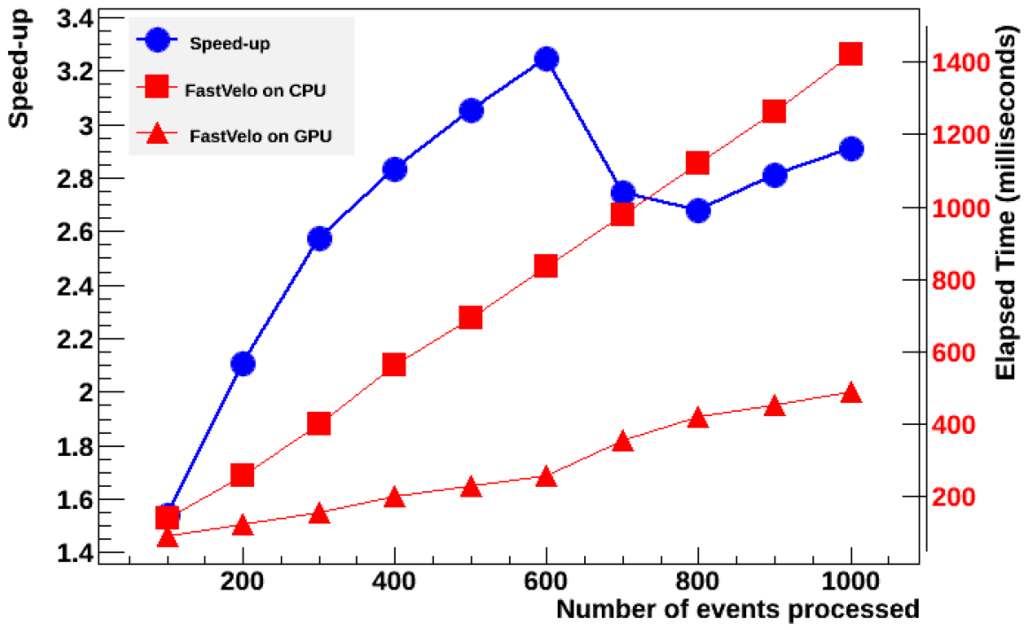


Figure 5.3: Tracking execution time and speedup versus number of events using a sample of NoBias data collected during 2012 run ($\mu = 1.6$).

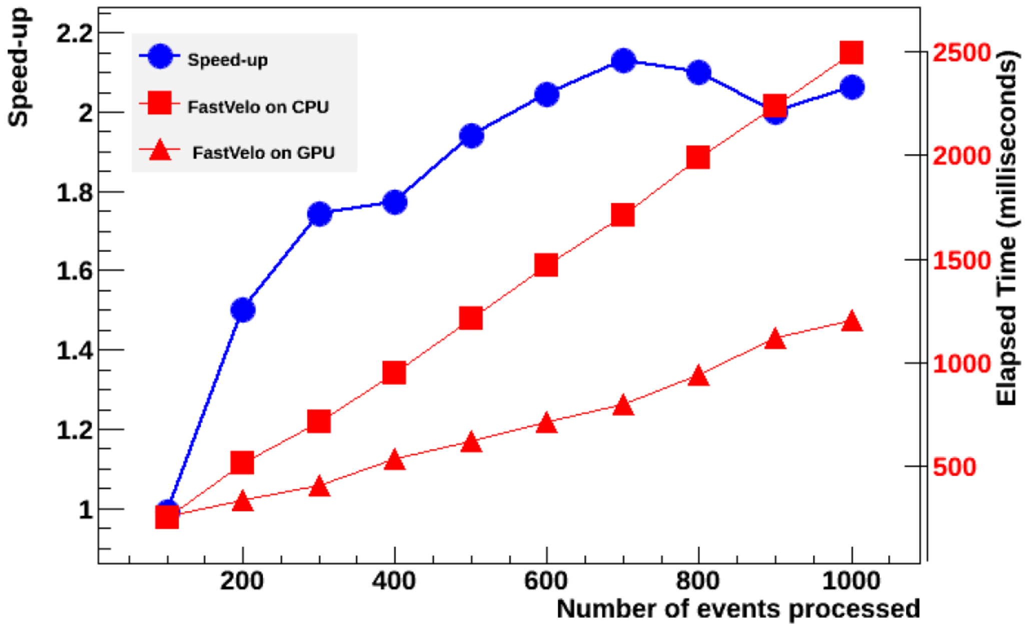


Figure 5.4: Tracking execution time and speedup versus number of events using a 2015 MC sample of b-inclusive decays generated with $\nu = 4.8$.

5.7 shows the CPU-GPU comparison for the impact parameter (IP) resolution as a function of momentum; the overall agreement is good, showing that the GPU implementation does not introduce any distortion on the resolution of the track parameters.

Host machine (typical HLT server)	GPU device
ESC4000 G3 based on $Intel^{®}Xeon^{®}$ processor 16 CPU cores E5-2630v3 @2.40GHz w/ hyper-threading 64 GB of RAM, 145W/socket ≈ 4 kCHF	NVidia GTX Titan X 3072 single-precision CUDA cores 12 GB of RAM, 250/300W ≈ 0.8 kCHF

Table 5.2: The hardware installed in the testbed and used for performance studies.

Track category	FastVelo on GPU		FastVelo	
	Efficiency	Clones	Efficiency	Clones
VELO, all long	90.8%	0.7%	93.7%	0.7%
VELO, long, p > 5 GeV	92.8%	0.6%	95.2%	0.5%
VELO, all long B daughters	91.8%	0.6%	94.3%	0.6%
VELO, long B daughters, p > 5 GeV	92.9%	0.6%	95.1%	0.5%
VELO, long K_s/Λ, p > 5 GeV	84.7%	0.5%	89.3%	0.4%
VELO, ghosts	12.3%		9.7%	

Table 5.3: Tracking efficiencies obtained with FastVelo on GPU, compared with the results obtained by original FastVelo code. The efficiencies are computed using 10000 $B_s \to K^* \mu\mu$ MC events, generated with Run2 conditions.

It's important to note that some parts of the official FastVelo algorithm have not their counterpart on GPU (e.g. the recovery of unused ϕ hits) which causes an efficiency loss of $\approx 1\%$ in the GPU algorithm compared to the sequential one. However, since the aim of this work is focused on the integration of GPUs in the LHCb Online environment rather than writing an alternative Velo tracking algorithm, we considered these results good enough for our purposes. For this reason, the VELO tracking algorithm was not re-written from scratch but the existing FastVelo code was adapted to run on GPU. If GPUs (or other accelerators) will prove to be useful for the LHCb upgrade, parallel reconstruction algorithms could be written in OpenCL [45] and ran on different architectures[1].

From this simple study we cannot conclude that GPU runs faster than a full loaded CPU; a fair and more realistic comparison of the GPU performance has been carried out during the data taking in a real-time environment using the testbed where we measured the GPU throughput and we compared the results with respect to a full HLT server.

[1]The efficiency of the algorithm when running on CPU through OpenCL should be carefully studied.

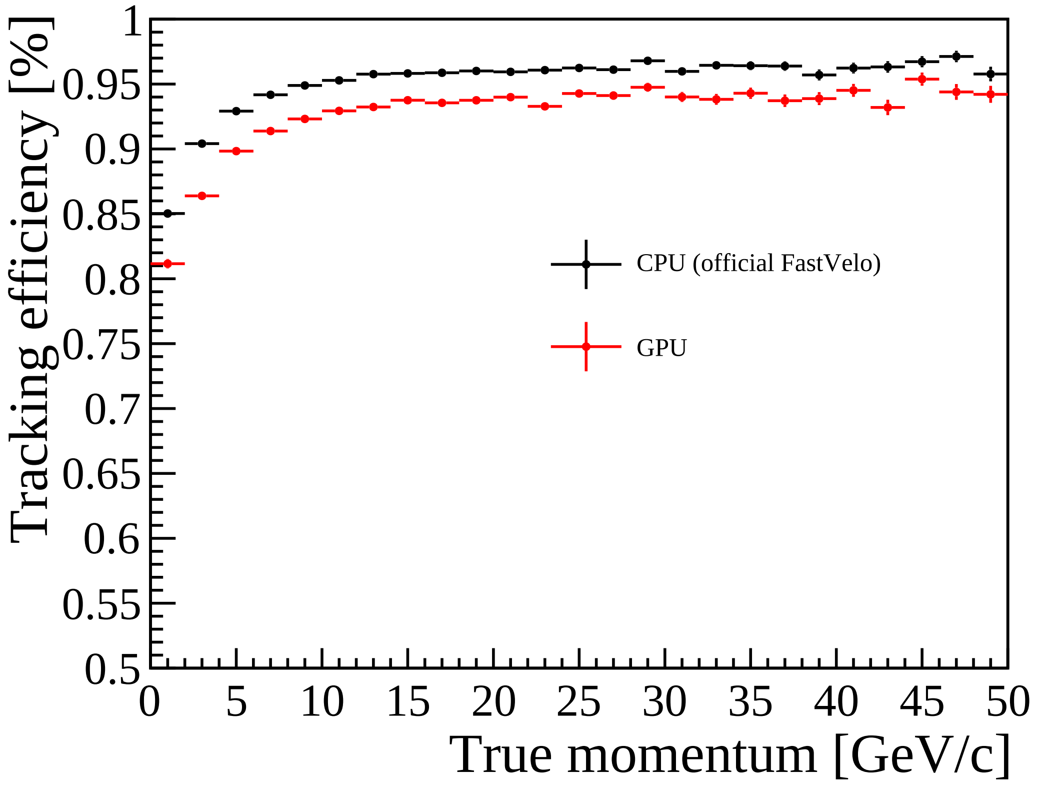


Figure 5.5: Tracking efficiency as a function of the true track momentum using a Run2 simulated sample.

5.3 Physics data

These results were gained during the data taking period with proton-proton and proton-lead collisions.

5.3.1 Timing performance

For the present timing analysis, we used *pp* collisions at 13TeV collected during 2016 (run 184269, threshold setting: *Physics_pp_MidJune2016*). The data acquired by the test-bed have been processed by FastVelo (CPU and GPU versions) without any preselection (except for a global cut on hit multiplicity). The hardware used in the tests is the same listed in Table 5.2. A problem encountered during the testbed operation was the size of the incoming data (i.e. hit multiplicity) which was bigger than the one used during the development of the algorithm; this produced some overflows in the data structures that needed to be tuned during all the tests (e.g. during heavy ions runs).

The throughput measured during data-taking vs the number of clients is shown in Figure 5.8: as expected, the CPU rate is constant being limited by

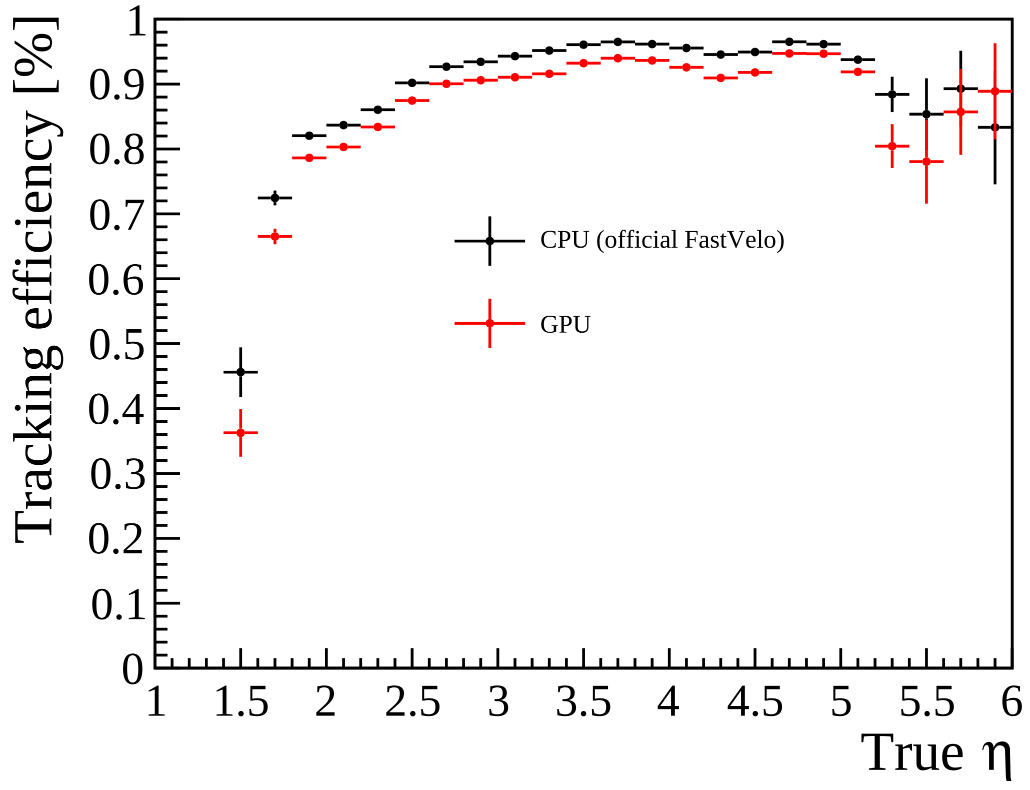


Figure 5.6: Tracking efficiency as a function of the true η using a Run2 simulated sample.

the reduced input rate of the MF, while the GPU performance increases with the number of events. However, the number of available clients is not enough to fully exploit all the processing power of the GPU. More clients would be needed to feed the GPU with the number of events required to fill up its cores (see Figure 5.4), get better performances and stress the system in a more realistic scenario. Table 5.4 shows the numbers plotted on Figure 5.9 together with the average number of events processed by GPU during the tests. It's worth noticing that there is a non negligible overhead ($\mathcal{O}(10\%)$ of the total time) introduced by the data transfer from and to the GPU, as can be seen in Figure 5.9.

An alternative measurement of the throughput has been done re-running data offline: for this test, the events have been equally distributed among all clients ($\approx 300k$ events/client), and the total time was taken from the begin of the test (when the first clients starts) to the end (when the last client finishes). The result of this test is shown in Figure 5.10.

It must be noted that the events feeding the GPU in this context are from a heavy ion beam. The increased multiplicity and complexity of the events

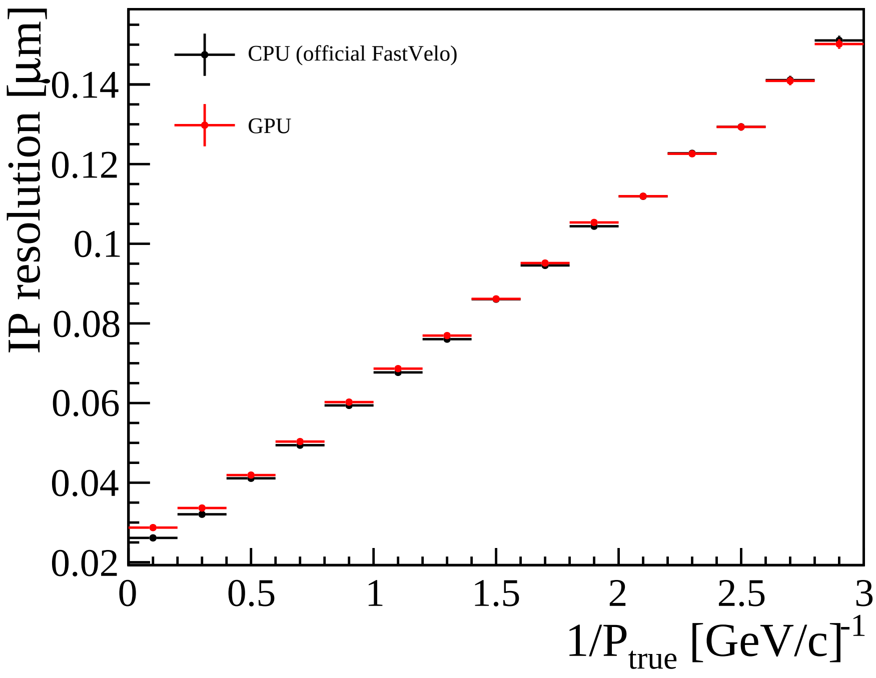


Figure 5.7: Impact parameter resolution as a function of the inverse of the true track momentum using a Run2 simulated sample.

No. clients	**Total time (ms/test)**	**Avg. events / test**
8	27.51	3.66
12	40.43	5.73
16	51.51	7.79
20	56.3	10
24	61.68	12.34

Table 5.4: The total FastVelo GPU time as seen by the HLT client for fill 4715 (time includes data transfer and framework latencies). The average number of events processed by the GPU per test is also reported. The quoted numbers are averaged over several tests.

affected the throughput as more tracks have to be reconstructed, slowing down the whole processing. These events required also a more careful management of memory, as the data structures allocated for standard *pp* collisions proved to be unsuitable for heavy ion events.

5.3.2 Physics performances on Run2 data

Physics performances have been studied using Run2 data collected during 2016. Signal and background yields of the particles reconstructed by the HLT monitoring (detached and prompt D^0, J/ψ, ϕ) have been compared to official reconstruction. For this analysis, the HLT monitor has been re-run offline on

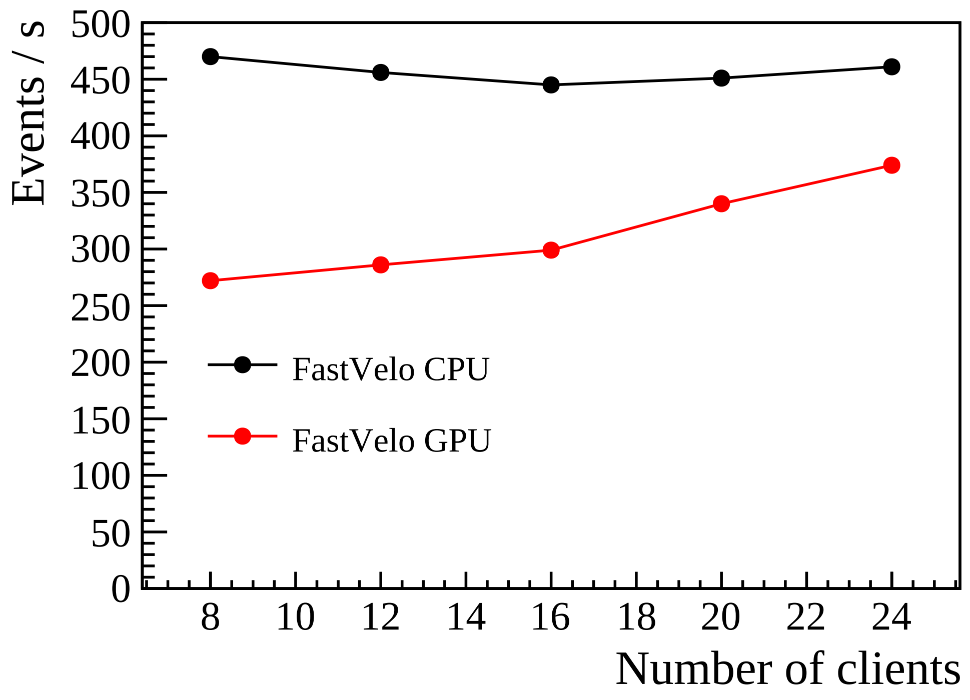


Figure 5.8: Event rate, measured during data-taking, in case of FastVelo running on GPU (red line) and CPU (black line). GPU rate includes latencies due to data transfer. The hardware used for the test is listed in Table 5.2.

the acquired data (threshold setting *Physics_pp_MidJune2016*).

Yields have been extracted by fitting the invariant masses provided by HLT monitor with a single Gaussian for the signal plus an exponential for the combinatorial background. Figures 5.11, 5.12, 5.13 and 5.14 show the mass fit for $D^0 \to K\pi$, detached $D^0 \to K\pi$ and J/ψ candidates, respectively (1M events). Figures 5.16, 5.15 and 5.17 show the mass fit for a D^0 with data taken during the heavy ion operations and proton-lead collisions.

The number of signal candidates reconstructed by FastVeloGpu is $\approx 10{-}15\%$ lower than the one obtained by the official algorithm, while the signal to background ratio is slightly better for the GPU. A $3-4\%$ loss in tracking efficiency was also observed in simulated events (see Tab. 5.3) and it can be partially due to tighter cuts applied in the GPU algorithm to remove clone tracks. Nevertheless, the fitted mass resolutions are very close between CPU and GPU.

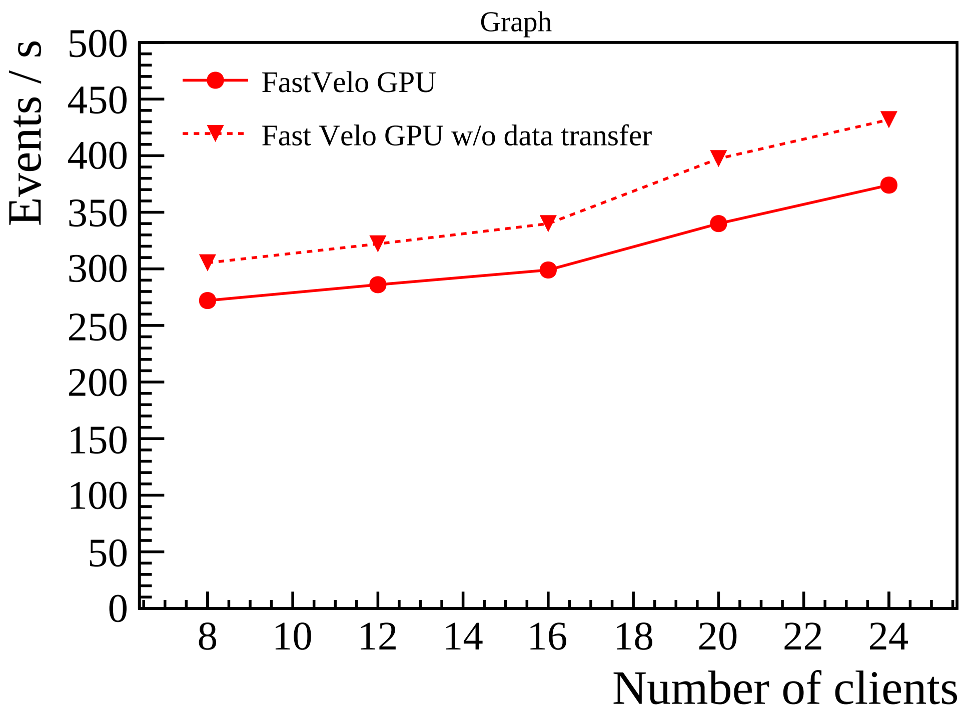


Figure 5.9: FastVelo GPU rate with (solid line) and without (dashed line) the over-head due to data transfer to/from the GPU. The hardware used for the test is listed in Table 5.2.

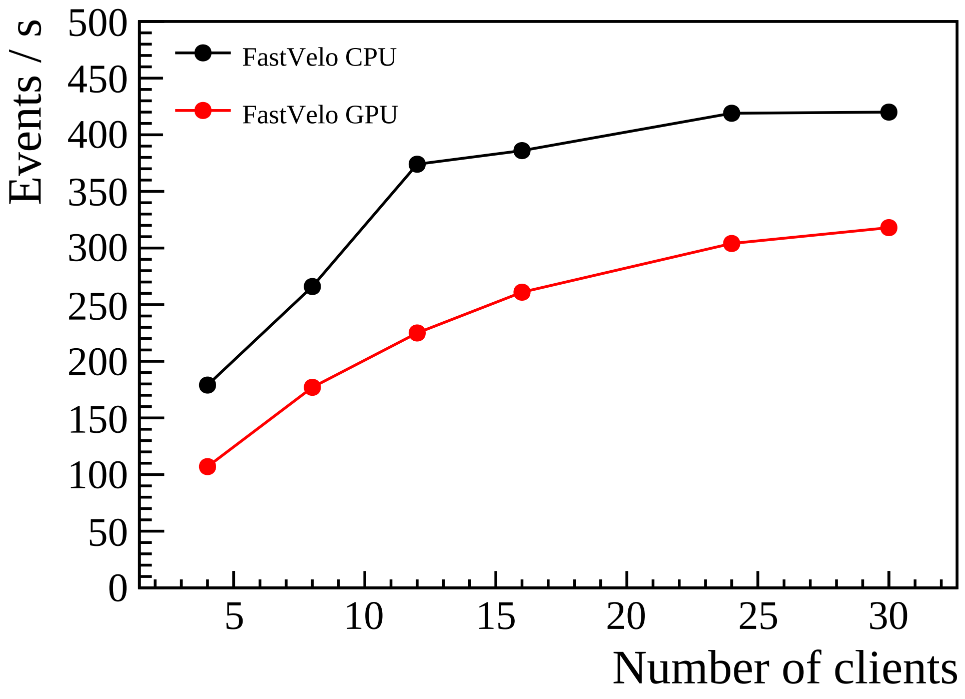
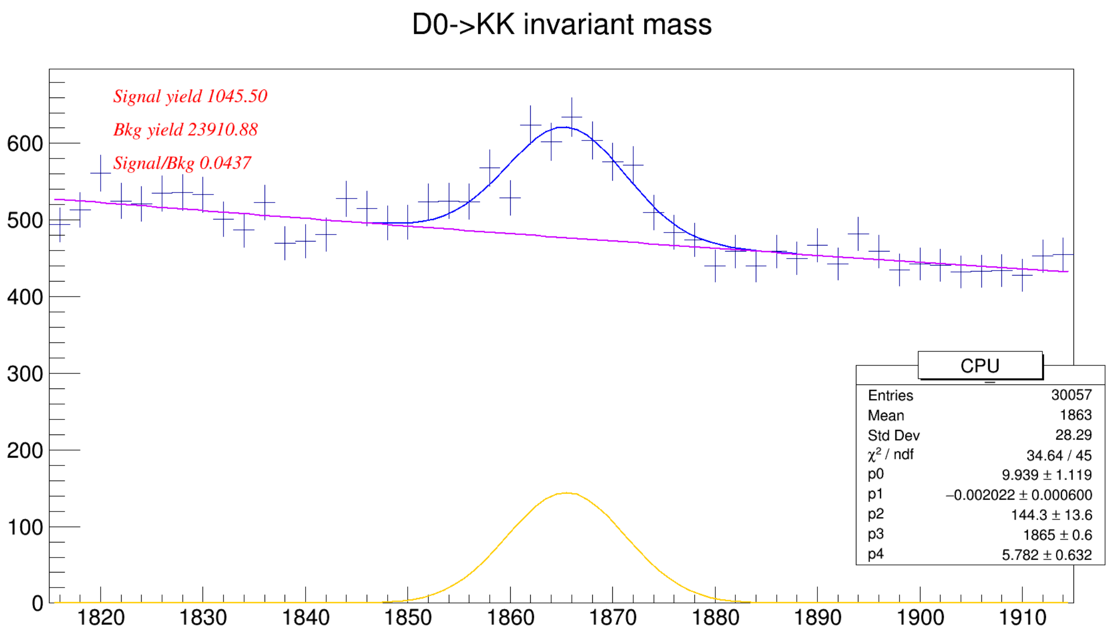


Figure 5.10: Rate comparison (events /s) between CPU and GPU. Test done re-running data offline with 300k events/client. The hardware used for the test is listed in Table 5.2.



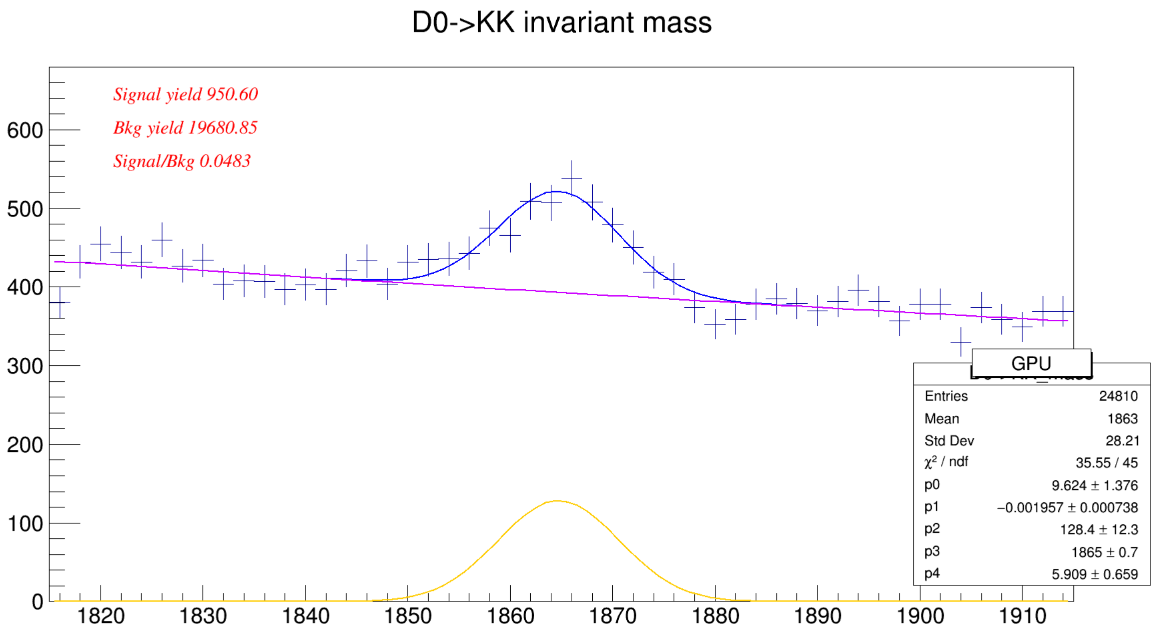
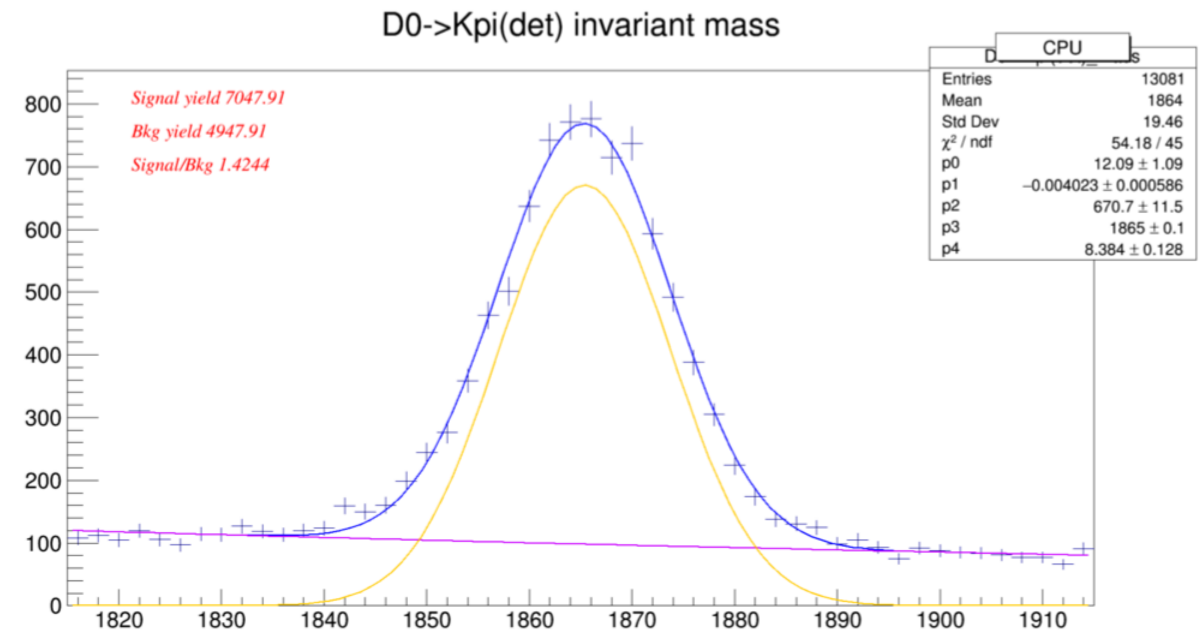


Figure 5.11: $D^0 \to KK$ invariant mass (MeV/c^2). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.



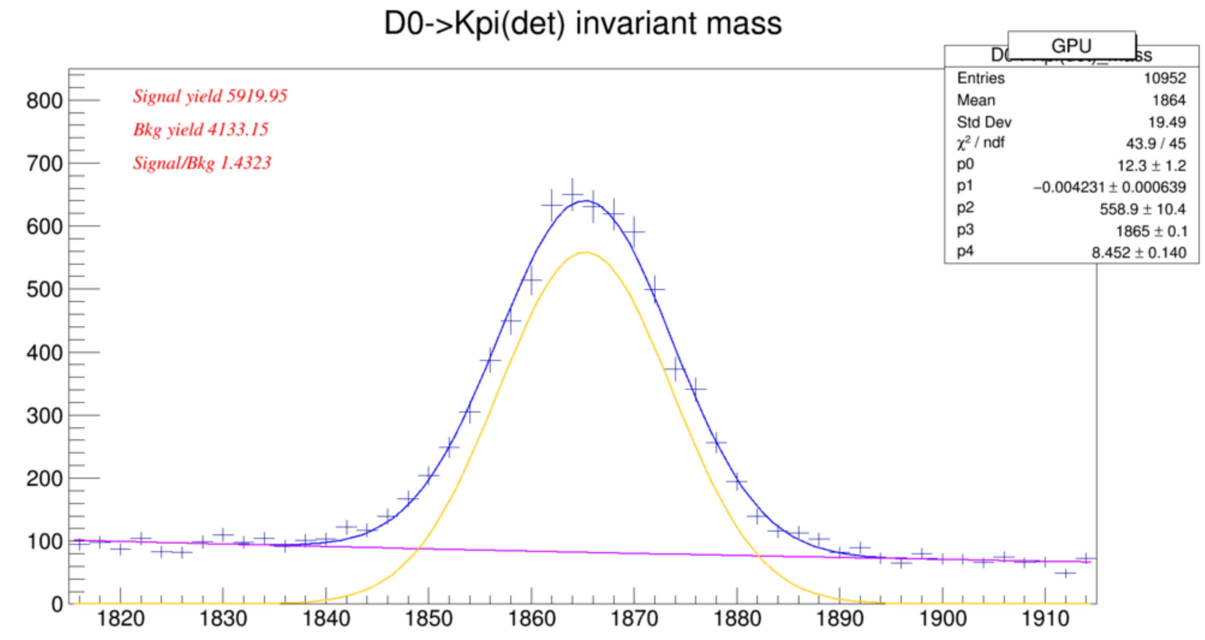
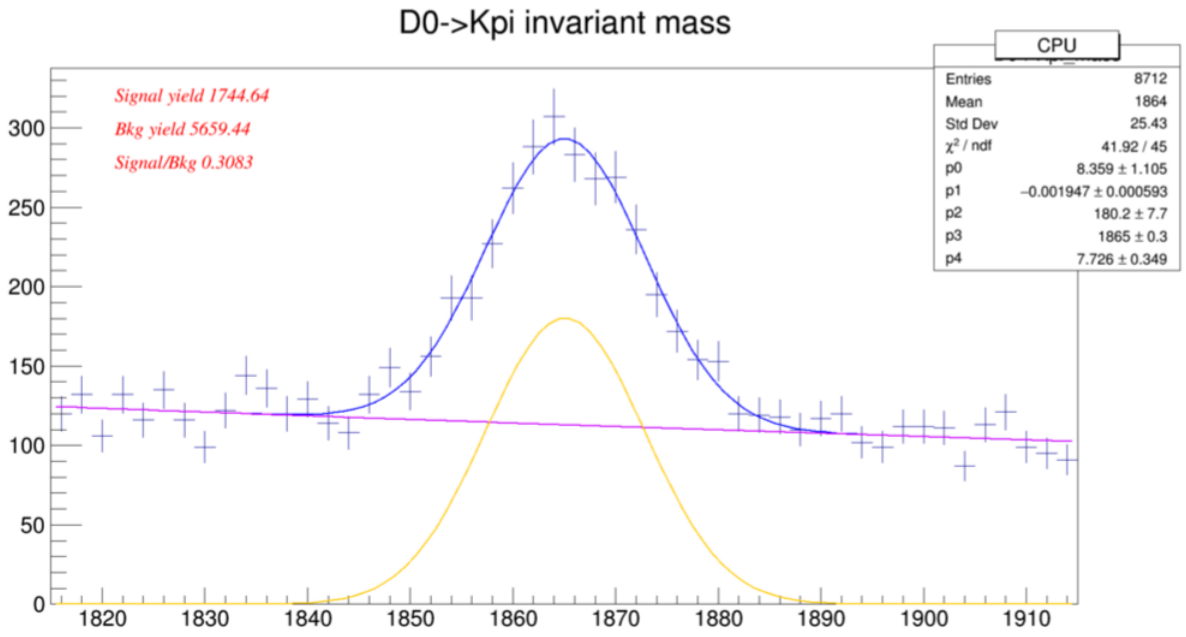


Figure 5.12: $D^0 \to K\pi$ invariant mass for detached D^0 (MeV/c^2). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.



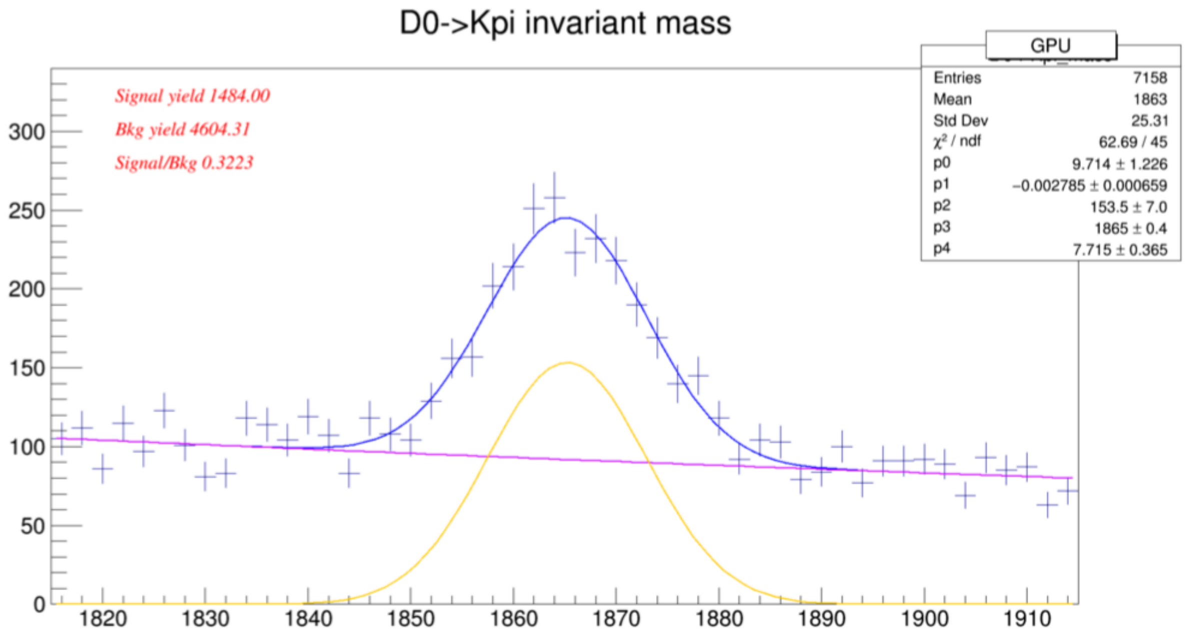


Figure 5.13: $D^0 \to K\pi$ invariant mass (MeV/c^2). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.

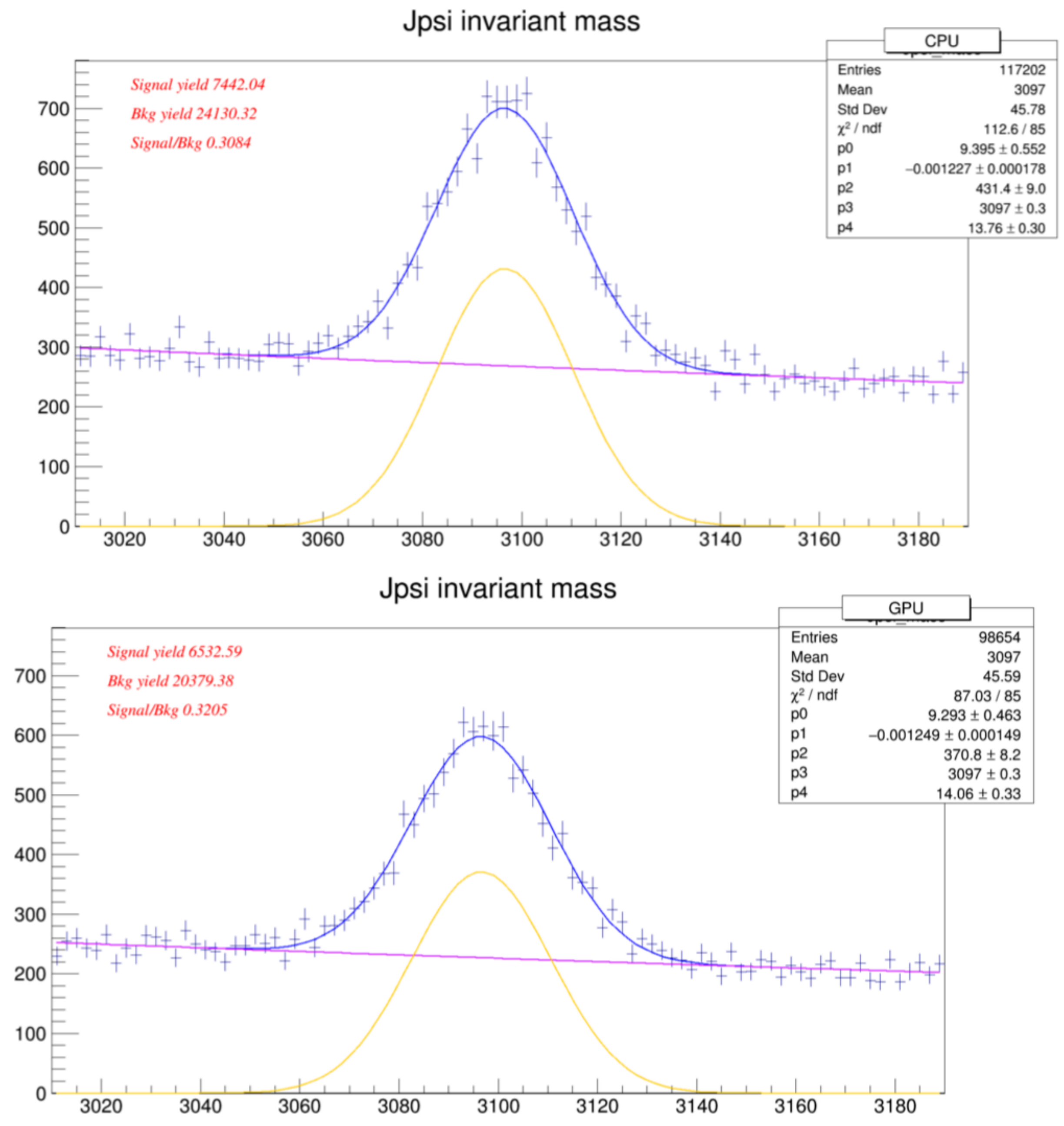


Figure 5.14: J/ψ invariant mass (MeV/c^2). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.

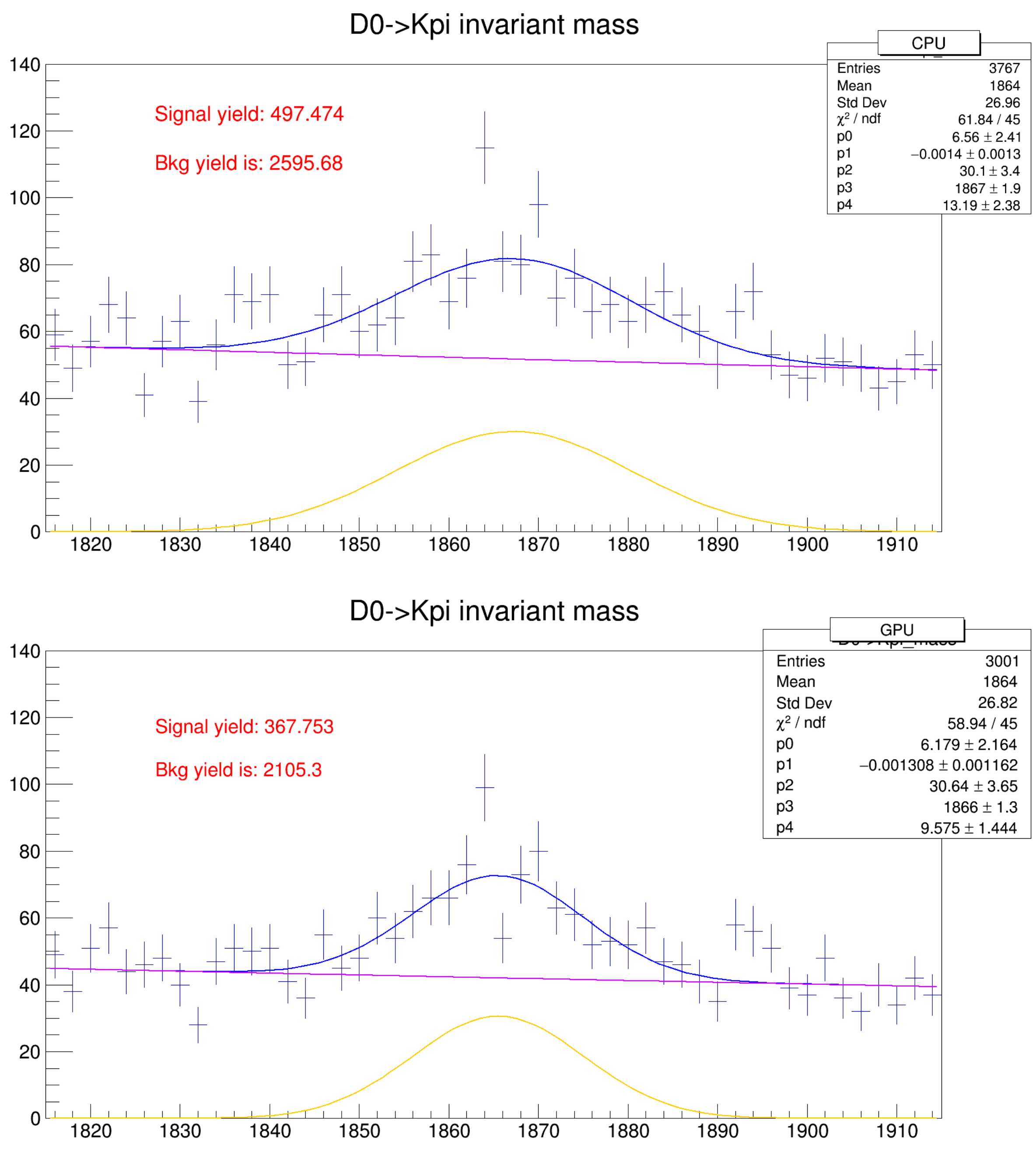


Figure 5.15: $D^0 \to K\pi$ invariant mass (MeV/c^2). Data are taken during the heavy ion operations (proton-lead collisions). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.

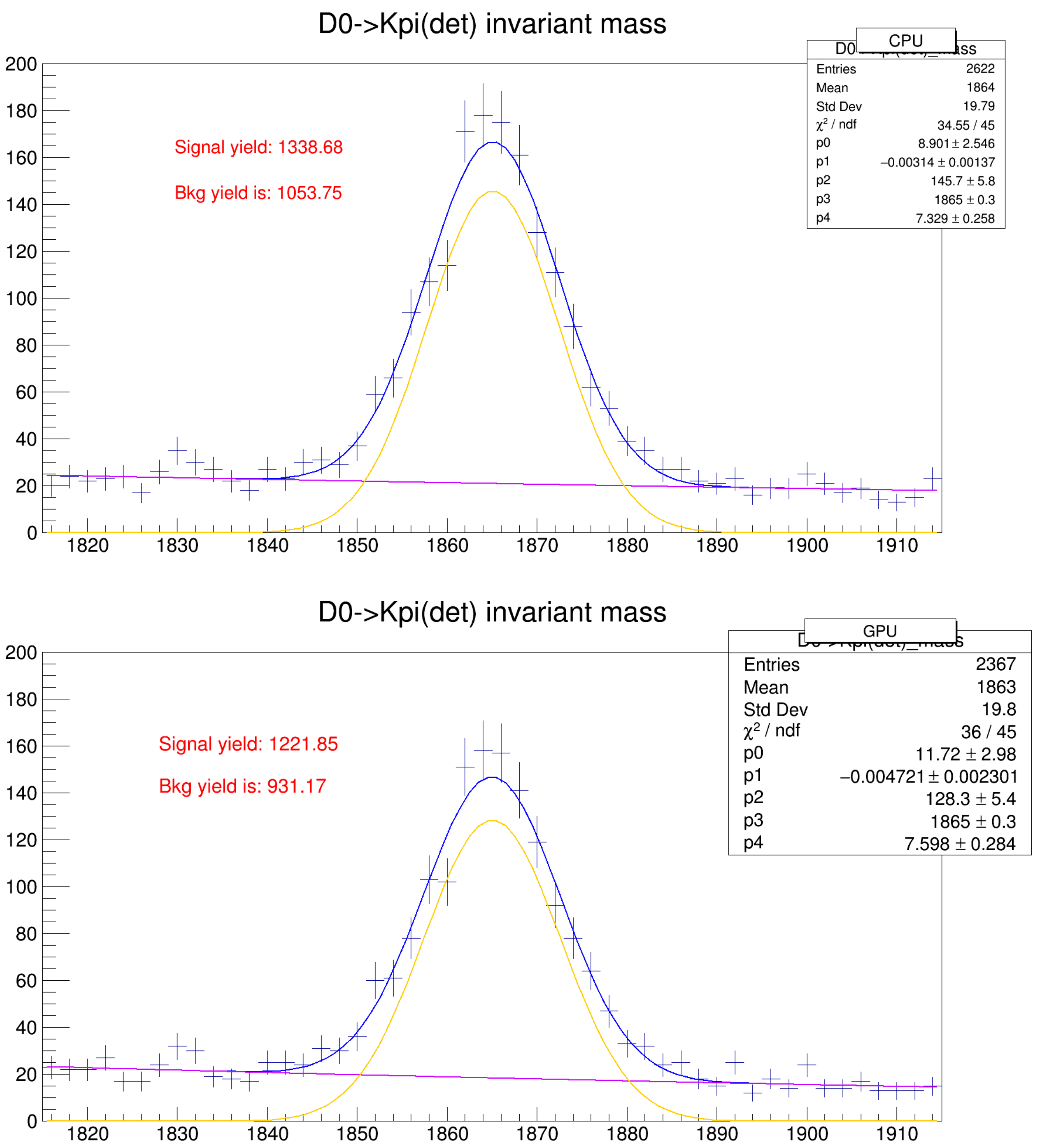
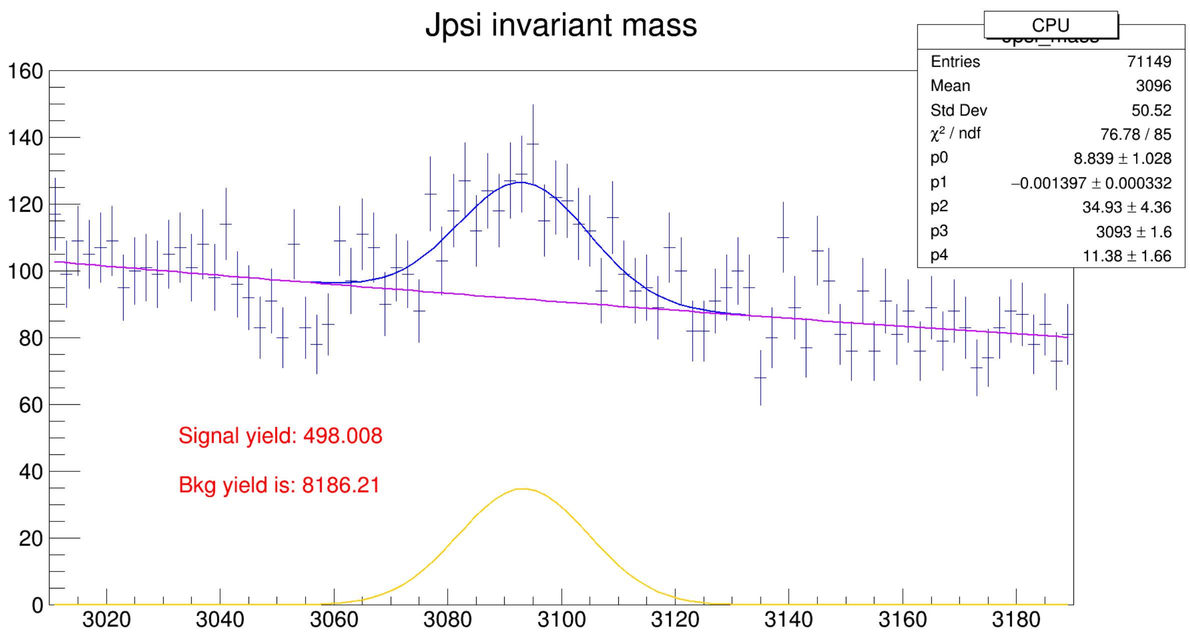


Figure 5.16: Invariant mass of a detached D^0 (MeV/c^2). Data are taken during the heavy ion operations (proton-lead collisions). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.



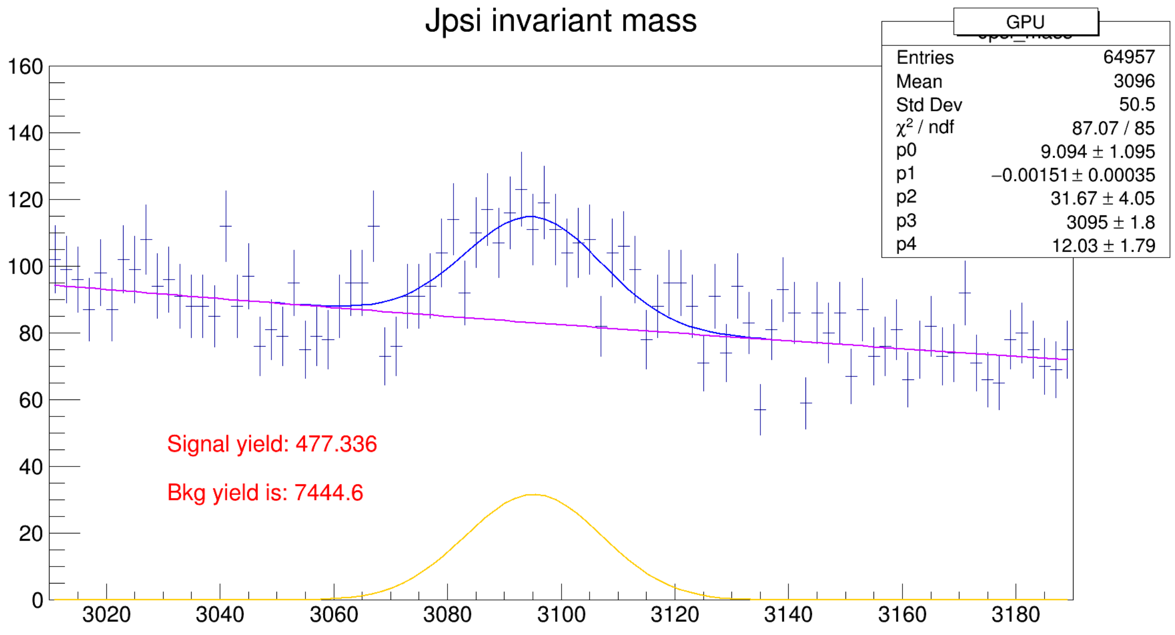


Figure 5.17: J/ψ invariant mass (MeV/c^2). Data are taken during the heavy ion operations (proton-lead collisions). Blue curve is the total fit to data, while orange and purple lines are signal and combinatorial components, respectively. (Upper plot) CPU version. (Lower plot) GPU version.

Chapter 6

Conclusions

The LHCb GPU Project proved to be a very interesting and important opportunity to test the feasibility of a pure software trigger in view of the upgrade of the detector.

The limitations imposed by the current L0 hardware trigger are too tight for the physics program of the collaboration, which would be strongly penalized by the very low trigger efficiency in all hadronic channels. To mitigate this effect, LHCb decided to remove the L0 hardware level and rely on a pure software level trigger. To this goal the collaboration started a thorough review of the possibilities offered by hardware accelerators to speed up its HLT software, along with a deep re-engineering effort to move to a real multi-threaded software.

The first choice fell on *Nvidia*® Graphics Processing Units for their versatility and relative ease of usage. In fact the programming language of these objects is a plain C with only some specific directives to drive the device. The difficulty in exploiting to a full extent these cards is not the language itself, but a radical change in the programming paradigm. It has been showed that the standard Object Oriented design doesn't fit the architecture of these objects, which are suited for a more functional approach and a different way to organize data (Structure of Arrays vs Array of Structures paradigm). These two aspects are relevant when considering the desired learning curve to allow physicists to manage these devices. In our experience this proved to be determinant in the overall time budget of the project while porting the original FastVelo code to a CUDA-compliant one. The relative poor speedup gained with 2015 simulated data should be reviewed, also, under this consideration: despite the efforts made to adapt the code to a real SoA paradigm, it remained conceptually the same as the one developed for a genuine serial workflow. This can be further extended when considering the huge amount of branching that the original code introduced to speed up the performances. While this approach proved to be determinant when applied to the current HLT 2, it showed its intrinsic limitations when applied to devices which suffer from "warp divergence".

This should make physicists aware that the algorithms developed so far should be redesigned to comply to the most modern software paradigms, e.g. multi-threading. LHCb is indeed actively involved in reviewing all the current reconstruction and analysis algorithms to write them in a fashion that's able to exploit the new task based parallel framework GAUDI. To sustain this effort, the collaboration started a program of courses on modern C++ and regular "hackathons", where users can ask for advice and submit their code to experts, who then provide feedback or design solutions to the proposed issues.

Another consideration that is worth citing is the unavoidable and penalizing overhead introduced by the offloading mechanism. GPUs are fed up by PCIe buses which, despite being "fast", are nevertheless adding a latency for moving data back and forth the main memory of the device. Our measurements show that this overhead amounts to nearly 10% of the overall computing time. Although the technological trend shows already an improved version of the PCIe bus, with a bandwidth that will be 4 times larger than the current PCIe 3.0 in a couple of years, it's clear that this will always be a bottleneck for machines whose main computing power will still reside in the CPU.

One of the most important things that these studies revealed, is that, in a world where hardware concurrency is, by now, a standard *de facto*, it is no more affordable to postpone a thorough review of the whole software stack towards a task based parallelism (see [38,42]). Other LHC experiments, namely CMS, already deployed a full multi-core computing infrastructure [46] to their Tier0 and Tier1 sites, in order to have the flexibility to adapt specific job requirements to the resources available on a given machine. In our experience one more aspects arose: the performance of hardware accelerators is strongly limited by the lack of a pure concurrent multi-event framework. Currently the LHCb framework manages only one event at a time sequentially. This mechanism penalizes devices such as GPUs because it cannot provide enough events to keep the occupancy to the full capacity. A partial mitigation of this effect has been achieved using more clients which concurrently feed the GPU, providing a kind of "fake" multi-event environment, but it was clear that, even with a reduced speed-up, we would need $\mathcal{O}$ (100) events in-flight to fully exploit the computing power of even a commercial gaming card. This goes in the same direction of what was said earlier, that a modern physics experiment cannot overlook a genuine multithreaded and parallel software able to take advantage of the multi- and many-core hardware revolution.

Last but not least, a word on the physics performances: the studies performed during this project where more devoted to computational aspects. Nevertheless these didn't overlook the final goal, that is provide a faster software which reconstructs particles inside the LHCb detector. Despite some inevitable discrepancies, the results where quite in agreement with those of the standard code. It is plain that the agreement must be instead complete, and this implies also further studies with simulated events to guarantee an exact match. The algorithm was also partially modified and some thresholds tightened, in order to cope with some limitations on the GPU version of the code, but in the end the "physics" behind the computation was preserved to a satisfying degree.

It must be said, furthermore, that it's not always easy to define some metrics with which to compare the performances of different architectures such as CPU and GPU. A sensible one could be the performances normalized to the cost, in order to spot the best balance between "standard" machines and GPU-equipped ones.

Nevertheless the critical aspect which must be considered is the lack of a factor ~ 6 in the time budget for the full reconstruction of a typical LHCb event after the upgrade, a factor that must be cut.

Currently it's not possible to say a definitive word on the decision that the LHCb collaboration will take regarding the usage of these devices. In fact according to the Computing TDR, the baseline for the new HLT are still standard CPUs, but a door has been kept open to allow for a more aggressive and performing solution using hardware accelerators [47].

Bibliography

[1] C. conseil europeen pour la recherche nucleaire, *Lhc*, `https://home.cern/topics/large-hadron-collider`.

[2] CERN, *The ATLAS experiment*, `https://home.cern/about/experiments/atlas`.

[3] CERN, *The CMS experiment*, `https://home.cern/about/experiments/cms`.

[4] CERN, *The ALICE experiment*, `https://home.cern/about/experiments/alice`.

[5] CERN, *The LHCb experiment*, `https://home.cern/about/experiments/lhcb`.

[6] LHCb collaboration, A. A. Alves Jr. *et al.*, *The LHCb detector at the LHC*, JINST **3** (2008) S08005.

[7] LHCb collaboration, R. Aaij *et al.*, *LHCb detector performance*, Int. J. Mod. Phys. **A30** (2015) 1530022, `arXiv:1412.6352`.

[8] M. Adinolfi *et al.*, *Performance of the LHCb RICH detector at the LHC*, Eur. Phys. J. **C73** (2013) 2431, `arXiv:1211.6759`.

[9] LHCb Collaboration, S. Amato *et al.*, *LHCb calorimeters: Technical Design Report*, Technical Design Report LHCb, CERN, Geneva, 2000.

[10] A. A. Alves Jr. *et al.*, *Performance of the LHCb muon system*, JINST **8** (2013) P02022, `arXiv:1211.1346`.

[11] R. Aaij *et al.*, *Performance of the LHCb Vertex Locator*, JINST **9** (2014) P09007, `arXiv:1405.7808`.

[12] LHCb Silicon Tracker Group, M. Tobin, *The LHCb Silicon Tracker*, Nucl. Instrum. Methods Phys. Res. , A **831** (2016) 174.

[13] R. Arink *et al.*, *Performance of the LHCb Outer Tracker*, JINST **9** (2014) P01002, `arXiv:1311.3893`.

[14] LHCb Collaboration, P. R. Barbosa-Marinho *et al.*, *LHCb inner tracker: Technical Design Report*, Technical Design Report LHCb, CERN, Geneva, 2002. revised version number 1 submitted on 2002-11-13 14:14:34.

[15] LHCb Collaboration, P. R. Barbosa-Marinho *et al.*, *LHCb outer tracker: Technical Design Report*, Technical Design Report LHCb, CERN, Geneva, 2001.

[16] G. D. McGregor, *Calibration of the LHCb VELO Detector and Study of the Decay Mode* $D^0 \to K^-\mu^+\nu_\mu$, PhD thesis, Manchester U., 2011.

[17] LHCb RICH Collaboration, A. Papanestis and C. D'Ambrosio, *Performance of the LHCb RICH detectors during the LHC Run II. Performance of the LHCb RICH detectors during the LHC Run II*, Tech. Rep. LHCB-PUB-2017-012. CERN-LHCb-PUB-2017-012, CERN, Geneva, Mar, 2017. Updated authors' details and DOI.

[18] R. Aaij *et al.*, *The LHCb trigger and its performance in 2011*, JINST **8** (2013) P04022, `arXiv:1211.3055`.

[19] *LHCb Trigger and Online Upgrade Technical Design Report*, Tech. Rep. CERN-LHCC-2014-016. LHCB-TDR-016, May, 2014.

[20] LHCb Collaboration, R. Antunes-Nobrega *et al.*, *LHCb trigger system: Technical Design Report*, Technical Design Report LHCb, CERN, Geneva, 2003. revised version number 1 submitted on 2003-09-24 12:12:22.

[21] S. Stahl, *The LHCb High Level trigger in Run 2*, .

[22] L. Collaboration, *LHCb Tracker Upgrade Technical Design Report*, Tech. Rep. CERN-LHCC-2014-001. LHCB-TDR-015, Feb, 2014.

[23] L. Collaboration, *LHCb PID Upgrade Technical Design Report*, Tech. Rep. CERN-LHCC-2013-022. LHCB-TDR-014, Nov, 2013.

[24] LHCb collaboration, J. Albrecht *et al.*, *The lhcb trigger system: Present and future*, J. Phys. : Conf. Ser 623 (2015) 12003, published in J. Phys.

[25] LHCb collaboration, I. Bediaga *et al.*, *Framework TDR for the LHCb Upgrade: Technical Design Report*, Tech. Rep. CERN-LHCC-2012-007. LHCb-TDR-12, Apr, 2012.

[26] N. Neufeld, *Evolution of computing hardware in the coming years*, , LHCb Week December 2016.

[27] LHCb collaboration, A. Badalov *et al.*, *The lhcb gpu acceleration project*, Journal of Instrumentation (2015) , published in JInst.

[28] A. J. Bevan, *The SuperB Experiment*, AIP Conf. Proc. **1182** (2009) 418, `arXiv:0906.1008`.

[29] S. Longo *et al.*, *A parallel framework for the SuperB super flavor factory*, in *Proceedings, 2012 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC 2012): Anaheim, California, USA, October 29-November 3, 2012*, pp. 2024–2029, 2012. doi: 10.1109/NSSMIC.2012.6551469.

[30] NA62, R. Ammendola *et al.*, *GPU real-time processing in NA62 trigger system*, J. Phys. : Conf. Ser. **800** (2017), no. 1 012046. 4 p.

[31] J. Nickolls *et al.*, *Scalable parallel programming with cuda*, Queue - GPU computing (2008) , published in Queue.

[32] S. Gorbunov *et al.*, *Alice hlt high speed tracking on gpu*, IEEE Trans. Nucl. Sci. **58** (2012) 1845, published in IEEE Trans. Nucl. Sci.

[33] D. Emeliyanov *et al.*, *Gpu-based tracking for atlas level 2 trigger*, , Atlas note.

[34] P. Mato, *Gaudi: Architecture design document*, , Cern note.

[35] Intel, *Pentium d*, `https://ark.intel.com/it/products/27512/Intel-Pentium-D-Processor-820-2M-Cache-2_80-GHz-800-MHz-FSB`.

[36] AMD, *Opteron*, `https://en.wikipedia.org/wiki/Opteron`.

[37] I. Foster, *Designing and building parallel programs*, 1995. Addison Wesley.

[38] I. Shapoval, *Adaptive scheduling applied to non deterministic networks of heterogeneous tasks for peak throughput in concurrent gaudi*, CERN-THESIS-2016-028. LHCb.

[39] D. H. Campora Perez, *LHCb Kalman lter cross architecture studies*, .

[40] R. Jones, G. Stewart, C. Leggett, and B. Wynne, *Evolution of the ATLAS Software Framework towards Concurrency*, Tech. Rep. ATL-SOFT-PROC-2014-008. 1, CERN, Geneva, Oct, 2014.

[41] INTEL, *The threading building blocks*, `https://www.threadingbuildingblocks.org/`.

[42] B. Hegner *et al.*, *Evolving lhc data processing frameworks for efficient exploitation of new cpu architectures*, IEEE NSS/MIC (2012) 2003, Published in NSS/MIC journal.

[43] O. Callot, *FastVelo, a fast and efficient pattern recognition package for the Velo*, LHCb-PUB-2011-001. CERN-LHCb-PUB-2011-001. LHCb.

[44] M. Dankel *et al.*, *Use of hardware accelerators for atlas computing*, , Atlas note.

[45] Khronos Group, *OpenCL - The open standard for parallel programming of heterogeneous systems*, `https://www.khronos.org/opencl/`, 2017.

[46] CMS Collaboration, A. Perez-Calero Yzquierdo, *CMS Multicore Scheduling Strategy*, Tech. Rep. CMS-CR-2013-387, CERN, Geneva, Oct, 2013.

[47] C. Bozzi *et al.*, *Upgrade Software and Computing TDR: a roadmap*, Tech. Rep. LHCb-INT-2016-016. CERN-LHCb-INT-2016-016, CERN, Geneva, Mar, 2016.