Doctoral Dissertation
Doctoral Program in Electronic Engineering and Telecommunications (32^{th} cycle)

Primary vertex reconstruction using GPUs for the upgrade of the Inner Tracking System of the ALICE experiment at LHC

Matteo Concas
* * * * * *

Supervisors
Prof. Danilo Demarchi, Supervisor
Prof. Stefania Bufalino, Co-supervisor

Doctoral Examination Committee:
Dr. Thorsten Kollegger, Referee, GSI Helmholtzzentrum für Schwerionenforschung
Prof. Donatella Lucchesi, Università di Padova
Prof. Marco Paganoni, Referee, Università di Milano Bicocca
Dr. Sara Pirrone, Istituto Nazionale di Fisica Nucleare, sez. Catania

Politecnico di Torino
July 28, 2020

I hereby declare that the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Matteo Concas
Turin, July 28, 2020

Abstract

In 2021 the Large Hadron Collider at CERN will start its third data taking period, the so-called Run 3, which will last until 2023. During the Long Shutdown 2, the pause between Run 2 and Run 3, the four major experiments at the accelerator (ALICE, ATLAS, CMS, LHCb), are undergoing through major upgrades and working on the apparatus maintenance. In particular, the ALICE (A Large Hadron Collider Experiment) experiment is performing a huge upgrade of some detectors at the hardware level and a big effort is dedicated to the online and offline processes to improve the data acquisition and processing. The ALICE experiment aims at studying the Quark-Gluon Plasma, a state of matter where quarks and gluons are not confined into hadrons and that can be inspected and characterised using high-energy ion collisions. Plans for ALICE in Run 3 include the collection of 10 nb^{-1} of Pb–Pb collisions, with an instantaneous luminosity up to 6 $\times 10^{27}$ $cm^{-2}s^{-1}$ and a collision rate of 50 kHz at 5.5 TeV, corresponding to a total of $10^{1}1$ interactions recorded. This is the minimum rate required to address the proposed physics programme that focuses on rare probes both at low and high momentum. For the proton collisions programme, the experimental apparatus will acquire data with a rate up to 400 kHz of interactions at 13 TeV, to obtain a a meaningful data reference, instrumental for the heavy-ion physics programme. ALICE will also move to a brand new paradigm for what concern the data acquisition of many sub-detectors, the so-called continuous readout. This approach foresees a trigger-less mode to acquire data continuously; furthermore, data are processed and reconstructed as a stream of data in an on-line mode during the data taking. Such conditions impose stringent constraints to the detector performance in terms of acquisition rate and challenges for what concerns the output data bandwidth, forcing a general upgrade of the hardware and the software of the experiment.

To comply with this ambitious scenario, many interventions are being operated on sub-detectors of the ALICE apparatus, mostly dedicated to the readout electronics and to sensor readout capabilities, to provide a faster acquisition rate compared to the Run 2 working conditions. Two brand new detectors have also be designed and constructed to improve the precision of physics measurements: a Muon Forward Tracker (MFT) and a completely renovated Inner Tracking System (ITS), the sub-detector dedicated to the reconstruction of the interaction vertex, placed within the innermost section of the detector, in the central barrel volume of the ALICE apparatus. The latter is a cylindrical detector built up with seven concentric layers of silicon pixels chips adopting a Monolithic Active Pixels Sensors (MAPS) layout, a new technology for faster and more precise measurements. The upgraded layout is characterised by finer pixel granularity, inner barrel closer to the beam pipe with respect to the ITS used during the LHC Run2 and counts one additional layer for a total of 7 layers. These hardware features translate in a better spatial resolution of the interaction point (tridimensional space point where beams collide) and better resolution for low momentum particles and tracks parameters useful at the data analysis level.

The continuous readout set also critical challenges in terms of online data reduction, compression and reconstruction. It is estimated that the whole online reconstruction workflow will be able to cope with a 3.5 TB/s bandwidth of data in input and will write $\sim$ 100

GB/s of data on persistent storage. To enable the processing of continuous readout data stream it has been developed a completely renewed software stack which includes parts to be run both online and offline on reconstructed data. The new framework is called Online-Offline (O^2). Such a new platform is thought to be deployed both on large computing clusters like the Event Processing Node farm (EPN), the main computing centre used for online reconstruction, located on-site close to the ALICE experimental apparatus, but also on smaller computing resources, down to the single workstation of a scientist. The O^2 framework presents a new operational design, compared with former software used in ALICE. It is based on workflows operated by generic logical "devices": computing processes that aims at continuously execute some routine depending on the designed roles. Devices are then *plugged* each others to create actual pipelines or topologies, where they can communicate and cooperate to execute more complex tasks. The O^2 embodies a multi-process paradigm where each device is represented by a process on a computing resource and the workflows are described by topologies connecting devices following a data-flow models. More in details, each device is responsible to perform a piece of a large routine and the output data are exchanged across devices by mean of messages that can be implemented using different technologies, depending on the underlying computing infrastructure. Every type of workflow in the O^2 is described and implemented using this design, from Monte Carlo simulation to online data reconstruction and even the analysis framework is being developed using the same core approach. The schema is flexible and scalable, each entity can be, if required, replicated to cope with specific demands and the deployment of a workflow can be done on a laptop as well as on a High-Performance Computing cluster, in some cases. These features are completely transparent to the final user, which will be able to agilely work with both resources with minimal techincal effort. Ultimately, the framework does support heterogeneous architectures, allowing devices to offload payloads on computing accelerators like Graphic Processing Units (GPUs) and Fully Programmable Field Arrays (FPGAs), to obtain high-throughput computations with a fully integrated data model to support them.

The work in this thesis will present the design, development and implementation of a primary vertex reconstruction algorithm with ITS-only data that will be used by ALICE in the online reconstruction phase during the Run 3. The estimation of the primary vertex position is instrumental for calibrating detectors because it provides useful information on the beam position. More importantly, it is a critical information for some online reconstruction processes such as the ITS tracking, the process that reconstruct the trajectories of charged particles, as a critical starting point for the process. The work presented is integrated in the O^2 framework, and provides both a CPU and a GPU-accelerated parallel version of the algorithm. The algorithm is able to identify also the so-called pile up of events, when data related to more than one collision vertex are present in the same input dataset, and provide their position with a resolution compliant with the physics programme requests. Concerning the GPU version, two implementations are presented: on using Nvidia GPUs and one using Advanced Micro Devices (AMD) GPUs. Two versions are coded using the two corresponding development frameworks, the Computing Unified Device Architecture (CUDA) for Nvidia and Heterogeneous-Computing Interface for Portability (HIP) from AMD. Consistency checks among three implementations are performed, together with performance benchmarks. Time measurements are also reported for comparison the 3 implementations. The primary vertex reconstruction is proven to be compliant with the O^2 requirements in

terms of resolution and time performance.

Knowledge acquired in working on this will be used in future also to further extend the dominion of the GPU-accelerated workflows in the O^2 context.

La natura, dal canto suo,
è quella che è, e noi la
scopriamo pian piano.
Se la nostra grammatica e
la nostra intuzione non si
adattano a quello che
scopriamo, poco male,
cerchiamo di adattarle.

– C. Rovelli

Contents

List of Figures

List of Tables

Chapter 1

Upgrade of the ALICE Inner Tracking System detector

The Large Hadron Collider (LHC) is the largest and most powerful circular accelerator and collider in the world. It is 26.7 Km long and able to deliver the high energy density required to melt hadronic matter. Large fraction of the LHC uptime is dedicated to the proton–proton (pp) physics. It led to the discovery of the Higgs Boson[2, 34] and of two charmed pentaquark states[3]. Significant part of the physics programme is also dedicated to heavy-ion physics and the characterisation of the Quark Gluon Plasma (QGP)[72, 29, 64], a state of the matter that is formed at extremely high energy density and temperature. A Large Ion Collider Experiment (ALICE) among the other experiments, is mainly focused on that, since it aims at the investigation of the properties of the QGP.

1.1 Studying the Quark-gluon plasma

The QGP is an exotic state of the nuclear matter that resembles the conditions occurred at the early stages of the formation of the Universe when there was a regime of almost infinite energy density and temperature. Some physics models[50, 48] foresee its occurrence in massive astronomical objects like the inner core of neutron stars due to the extreme compression of nuclear matter. The QCD Phase diagram of strongly interacting matter shown in figure 1.1, describes the thermodynamical states of nuclear matter as a function of the temperature T and the baryon-chemical potential μ_b, a parameter represents the net density of baryons and ensures its total conservation. For large values of the baryon-chemical potential, a first-order transition between hadronic matter and QGP should occur, whereas a crossover is expected at lower μ_b. Moving along the two axes at the two most meaningful extremes there can be found two exotic behaviours:

- $T \gg 0$ and $\mu_b \approx 0 \rightarrow$ strongly interacting Quark Gluon Plasma (sQGP)
- $T \approx 0$ and $\mu_b \gg 0 \rightarrow$ coloured superconductor

During the earliest stages in the evolution of the Universe, the temperature was too high for elementary partons to combine into bound states, the Universe had first to cool down in

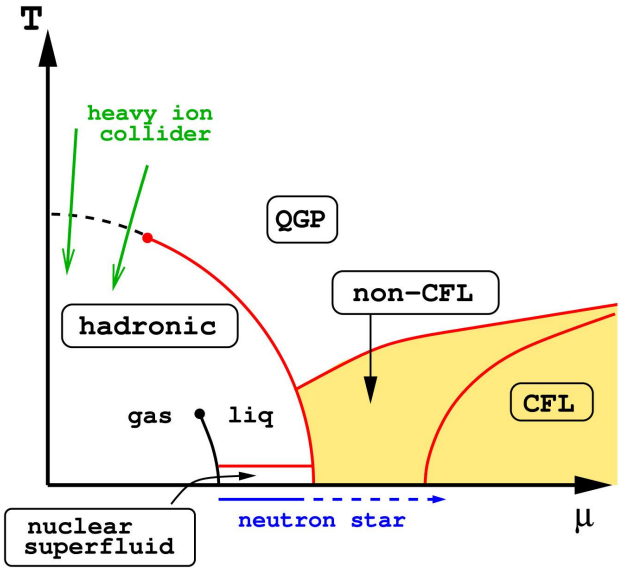


Figure 1.1: phase diagram

order to form hadrons at the beginning most were unstable. Mesons are unstable hadron composed by a quark and an anti-quark, with a relatively short lifetime meaning they decay after an average period, depending on the type of meson and they do not combine in ordinary matter (the matter that can be found with ordinary temperature and energy density regimes). (Anti-)Baryons are particles composed by three (anti-)quarks, they may exist in a stable form in the core of some neutron stars, otherwise they are unstable and decay after a characteristic lifetime. In the early stage of the formation of the Universe took place also the annihilation between matter and anti-matter, which left an excess of matter, in contrast with predictions from models that usually foresee a symmetry between matter and anti-matter that would lead to have them in equal proportions. This phenomenon has not been completely understood. With the Universe lowering its temperature more stable hadrons were produced, a phenomenon called *primordial nucleosynthesis* nucleons, stable baryons such as protons and neutrons, combined together to produce nuclei up to the Lithium. At this point of the process the Universe is "opaque", in the sense that electromagnetic radiation, interacting with charged medium made of ions and electrons, was not able to freely travel across the space. When the temperature reached (T$\sim$3000K), electrons and nuclei were confined into electrically neutral atoms, becoming transparent to the electromagnetic radiation. From that time onwards the electromagnetic radiation started propagating throughout the space. The opacity to radiation, before to the decoupling of photons, limits the maximum moment in time reachable with astronomical observations, thus prohibiting any direct investigation of the evolution of the early stages after the *Big Bang*. This moment poses a minimum instant for direct observation of early universe, out of the inspection of remnant "relic radiation". The oldest electromagnetic radiation, the first being able to freely propagate right after the recombination of ions and electrons, is called Cosmic Microwave Background (CMB, CMBR) first time presented in this article [69] and it is an important source of data on the early stages of the Universe.

Heavy-ion experiments are able to investigate the "opaque" region before photo decoupling, by reproducing the QGP state in laboratory. To recreate such a state in heavy-ion collisions, the experiments must comply with some working conditions:

- the temperature has to be sufficiently high to de-confine the hadronic system.
- to be able to describe it with a thermodynamical model, the system has to be in thermal equilibrium. The equilibrium is reached through particle interactions, a sufficiently dense and interacting medium has to be produced during the collisions
- the scale of the system must be larger than the range of the strong interaction ($\sim$1 fm), in order to describe the evolution of the system with macroscopic observables taken from a thermodynamic model

Heavy-ion physics studies how collective phenomena and macroscopic properties, involving many degrees of freedom, emerge from the microscopic laws of elementary particle physics.

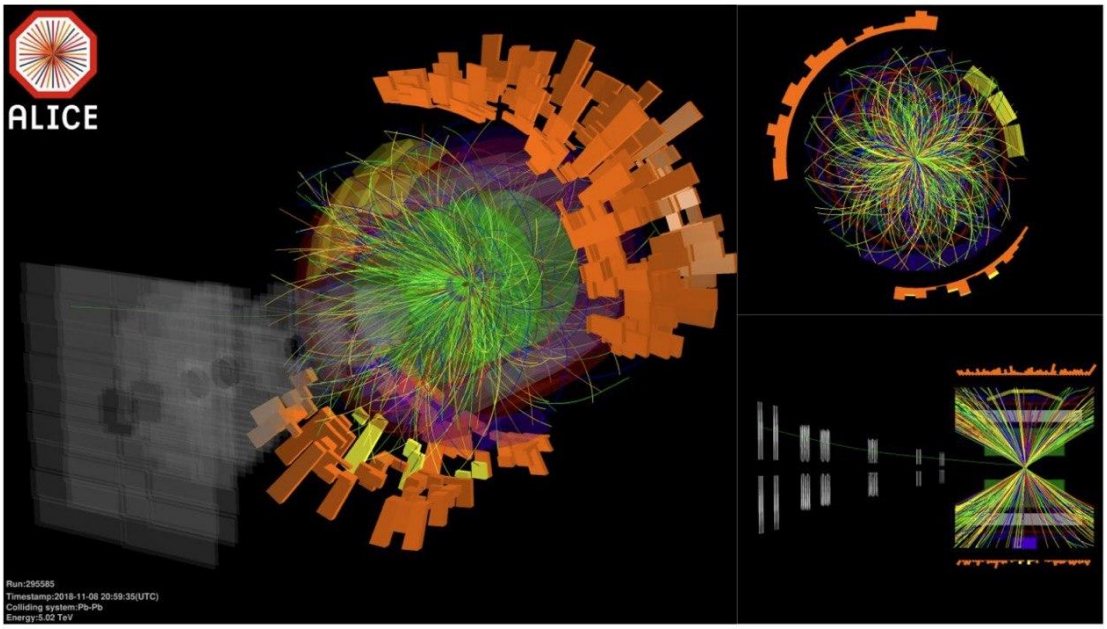


Figure 1.2: Pb–Pb event display from data taken at $\sqrt{s_{\mathrm{NN}}} = 5.02$TeV.

1.2 The Large Hadron Collider at CERN

The Large Hadron Collider is the last component of a longer acceleration apparatus. The whole accelerator is located at European Centre for Nuclear Research (CERN) near the city of Geneva, straddling the Franco-Swiss border. Each stage in the acceleration chain increases the energy of the protons or lead ions to the maximum achievable value for its setup, then the beam is redirected to the next acceleration segment in the line. Depending on the type of hadron to be accelerated, there are two sources: LINAC 2 for

the protons and LINAC 3 for lead ions. Resulting beams are then injected in the Proton Synchrotron Booster (PSB), which accelerates the hadrons to 1.4 GeV[i] and delivers the beam, now regrouped in bunches, to the Proton Synchrotron (PS). The latter then pushes ions up to 25 GeV into the Super Proton Synchrotron (SPS), until they reach the energy of 450 GeV before being injected into the LHC beam line. the aforementioned acceleration chain is depicted In figure 1.3, among all the other accelerators present on CERN site. The

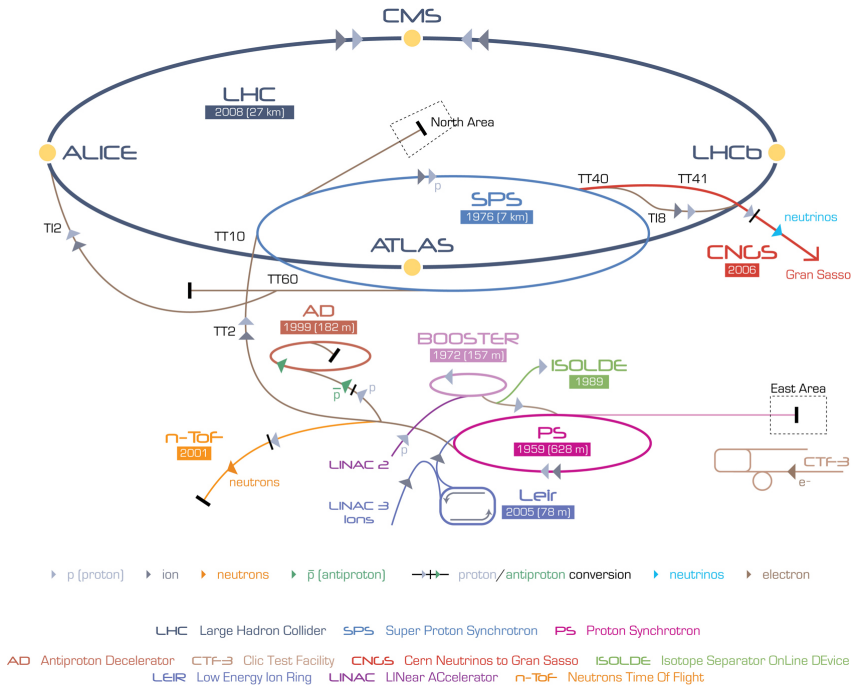


Figure 1.3: LHC experiments

counter–rotating beams at the LHC circulate in two separate *vacuum–filled* pipes, where they are accelerated up to the energy of 6.5 TeV in the case of protons. Along the ring there are four sites where beams are brought into collision in the correspondence with the major experiments. The highest centre-of-mass energies reached at the LHC in the collisions are 13 TeV and 5.02 TeV per nucleon pair for pp and Pb–Pb collisions respectively. Along with the energy, a relevant parameter for the experiments at the LHC is the *luminosity* of the beam. Being R the reaction rate for a process, it can be evaluated as a function of the luminosity L and the cross section of the specific process σ_p

$$R = L\sigma_p. \tag{1.1}$$

The instantaneous luminosity L delivered by a collider is measured with a procedure called "van der Meer" scan which is described by equation 1.2

$$L = \frac{N_b N^2 f_{rev} \gamma}{4\pi\varepsilon_n \beta^*} F, \tag{1.2}$$

[i] 1 elettronvolt (eV) $= 1.06 \times 10^{-19}$ J.

with N_b being the number of bunches in the accelerator ring, N the number of charges per bunch, f_{rev} is the revolution frequency of the beam and γ the relativistic factor. In the denominator ε_n is the normalised emittance[ii].

Considering the $\beta(s)$ function, which embodies some accelerator features such as the magnet configuration (the quadrupole configuration used for the beam focusing) and powering. It has a formulation as a function of the cross-sectional size of the bunch $\sigma(s)$ and the transverse emittance ε_n (which is constant and depends on the initial conditions) reported in equation 1.3

$$\beta(s) = \frac{\pi\sigma^2}{\varepsilon_n}. \tag{1.3}$$

The $\beta(s)$ can be represented as the distance from the focus point where the beam width $\sigma(s)$ is twice as wide as the focus point, for instance the interaction point. The β^* parameter if equation 1.2 is defined as the amplitude function $\beta(s)$ evaluated at the interaction point and describes the amplitude of the longitudinal trajectories of the particles in the beam bunches, where L is measured. In the experiments, the beam is squeezed as much as possible, to increase its transverse density thus the number of collisions per bunch crossing, so at a distance of beta before the focus point, the beam is also twice as wide. Of particular significance is the value of the amplitude function at the interaction point, β^*. Clearly one wants to be as small as possible; how small depends on the capability of the hardware to make a near-focus at the interaction point.

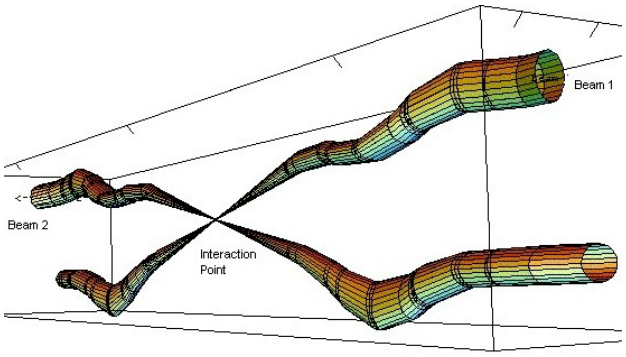


Figure 1.4: Representation of a typical evolution of the beam width in proximity of the interaction point, where the beam is squeezed to increase the number of collisions.

The geometrical factor F takes into account some physical specifications of the collider itself such as the relative inclination of the beams at the interaction point, which for LHC is 300 microradians, and parameters related to the transverse and longitudinal size of the beam. Typical numbers for the protons per bunch at the LHC can arrive at $\sim 10^{11}$ with the ring circulating up to 2808 bunches with a 25 ns spacing in between [43, 22]. The normalised emittance at the end of the acceleration is 3.75 μm rad whereas β^* depends on the beam focusing at the interaction point.

[ii] normalised emittance is defined as $\varepsilon_n = \beta\gamma\epsilon$, where β and γ here are the relativistic factors, ε quantifies spread of beam particles in the position-momentum phase space. The emittance is the spread of beam particles in the position-momentum phase space.

An important piece of information for the physics analyses at a collider experiment is the position where the collision between the two beams takes place, it is the so-called primary vertex. In the experiment the two well-focused beams cross each others in a limited spatial region, called *interaction diamond*. The centre of such a region is chosen as nominal position for the primary vertex, that is where collisions are mostly expected to happen. The coordinate frame of the experiment is therefore centered in such a point, for convenience. Because a bunch has finite size, the position of the primary vertex fluctuates around the nominal position. Being $\sigma^{bunch_{x,y,z}}$ the rms size of one bunch along transverse and longitudinal directions with respect the beam axis, assuming a gaussian dispersion of the bunch positions, the *rms* of the vertex fluctuations is

$$\sigma^{vertex}_{x,y,z} = \frac{\sigma^{bunch}_{x,y,z}}{\sqrt{2}}, \tag{1.4}$$

where the *rms* size of the bunch is a function of β^*:

$$\sigma^{bunch}_{x,y,z} = \sqrt{\frac{\varepsilon_{x,y,z}\beta^*}{\sqrt{\pi}}}. \tag{1.5}$$

Typical values in *pp* collisions at the IP2, where ALICE apparatus is installed, are $\sigma^{vertex}_{x,y} \sim 50\mu$m and $\sigma^{vertex}_{x} \sim 5$cm.

Other two accessory quantities that is useful to define are *rapiditiy* and *pseudo-rapidity* of yielded particles form a collision. The former is related to the Lorentz linear transformation connecting positional quantities across two reference frames in relative motion. The transformation is represented by the matrix equation in 1.6

$$\begin{pmatrix} ct' \\ x' \end{pmatrix} = \begin{pmatrix} \cosh y & -\sinh y \\ -\sinh y & \cosh y \end{pmatrix} \begin{pmatrix} ct \\ x \end{pmatrix}, \tag{1.6}$$

With c being the speed of light and x, t the space-time coordinates. The rapidity for a single particle y can be expressed as a function of the energy E and its the z component of its momentum p_z in the equation 1.7

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{1.7}$$

To compute this quantity it is therefore required full information about the particle momentum and mass, since to know the energy it is necessary the mass of the particle, hence the type. This is not easy on most of the cases, hence the latter quantity is used, instead since it is easier to compute experimentally . The pseudorapdity is then defined in equation 1.8

$$\eta = \frac{1}{2} \ln \left(\frac{|p| + p_z}{|p| - p_z} \right) = -\ln \left[\tan \left(\frac{\theta}{2} \right) \right], \tag{1.8}$$

with the angle θ formed by the particle trajectory and the beam axis.

1.2.1 The ALICE Run 3 programme

Starting from 2021, the LHC will start its third data taking period, the so-called Run 3. The accelerator will provide higher performance compared to the Run 2, reaching the

maximum energy of $\sqrt{s_{\rm NN}} = 5.5$ TeV in Pb–Pb collisions and $\sqrt{s} = 14$ TeV in *pp* collisions. All of the 4 experiments will run with an updated setup, aiming to increase their physics detection capabilities in terms of resolution and readout. ALICE is undergoing to major upgrades, with the substitution of the Inner Tracking System (ITS), its innermost detector, the closest to the beam pipe, and the addition of a forward-rapidity detector for muons, the Muon Forward Tracker (MFT). These major and other upgrades will improve the overall capabilities of the experiment, allowing physicists of the collaboration for more precise measurements of the QGP. Improvements in the measurements are driven by increasing the size of the collected data sample, as it will give access to the very rare physics channels needed to understand the dynamics of QGP. Converted in integrated luminosity, the objective is to collect of the order of 10 nb^{-1} [iii] in Pb–Pb collisions at $\sqrt{s_{\rm NN}} = 5.5$TeV , considering an instantaneous luminosity of 6×10^{27} cm^{-2} s^{-1}. It represents the most challenging requirement to achieve the proposed physics programme. The new ALICE setup will be able to inspect 50 kHz Pb–Pb collisions with the minimum possible bias. All of the detectors input data will have to be transmitted to the online readout systems in a continuous fashion. Proton-proton collisions will be collected by the experimental setup at 400kHz rate.

1.3 The ALICE experimental setup in Run 3

Since the beginning of the Long Shutdown 2 (LS2), (2019-2020) ALICE is undergoing a major upgrade of the critical components that will allow the experimental apparatus to cope with the enhanced luminosity delivered by LHC. The planned upgrades will preserve the excellent particle identification capability of the detector but also permit physicists to accumulate 10 nb^{-1} of Pb–Pb collisions inspecting about 10^{11} interactions [17, 16]. Moreover, the upgrade will improve the resolution, that is the precision in the measurement of the position of the primary and secondary vertices identification. While the primary reconstructed vertex is the interaction point, where the primary collisions take place, as per *secondary vertex* it is intended the displaced points in space where single particles decay, originating other particles. Depending on the nature of the particle decaying and the type of decay, secondary vertices are displaced from the primary one, and show different topologies of yielded particles. the low-momentum vertexing and tracking capabilities This is the minimum needed to address the proposed physics programme that focuses on rare probes both at low and high transverse momentum. In figure 1.5 it is reported a dump of the internal structure of the ALICE experimental apparatus as it will be after the upgrading phase at the end of LS2. Physics analyses for heavy-ion physics require high large integrated data samples for Pb–Pb measurements but need also to be compared with precise data acquired during run of *pp* and p–Pb collisions. This is mainly because they are used as references to be compared with, to highlight different behaviour between the three different collision systems. Therefore, the size of a *pp* reference with a statistical significance comparable to the Pb–Pb data size is estimated to be about 6 pb^{-1}, corresponding to a data taking of *pp* collisions with event readout rate of 200 kHz over a period of a few

[iii] 1 b (Barn) = 10^{-28}m^2

months.

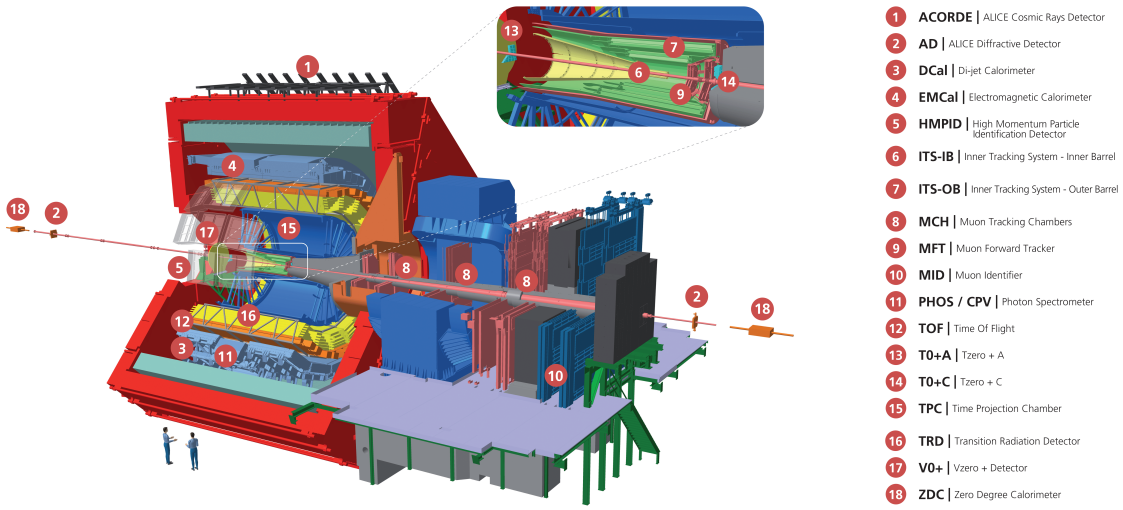


Figure 1.5: ALICE apparatus layout with upgrades in Run 3

1.3.1 Overview of the ALICE detectors and their upgrades

In this section will be presented a brief overview of the detector upgrades that will be included in the ALICE setup for the Run 3. Highlights will be given to the changes with respect to the setup used during Run 2. Finally, particular attention will be dedicated to the Inner Tracking System (ITS) detector, whose upgrade motivated the development of this work. ALICE experimental layout for Run 3 foresees 18 detectors, each one serves the measurement of physical observables using different technologies. A brief count of the detectors is presented below, with highlights to the upgraded parts. The volume of the detector that covers a an angle corresponding to a pseudorapdity interval $|\eta| < 0.9$ is also called **central barrel** and it is composed by many elements which are enumerated hereafter together with a brief description.

- The **beampipe** is not an actual detector, it serves to contain the region where each beam travels keeping the vacuum along all the trajectory and in the collision point. It will have a smaller radius, moving from 29.8 mm of the Run 2 to 19.88 mm, letting the innermost layer of the ITS to be placed closer to the interaction point.

- The **Inner Tracking System** (ITS), is the innermost detector, the closest to the beampipe. The one used in Run 2 will be replaced by a new high-resolution, high-granularity and low material budget one covering the mid-rapidity region ($|y| < 0.9$)[16]. It is composed by seven cylindrical concentric silicon pixel layers that will be described in section 1.4.

- The brand new **Muon Forward Tracker** will be placed between the ITS and the T0+C, and will drastically improve the measurement done with the Muon Spectrometer during Run 2, also allowing for new others that were not possible before [82].

- The **High-Momentum Particle Identification** (HMPID) is built with seven modules of Cherenkov counters coupled to Multi-Wire Pad Chambers (MWPC) equipped with CsI photocathodes [62, 73].
- The **Fast Interaction Trigger** (FIT) is the integration of the upgraded versions of two currently present detectors: the **V0-Plus** and **T0-Plus**. Current T0 detector consists of two arrays (T0-A and T0-C) of Cherenkov counters with the interaction point in–between. The distance from the IP to T0-C is 70cm while on the opposite side the distance from T0-A to the IP is about 3.6m. The T0 detector is the only ALICE sub-detector capable of delivering high-precision start signal for the TOF detector. Current V0 detector is a small angle detector made if two array systems on both sides of the ALICE interaction vertex. It is mainly a minimum bias trigger, a centrality indicator and a luminosity control. Both of the upgrades aims at complying with new data-acquisition requirements[37, 20, 26].
- The **Time Projection Chamber** is the main charged particle tracking and particle identification (PID) detector of ALICE. To operate in continuous readout (CR) at a rate of 50kHz in Pb–Pb, the Multi-Wired Proportional Chambers (MWPCs) at the caps of the detector will be replaced by Gas Electron Multiplier (GEM) detectors that exhibit excellent readout rate capabilities. Together with new readout electronic it will the expected data taking conditions[59].
- The **Transition Radiation Detector** (TRD) is used to identify electrons with momentum higher than 1 GeV/c. It will upgrade its readout electronics to be able to run at higher frequencies [19].
- The **Time Of Flight** detector identifies charged particles in a momentum range of $0.2 \div 2.5$ GeV/c. It will have an upgrade in the frontend electronics [44].
- The **ElectroMagnetic Calorimeter** (EMCal) enhances the ALICE performance in the study of jet quenching in Pb–Pb collisions. It reveals direct photons, jets and electrons from heavy-flavours decays [4].
- The **Di-jet Calorimeter** (DCal) is *de-facto* an extension of the EMCal, to enhance the jet quenching measurements [11].
- The **PHOton Spectrometer** (PHOS) has been designed to measure photons, distinguishing those directly yielded in the collision from those coming from the decay of π^0 and η. It is integrated with another detector the **Charged-Particle Veto** (CPV), a MWPC with a cathode pad readout that serves as suppressor for the charged-particle background of the direct photon sample. It also will benefit from an upgraded frontend electronics same as the HMPID, providing performance up to 50kHz [73].
- The **magnet** used in ALICE was built for the L3[8] experiment at the Large Electron-Proton (LEP)[77] collider and surrounds central barrel detectors. The trade-off between the necessity to bend high-momentum particles and at the same time to resolve low p_{T} particles in the TPC, led to the choice of a solenoidal magnetic field of $0.2 \leqslant \mathcal{B} \leqslant 0.5$T. The magnet is "hot" in the sense it is not required to exploit superconductivity (obtained with cold magnets at temperatures $\sim 0K$)to achieve the operational field values.

Moving to the **muon arm**, the forward section covering the $-4 \leqslant \eta \leqslant -2.5$ acceptance region, the detectors that will be employed in run 3 are listed below.

- The **Muon Tracking Chambers** (MCH) will be upgraded in their frontend electronics designed to cope with a 100kHz readout frequency, the technology of the new frontend has been developed together with the TPC upgrade project [74]
- The **Muon Identifier** (MID) is an upgrade of the former muon trigger chambers. They have been improved to operate with a continuous readout mode. The same type of Resistive Plate Chambers will be used with lower gain and thanks to the new frontend chips signal will be later amplified [74]

and very-forward or peripheral detectors:

- the **Alice Diffractive Detector** (AD) consists of two stations made of two layers of scintillator pads, one station on each side of the interaction point. With the AD substantially increases the ALICE forward physics coverage [83]
- the **Zero Degree Calorimeter** (ZDC) gives information about the centrality in Pb–Pb collisions, measuring the energy of the nucleons not involved in the collision, called *spectators* [39]

1.3.2 Heavy-ion collisions in ALICE

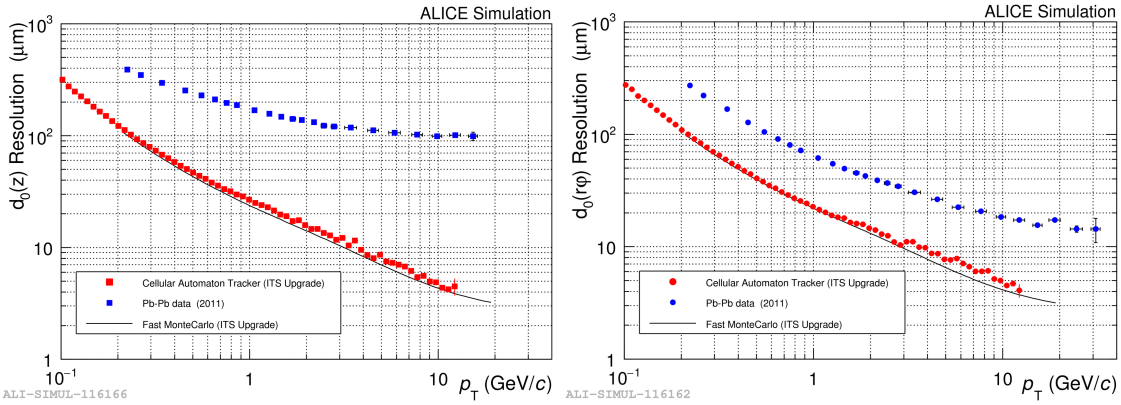
This section will be dedicated to define some of the concepts about heavy-ion collisions that will somehow be involved in the development of this work.

The ALICE detector reference system To refer something within the ALICE geometry a common reference right-handed orthogonal Cartesian system frame is adopted. The origin $O(x, y, z) = 0{,}0{,}0$ is assumed as the nominal position of the interaction point. The x axis is coplanar with the plane on which the LHC ring lays, and orthogonal to the beam direction, pointing to the centre of the accelerator. The y axis is vertical, pointing upwards. The z axis is parallel to the beam direction, with orientation determined by the chirality of the coordinate system. Central barrel orientates its cylindrical symmetry parallel to the z axis. The plane spanned by x and y, orthogonal to the beam line, is called *transverse plane*. Particles yielded during the collisions also have uniform distribution of their ϕ coordinate, showing an azimuthal symmetry on the transverse plane. A visualisation of a single Pb–Pb collision at $\sqrt{s_{\mathrm{NN}}} = 5.02$ TeV is reported in figure 1.2.

1.4 Upgrade of the ALICE Inner Tracking System

The ITS is one of the pivotal elements of the new experimental setup for Run 3. The detector will have ultra-light design, with high-resolution to play a crucial role improving the resolution on the position of primary vertex of a collision. It will also increase the precision in determining the distance of closest approach (DCA) of particle tracks from the interaction vertex and will allow for efficiently tracking them also at low p_{T} ($< 1\mathrm{GeV}/c$). In figures 1.6a and 1.6b are reported preliminary studies compared also with prediction done

in the ITS technical design report [16]. Comparisons are done with performance obtained with the former ITS used in Run 1-2. Moreover, it will be done operating on a continuous stream of data, delivered by the continuous readout and minimum bias trigger in data acquisition.



(a) z direction (b) xy plane

Figure 1.6: Impact parameter resolution along the z direction and on the transverse, xy, plane for former ITS apparatus vs upgraded as a function of p_{T}. Data in black is taken from a 2011 period, the same used also in the TDR for preliminary studies. the points for the upgrade setup have been computed using a Fast Monte Carlo tool (black) and the Cellular Automaton tracking algorithm preview for the upgrade.

Key features of the new ITS are notable both from the readout and the performance sides. It will support a readout rate greater than the required 400 kHz in pp collisions and 100 kHz in Pb–Pb as per design requirements. It will grant an improvement on the impact parameter resolution by a factor $\simeq 3$ in rφ direction and $\simeq 5$ on the z direction with a $p_{\mathrm{T}} = 500\mathrm{MeV}/c$ (low p_{T}). Finally, the improvement of the tracking efficiency and of the p_{T} resolution at low transverse momentum is possible thanks to the increased granularity for space points.

The aforementioned great improvements are possible thanks to:

- lower material budget that will go down to the 0.3% of X_0 for the inner layers and $\simeq 1\%$ for the outer layers
- smaller pixel size compared with the ones in the former ITS: the size is now $27\times29\mu\mathrm{m}^2$, compared to the $50 \times 425\mu\mathrm{m}^2$ of the Silicon Pixel Detectors (SPDs) used in Run 2.
- the first layer is closer to the interaction vertex than before: only 22 mm instead of 39 mm
- seven layers in the ITS layout instead of six

1.4.1 Layout of the detector

The new ITS (depicted in figure 1.7) is equipped with seven layers of silicon pixels detecting surface. The average radii of the layers are 22 mm, 31 mm and 39 mm and 194 mm, 247 mm, 353 mm and 405 mm, respectively. In figures 2.3 are reported more more precise measures also fo minimum and maximum radius for each layer. Each layer is made out of adjacent staves combined to form the cylindrical geometry. In the Inner Barrel (IB)

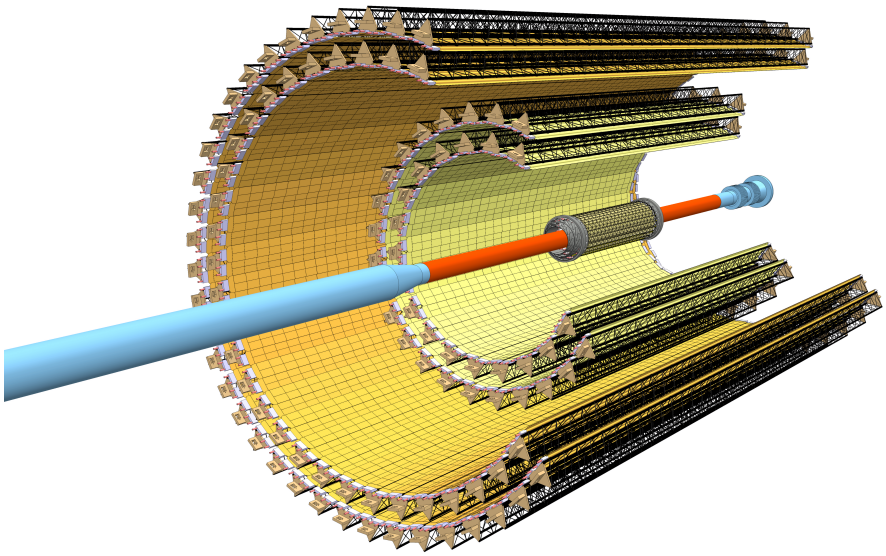


Figure 1.7: Layout of the new ITS detector. In red the size-reduced beampipe with the three layers of the inner barrel Inner Barrel. Two Middle Layers (ML) and two

the staves are slightly superimposed each other in an "oriented" way; this arrangement is also called *turbo geometry*. It ensures a full coverage of the whole the azimuthal angle, also in the junction areas between the inactive borders of the staves. In figure 2.3 two cross-sections of the inner and the outer barrel are reported, the schema on the left shows its turbo geometry. This detail is particularly relevant for the sake of the work presented in this thesis. Having an overlap of active regions will produce a duplication of the measured signal from chips. A particle crossing the overlap region of two staves will possibly impress pixels from two staves which will record the single passage twice. This detail is certainly included in the geometry of the simulated detector hence in the data used to develop the vertex finder algorithm. In figure 4.4 this effect is well shown as each of the three layers used for the reconstruction presents these duplications. It is also interesting to investigate the material budget distribution of the detector, that is the "sequence" of kind of material crossed by a particle travelling through the detector. The azimuthal distribution of the material (plot on the right) corresponding to the arrangement of three staves (left) of the IB is shown in figure 1.9. These different materials are included in the geometry information related to the simulated detector, that will be later used as conditional data to generate monte carlo data. The transport code will simulate the interaction of each single particle, depending on its direction, with the corresponding material distribution encountered on its trajectory. The detection of the passage of a particle across the staves is performed by

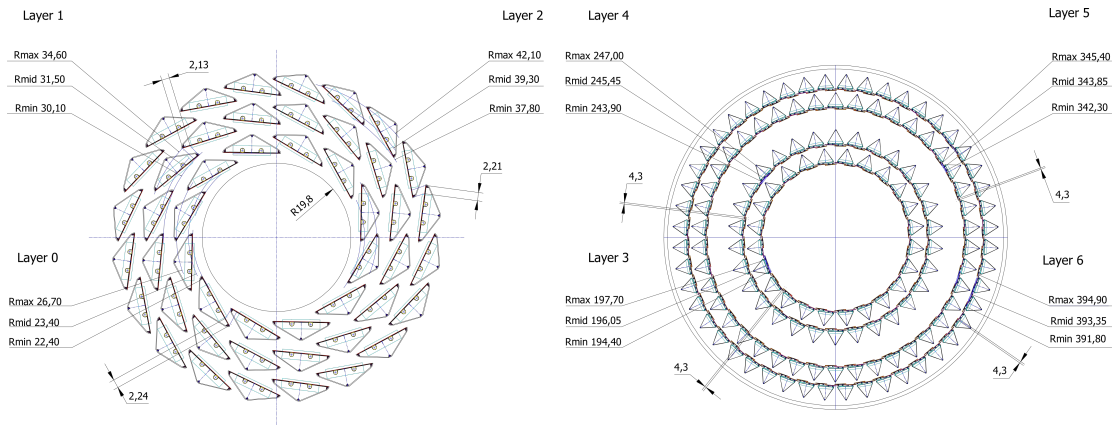


Figure 1.8: View of the cross-section of the Inner Barrel (left) and Outer Barrel (right) layers. The active region is in correspondence of the basis of the triangles. other segments are the supportive carbon structure of each stave.

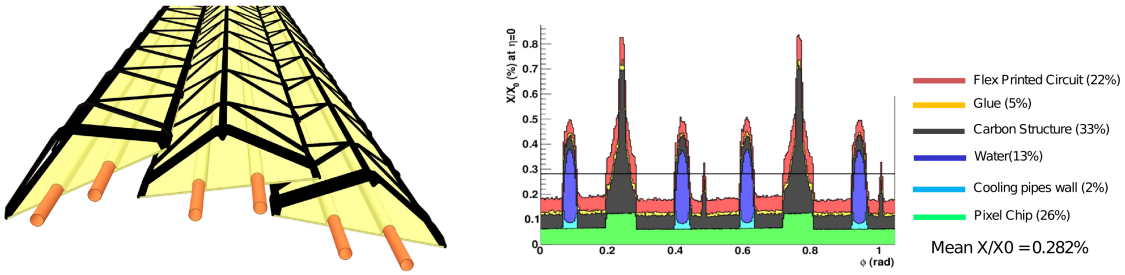


Figure 1.9: Overlap between staves of the Inner Layers (left) and the corresponding material budget distribution (right). Peaks correspond to the overlap of the reinforced structures at the edges of the Space Frame, while the narrow spikes to the reinforcement at the upper vertex. The peaks around 0.5% X_0 are due to the polyimide cooling pipes filled of water.

Monolithic Active Pixel Sensors (MAPS), called ALice Pixel DEtector (ALPIDE)[9].

Depending on the considered layer, the size of the stave is variable, hence their setup differ as it can be seen in figure 1.10.

Each stave is composed by:

- a Space Frame: ultra-light (heaviest, the ones in the outer barrel, are $\simeq$ 80g) support structure made with carbon fiber
- a Cold Plate: a plate made out of carbon fiber, embedding two polyamide cooling pipes with an internal diameter of 2.05 mm. The cooling system maintains the chip temperature below 30°
- a Hybrid Integrated Circuit or Module: it regroups from 9 in the IB staves up to 14 ALPIDE chips in the OB ones. The chips are placed on a Flexible Printed Circuit (FPC) responsible for powering them, but also to deliver clock signal, control and data signals

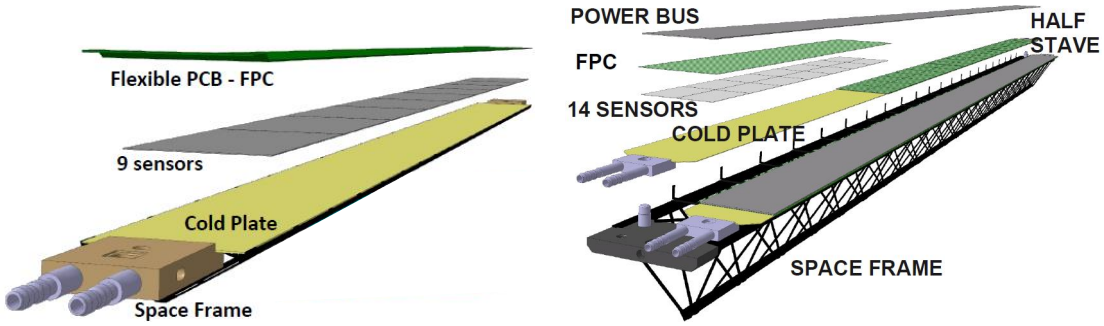


Figure 1.10: Overview of the components of staves of Inner Barrel (left) and Outer Barrel (right).

- a Power Bus for the staves on the OB. For IB staves the power supply is embedded in the FPC

In the Outer Barrel, each stave is further divided in two half staves. The size of the staves depends on the layer we are considering: for the Inner Barrel the total length is ~ 27 cm, whereas on the Outer Barrel sizes go from ~ 80 cm for the middle layers up to ~ 150 for the outer one. In the IB of the ITS there are in total 48 staves, 12 on the first layer, 16 on the second and 20 on the third. In the OB a total of 54 (30+24) for the middle layers and 90(42+48) for the outer layer.

1.4.2 The ALPIDE monolithic chip

The ALPIDE chip is the core technology of the new ITS. It has been developed to fulfill the stringent requirements of the upgrade. It is a Complementary Metal Oxide Semiconductor (CMOS) monolithic active pixel sensor that contains a matrix of 512×1024 pixels, with a pixel dimension of about $27 \times 29 \mu m^2$. It is equipped with in-pixel amplification, shaping, discrimination and multi-event buffering. In figure 1.11 there is a schema of the cross-section of a CMOS produced with the *TowerJazz* imaging process at 0.18 μm. Such a procedure can provide up to six metal layers, which allow for a high density circuitry at low-power consumption. It is radiation-tolerant for the running conditions in which it will be used, thanks to the presence of a gate oxide having a thickness of 3 nm. The presence of p-wells shields the n-well from the electrons produced by a particle crossing the pixel. This allows only the n-well diode to collect the produced charges without any loss. The presence of the deep p-well allows the entire CMOS circuitry to be deployed directly within the volume of the sensor (monolithic). The epitaxial layer P- constitutes the active volume of the detector. A moderate reverse substrate bias voltage is applied in a range between -6 V and 0 V. This is crucial to increase the output signal of the collection n-well diode and it may also improves the resistance to non-ionizing irradiation effects. The white area between the diode and the epitaxial region is a *depletion region*. Electrons produced in the process perceive a small electric field, also enhanced by the reverse bias voltage applied, and start drifting toward che collection point. Typically, the charge is not collected by one single diode which corresponds to a single pixel, but it is spread also on surrounding ones, forming a *clusters*.

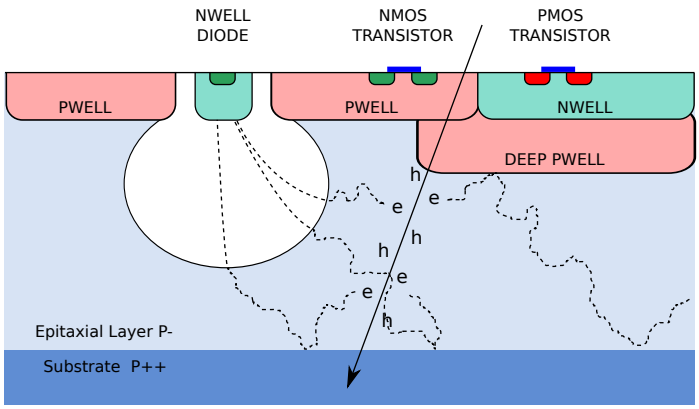


Figure 1.11: Cross-section of the ALPIDE CMOS pixel unit.

Chapter 2

The ALICE Online-Offline software programme for Run 3

The goal of the hardware and software upgrades of the ALICE apparatus is to readout and inspect Pb–Pb collisions up to the rate of 50kHz, sampling the *pp* and p–Pb at up to 1MHz. Not only the tracking precision of the experiment will be improved by the hardware upgrades mentioned in previous chapter, but also the, switch to the continuos acquisition of data will enable the collection and inspection of larger data samples in terms of physical collisions processed. For instance, the amount of heavy ion events recorded is estimated to increase up one hundred times more than the one acquired during the Run 1. A reduction of the data volume will be operated by processing them *on-the-fly*: during the data collection.

2.1 Data life cycle and upgraded computing system

On the contrary to most other high energy physics experiments, neither a high-level trigger, nor an event-filter farm will be employed to steer the ALICE data acquisition in Run 3. In fact, the upgrade on the ALICE computing system foresees to transfer data from all the detectors to an online processing computing system, embodying a so-called **continuous readout** paradigm for data readout. Such a system will be responsible for processing a stream of information, applying a partial online calibration and reconstruction, to replace most (>90%) of the raw inputs with compressed and reconstructed data. This phase will be instrumental to work with the aimed acquisition rates thus having a data throughput adequate to the available computing resources *ad-hoc* deployed in a dedicated facility residing in proximity of the experimental site.

After the acquisition from the frontend electronics of the detectors, the data stream enters a processing pipeline that begins with the *First Level Processors* (FLPs), and finishes with processed data being written on permanent storage. Every operation or post-processing that will have as source the stored data has to be considered as an *offline* operation. ALICE is developing a brand new software stack, a logical *umbrella* to supply all the require computing tasks that will possibly arise along all the Run 3 duration. The same software stack will be adopted both for online and offline data processing, reconstruction

and analysis.

It is named after O^2, standing for Online-Offline (framework) to highlight its versatility, flexibility and universality towards each of the ALICE future computing needs. Concerning its usage on the online "fashion", its structure foresees a functional flow including a sequence of stages that aim at reducing the output data bandwidth of a factor greater than 300 comparing the instant they are read to the one they are stored permanently on magnetic tape.

The data links from detector electronics will deliver information either in a continuous fashion or in a minimum-bias triggered mode. As anticipated some detectors will not work in continuous mode, albeit their acquisition rate has been improved with upgrades in LS2. Specifically, all the detectors that will operate in continuous readout will benefit of the new common readout units (CRUs) operated by Field Programmable Gate Arrays (FPGAs) installed on tightly coupled acquisition boards linked to high speed network. Exception is therefore done for EMCal, PHOS and HMPID, that will continue to operate in fast-paced triggered mode. The signals are then sent to readout nodes that will achieve a global **data reduction** by means of compression and filtering. Specific time markers are used to cut the data flows into more manageable fractions called Time Frames (TF). The process of writing a time frame starts from its sub-components, finer time slices that are aggregated after being processed. The clock signal delivered by the LHC is used as a reference to synchronise, aggregate and buffer the information. The process receives in input the "HeartBeat Frames" (HBFs): data slices corresponding to integrated information corresponding to the duration one orbit from each input link, whose duration is roughly 90μs. Data are then aggregated, it is performed a quality control and are then organised into "sub-timeframes" (STFs) that are collections of a hundreds (~ 256) of HBFs on each input node, with a duration of $20ms$. Calibration and detector-specific pattern recognition is carried out with a high degree of parallelism on multiple devices, thanks to the local and independent nature of the data. The data throughput at this point int the pipeline is already reduced from ~ 3.5TB/s down to 600GB/s.

A second round of data aggregation is performed on the computers of the Event Processing Nodes (EPNs) farm, to combine the data from all the detector inputs and assemble the *timeframes*. The first **Synchronous reconstruction** (SR) is then run online. It is important to highlight that all of the processing, up to the SR included, are carried out during the data taking. The synchronous reconstruction performs a partial reconstruction of data related only to few detectors such as the ITS, TPC, TRD. Full TPC tracking is done in synchronous mode to achieve compression and data "skimming" from all the unneeded information left behind by the reconstruction process. Through the reconstruction of each tracks, the trajectories followed by particles through the TPC, it is therefore possible to apply selections that exclude electromagnetic noise. Its main component is represented by low-momentum secondary electrons (yielded by secondary decay processes) that "coils" through the gas detector following helicoidal patterns with small radii, compared with the scale of the TPC, because of the presence of the magnetic field. They are also called *loopers* and are subtracted from the signal since they are not relevant for the physical measurements, reducing the final size of the output. This is the most computationally-intensive and most relevant for the data volume. Raw data from TPC are never stored because of their memory weight, they are instead replaced by the results of cluster finding routine.

Results of the synchronous processing temporarily stored on a **disk buffer** with a size

of approximately 20 PB. If during the data taking the size of temporary stored data exceeds a threshold part of them is flushed on the permanent storage at the *Tier 0*, the largest data centre present at CERN. Finally, during technical stops, scheduled periods of inactivity of the experiment, the **Asynchronous Reconstruction** (AR) is performed. It involves a full reconstruction with final calibration and is performed on the partially reconstructed data stored in the temporary buffer at the EPN centre. This final step may also use computing resources from the Grid[25] to absorb the peak needs beyond the capacity of the online cluster. Results are then written on permanent storage at a rate of less than 100GB/s. The reconstructed events will then be available for analysis on the Grid.

In figure 2.1 it is represented a schema summarising the described phases of the data reconstruction life cycle with compression and reduction.

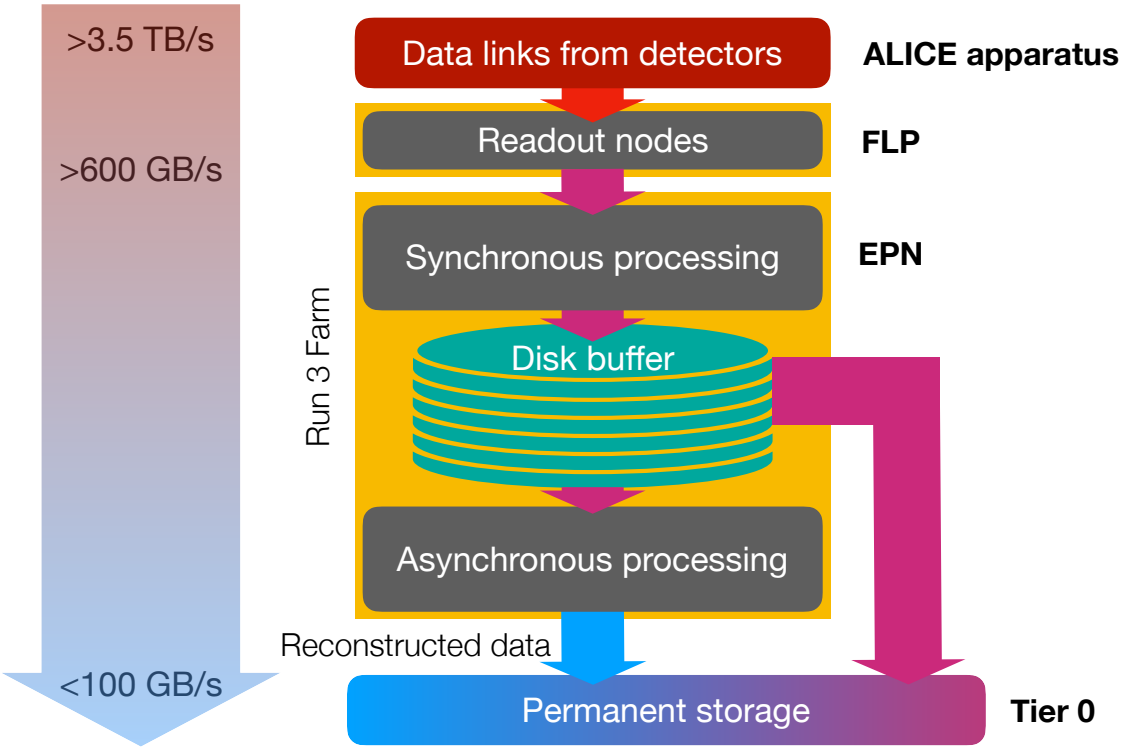


Figure 2.1: Schema of the main phases of the data processing. From raw data of detector links to the permanent storage after the reconstruction process, data reduction factor is ≈ 350.

2.1.1 The EPN cluster

The ALICE dedicated data centre for the synchronous and asynchronous processing will be a computing cluster located on the surface of the experimental area at *Point 2* at

the LHC. The so-called O^2 facility, will be interfaced to the Grid and to the CERN Tier 0[i] permanent storage the same way is used for data processing in other computing centres. It will not be an isolated computing farm with outbound-only connectivity, as it was the High Level Trigger (HLT)[56] farm in the past. As it will be illustrated more in detail in section 2.5, the O^2 reconstruction chain will rely on runtime acceleration provided by Graphic Processing Units (GPUs). Consistent part of its codebase is written to exploit the high throughput computing power provided by means of offloading large pieces of executions on them. Therefore, nodes on the EPN farm will be equipped with graphics cards.

Albeit definitive decision on the hardware specifications is not yet taken at the time of this work is written, some setups are being currently evaluated. Commonly agreed layout foresees, pending the final results, a system built up with at least 250 computing nodes, possibly double sockets for CPUs, and equipped with 8 GPUs, for an approximately total of 500 CPUs and 2000 GPUs[ii].

2.2 The Online-Offline computing framework: O^2

Upgrades in the hardware section of the experiment always involve some refurbishment of the software used to elaborate information and cope with new physics requirements implied by new readout strategies. In the specific case of the ALICE upgrade for Run 3, a complete replacement of the old codebase takes place to address the brand new paradigm of the continuous readout together with the replacement of the ITS, addition of the new Muon Forward Tracker (MFT) and global upgrades for many of the other detectors included in the apparatus.

First relevant feature, before even considering anything related to data transmission and processing, is the unification of the *online* and *offline* codebase under the same project: the ALICE Online-Offline (O^2). Such a merge correspond in having a codebase stored in one single place, centrally managed. There the whole software quality is maintained through automatic tests, specifically:

- **unit tests**: to verify that the minimum piece of code that can be tested is works upon any change from incoming later contributions
- **functional tests**: to test at a slightly higher level than unit tests, they grant that given fixed input to tested functions, the output is preserved being consistent upon code evolution
- **performance tests**: to measure behaviour of the code in terms of agreed definition of accuracy, efficiency but also speed in performing the computation. Tests fail when a certain threshold is exceeded or not reached

[i] more details on the tiered structure of the LHC computing model are presented in appendix A, section A.1.1.

[ii] These esteems have to be assumed as reasonable in term of scale: are not yet final, numbers can slightly change in future.

- **regression tests**: to surveil that after every upgrade or fix in the code the functional and performance tests produce the same level of quality. If not it is called a regression and may happen also in case the performance tests requirements are fulfilled

However, it will be hereafter described how the integration of the two faces of the data processing si something more rooted in the structure of the framework itself than to share tests.

The unification of the two aspects of the data processing of the experiment literally means that the software that performs online *synchronous* and *asynchronous* reconstruction is the same. In general, whenever it is possible there are many advantages in having a single interface. One among others is the code consistency: no differentiation between the utilised online and rerun offline ensures more consistency for the obtained results. In addition, there is only one version of the code to maintain and every improvement, fix, further addition done later in subsequent releases is immediately propagated to every context where the code is used, keeping an overall consistency across every possible implementation of the stack.

Into another perspective it is also interesting to analyse the context in which the implemented O^2 version is run. Reconstruction software dedicated to the online preprocessing and reconstruction is easier to maintain aligned with its underlying architecture and production environment, since the latter is hardly like liable to evolve very much over the duration of the Run 3, a part from maintenance operations that are likely to replace broken components with same piece of technology. In any case each operation is performed in a "controlled" environment where hardware and network capabilities, software pre-requisites and operating system are well known and uniformed by definition. On the one hand, in real life cases some specific tuning and arrangement in the software is feasible and rather easy, since it only requires to run on one known architecture for the online production. On the other hand, the software stack is shared across the EPN farm and any other affiliated data centre that aims at processing ALICE data (either reconstruction of the exceeding data not processed by EPN) thus requiring more flexibility and adaptation to the different hardware setup and configuration that possibly is much different in its layout.

A high degree of flexibility is required by the usage of GPUs, since different vendors on the market are bound to their languages and frameworks, posing great challenges with respect to a generalisation of the source code, heavily affecting the portability. The data centres may own different setups of computers equipped with different graphic cards brands with respect to the O^2 facility, also because usually they are not built on purpose to serve only ALICE computational use cases. It is therefore critical to develop a platform that is portable and flexible enough to be portable on heterogeneous architectures producing consistent results. That is why high-granularity tests are mission critical, since they ensure that despite the underlying computing units and accelerators running each O^2 task, the results are consistently comparable and used homogeneously.

There are two fundamental technologies that fulfill the demand for scalability, flexibility and portability of the O^2 framework: **ALFA**[79] and the O^2 **Data Processing Layer (DPL)**. The first is the fundamental one, the latter is built mostly on top of the first and is instrumental to specify the ALICE use case on top of the more general-purpose tool ALFA. Following sections will describe more into details these two pieces of software.

2.2.1 ALFA: the fundamental component for new simulation and reconstruction paradigm

The commonalities between ALICE and the Facility for Antiproton and Ion Research (FAIR)[47] in terms of computing requirements, led to a joint development of a common experiment-independent fundamental software substrate on which plug many known simulation and reconstruction frameworks but also brand new component to be used in high-energy physics experiments. The result is the ALICE-FAIR (ALFA) framework, developed by the FairRoot[81] and the O^2 teams. ALFA embodies a data-flow based model for data processing powered by message queues and multiprocessing. It is the core of essentially any workflow such physics simulation, reconstruction, and analysis in their implementation in Run 3 software. It is conceived for horizontal scaling [iii] and is capable of communicating with heterogeneous hardware, included computing accelerators. It has also supports different computing languages for integration and usage.

Each of the three aforementioned computing cases are so-called data-driven workloads and can be seen as processes around a flux of data that are processed and possibly changes during their processing. Either data are generated or taken as input and passed to a *pipeline* of stages where, depending on the kind of workflow and also on the topology of the workflow, will eventually produce some final objects (simulated data, reconstructed physical entities, analysis results). To enable all these scenarios with a unique interface, ALFA provides a *data transport layer* able to coordinate multiple entities for the data processing assembling the topology corresponding to the needed workflow. Such a layer is empowered by FairMQ[80]: an intelligent mechanism for message passing that supports for different communication mechanisms, relying on *ZeroMQ*[89] and *nanomsg*[65], shared memory transport via *Boost*[27]. It is intelligent in the sense that it is able to automatically deploy an abstract topology of objects communicating each other by optimising the structure of the inter communication graphs and also to transparently cope with different and heterogeneous deployment scenarios using the same description of the abstract workflow. For instance, shared memory via Boost is used for messaging when the communication happens within the same computing node, whereas nanomsg and ZeroMQ are used for inter-node communication[iv] also exploiting the Remote Direct Memory Access (RDMA), a technologies that allows different processes running on separate computing units to share the same logical description of the memory pool, *de-facto* sharing the same memory pages within the scope of two different running programs. On the side of the objects that do actual computation on the data stream, the fundamental building block for ALFA is an extension of the concept of *device*, directly inherited from FairMQ and enriched of features useful for ALFA specific use. ALFA devices are divided in three main classes:

- **Source**: it is a device that does not require any input. For instance, a sampler used to feed the *pipeline* (task topology) with data from files, is a source. It can be very useful to simulate the data stream coming from detectors data links, the CRUs.

[iii] i.e: start and plug to the topology new instances, either on the same node or a newly added node, to meet computing and throughput demands

[iv] communication across different computers, therefore between different instances of the program

- **Message-based Processor**: it is a device that operates on messages without interpreting their content in decision making. It is agnostic of the payload of the message itself, can be used for more general tasks, like data duplication (to be used at the same time by two different devices).
- **Content-based Processor**: it is a device that accesses the actual payload of a message, it is the skeleton of every user-based algorithm that is run on the input data.

ALFA is also integrated with the FairRoot module. FairRoot is an object oriented framework for simulation, reconstruction and data analysis based on the ROOT[18] framework from CERN. It includes core services for detector simulation and offline analysis of article physics data. The data simulation in the O^2 is purely implemented based on the ALFA substrate.

2.2.2 The O^2 data model

The O^2 *Data Model* (Data Model), is an ALICE-specific description of the messages being exchanged by the various devices. It means that the general-purpose structure provided by the ALFA layer is here specialised to address the requirements from the experiment computing model. The Data Model, as a result shows three crucial features. The first is that it is language agnostic, in the sense it is an abstract interface to describe quantities, then there can be multiple different software and libraries to support the description and their implementation. It is extensible allowing to deploy a further layer of specification, reducing the typical ALFA generality but at the same time facilitating the usability of the models. It allows for efficient transport between nodes; the feature is straightforwardly inherited from the FairMQ and FairRoot strong points. In addition, it can also provide mapping of the data objects into shared memory or to GPU memory upon request. Layout of a single message is split in two parts, a **header** and a **payload**. The first contains pieces of *meta*-information like the data source (e.g. the detector or process creating it), a data description (the type of data contained in the payload) and other ancillary information like the encoding type in the serialisation. Different types of payload support different serialisation methods and data formats and in all the cases where serialization of complex objects can be afforded, native serialization of C++ objects, for instance the histograms, is assured via a native ROOT serialization.

2.2.3 Data processing Layer (DPL)

The transport Layer provided by FairMQ is generic and experiment agnostic. It hides lower level implementation details of the transport but it at the same time is quite basic with very low level abstractions that are not familiar to the final user willing to build algorithms on top of them. the Data Processing Layer (DPL) is deployed to facilitate this phase on top of the O^2 data model, as a third abstraction layer that merges the concepts of a computational workflow described by ALFA together with the ALICE-specific data entities to be used in any of the computations connected with Run 3. The DPL allows developers to describe any computation as a set of topologies of data processors implicitly organized and connected in a logical dataflow that describes how data will be transformed.

Each processor needs to declare upfront its input and outputs types. The user can specify the configuration of the workflow in a declarative manner. Different workflows can also be connected into pipelines where the output of a first processing is passed as input of the next one. In figure 2.2 it is reported the representation of a typical workflow. Taking the

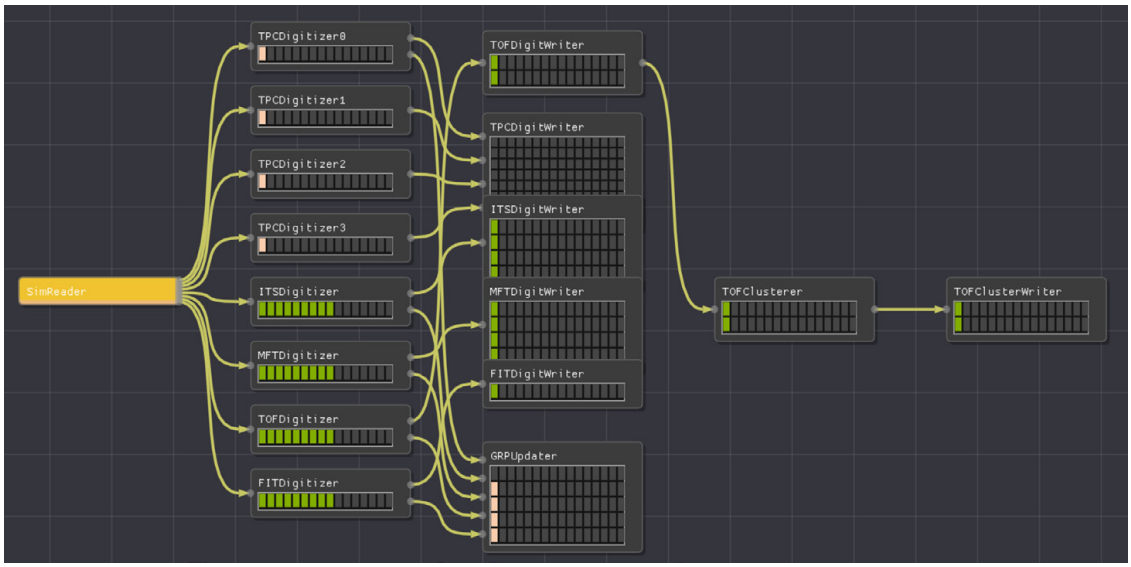


Figure 2.2: Visualisation of an example of workflow, specifically for the time projection chamber (TPC), the picture has been taken on a laptop, with the whole workflow running there. The same interface would have been available for a deployment on a set of nodes hosting a device each. Source is [42].

configuration as input DPL will create a topology linking the producer of some kind of data to all of the consumers that required it. The framework will also, transparently optimise the logical dataflow description to its physical topology. Support for other languages or interactive construction in a GUI is foreseen.

These three software tools are the enabling technologies for the design of almost every section of the O^2 in terms of flexibility, scalability and portability. Next sections will focus more on the parts of the O^2 that have been relevant or correlated to this work.

2.3 Detector simulation with ALFA: an asynchronous and scalable system

A simulation is the software procedure that aims at producing numerical *pseudo-data* with the same format of those collected by the experimental apparatus but being reproducing the whole lifetime of particles produced in computer-simulated collisions in each of their possible interactions. Simulation software is well known to be, in general, very demanding in terms of computing power resources and storage. In recent years, High Performance Computing (HPC) facilities are becoming more and more popular because of their convenience: they are very compact clusters equipped with multi-core computing

nodes, delivering a lot of power in terms of floating point operations per second (FLOPs). Usually, on board of the nodes of these centres the fraction of total RAM memory available in average per core is very low, because the majority of HPC computing use cases are very computing intensive (in terms of operations per seconds on the same data) compared with input/output operations. In contrast, the high energy physics computing (HEP) cases, mainly simulation, data reconstruction and physics analyses, are defined as high-throughput computing (HTC). The HEP computing model is usually data-centric, meaning that the size of the data (either in input or output) is the greatest contributor to the computational overhead of a process. Simulations are a notable exception: their algorithms are usually very CPU intensive, with very low-sized input data and large outputs. They are the most computing efficient workloads that can be run on a data centre, defining the computing efficiency as the ratio between the amount of time spent by the CPU in doing operations and the wall-clock time. Therefore, simulation resembles very closely a HPC use case, a part from the large amount of virtual memory usually needed to store detector geometry and Monte Carlo engines. High Energy Physics computing cases are intrinsically parallel by their nature, since at the level of the single event (simulation and reconstruction) each processing can be done independently from the others by separated processes. In the ALICE computing model before the O^2, the only level of parallelism was at the *embarrassingly parallel* level, meaning that assigning one event to a single core for many cores, represented the relevant speedup. In the case of simulations the limitation, especially concerning Pb–Pb collisions, up to hundred of thousands particles are generated and propagated through the detectors by each process. The total memory used by the single process, of the order of ∼O(GBs)/core to run jobs optimally. On an HPC-like architecture is therefore is more likely that the execution of HEP workloads will saturate the memory if all the available cores are used, making HPC farms not suitable for being efficiently exploited.

Simulation in the O^2 overtakes this approach, aiming to break the monolithic process that was taking care of simulating the whole event with a single core using a large amount of memory. A portable[v] system based on message-passing has been developed also migrating part of the existing code from Run 2. Thanks to the aforementioned software components, it is also horizontally-scalable efficient in using the available resources on the target architecture. Typical HEP simulations are written relying on standalone programs like the Geant4 toolkit[12], natively or using the Virtual Monte-Carlo (VMC)[51] layer. VMC enables then the possibility to use other simulation engines, for instance Geant3[30] or Fluka[21]. For LHC Run3, ALICE and FAIR are sharing some software components to reduce the amount of custom code to write a VMC detector simulation. One of the major challenges in Pb–Pb collisions simulation in ALICE, is the complexity physics events can reach, counting thousands of primary particles (originated directly in the collision vertex) emerging from a single interaction. In contrast to the much smaller *pp* events, this can lead to a very long CPU time required to obtain the full detector response for each single event (∼O(hours)). Most simulation engines and frameworks commonly treat an event as an atomic unit of work, leading to long critical computations, not "checkpointable", meaning they cannot be paused and resumed later upon dynamic scheduling of other tasks on the

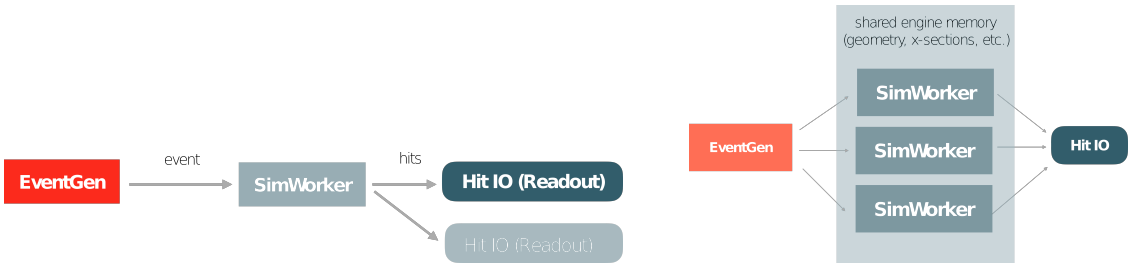
[v] deployable on all sorts of computing equipments, HPC cluster included

same computing node. This was historically a blocker for projects of dynamic and opportunistic usage of computing resources, whenever timescale of the opportunistic resources was lower than the average duration of a simulation workload that was worth to deploy on them.

2.3.1 Simulation structure

Referring to the initial monolithic simulation task, the O^2 simulation process handles everything from the event generation to the input-output (IO). Three ALFA devices are defined: a generation process, a simulation worker and a hit collector process for the IO. The event generator simulates the properties (number, nature and kinematical quantities) of the particles produced in a collision and forwards them to the simulation worker in a *request-reply* pattern. The worker runs the detector simulation via the VMC engine and sends the generated hits to the merger, only responsible for writing results on a ROOT file. Flexible management of the workflow topology allows for increasing the number of writers, each one taking care of one single sub-detector, to achieve also logical parallel IO. Figure 2.3a shows a typical simple layout for full-event generation with a single worker. Event generator can in some scenarios be promoted to an actual "server" responsible of distributing events or chunks of them to workers that are also deployed on other computing nodes. The event-splitting, also called **sub-event parallelism**, becomes extremely valuable in simulations of Pb–Pb collisions. It allows the user to scale the number of simulation workers and asynchronously process chunks of large events in a parallel fashion. Multiple independent devices of the "worker" type are spawned contemporaneously and each of them starts asking the event server for payloads to process. In the case of simulating full events, it is still available the native multi-threaded parallel feature introduced with *Geant4-MT*. Also multi-processing (from FairRoot) and multi threading can be combined to go beyond the initial features, in some specific architectural configurations where it can be beneficial. Finally, since the simulation worker supports the VMC engine, parallelism is achievable also with multiple instances of Geant3 and Fluka. They do not support natively in-event parallelism, the possibility of using them together with the sub-event parallelism provides new perspective to their engines. Current implementation of the multi-process approach overtakes also the issue due to the replication, operated by the different processes, of the static objects like the VMC engine and the instance of the geometry of the detector. The technique is based on *late-forking* of the processes: a single VMC engine is fully initialized at the beginning of the computation and only thereafter the other workers are created by repeatedly forking off from the first process, thus allowing for using native operating system *zero-copy*[vi] directives to share the objects across multiple workers. Ultimately, by sharing common data across parallel processes, the total amount of allocated memory per process is drastically reduced down to $\sim$ 500 MB per process[85] instead of $\approx$ 2 GBs with former framework. With the new simulation workflow included in the O^2, ALICE will be able to fully benefit from parallel architectures with multi-threading multi-process and message passing. The reduction of the needed resources per single process, allows for

[vi] only references to data are replicated, mocking the accessed object in memory but actually reading all from the same source.



(a) (b)

Figure 2.3: Two possible layouts for different scenarios. In scenario 2.3a a typical layout with one event generator or server, multiple I/O units can be deployed to achieve parallel logical writing of the simulation results. In scenario 2.3b multiple pieces of a single events are delivered to separate workers, they will process the simulation in parallel and results will be merged later. Recently a parallel merging facility is also available.

running on architectures not natively conceived for HTC, as the HPC nodes, although with a computing efficiency usually limited by the memory-per-core ratio. In appendix A will be presented a work done contextually with the development of software for O^2, that tested feasibility and performance of the deployment of the simulation stack on HPC nodes at the A2 partition of the MARCONI[71] cluster at "Consorzio Interuniversitario del Nord-est Italiano per il Calcolo Automatico" (CINECA).

2.4 Reconstruction using DPL

Reconstruction in high-energy physics software is the process of extrapolating from the *raw data* the particles yielded in a collision and stored in physical objects representations. Raw data are either generated via Monte Carlo simulations or acquired from experimental apparatus, hence detector reconstruction is not aware of their origin. In the O^2 framework reconstruction is steered by DPL, each computing task is implemented as a "DataProcessor", a (FairMQ) device that can be defined as *input*, *algorithm* and *output*. Data exchange is managed by FairMQ, and reconstruction workflows are combinations or chains of DataProcessors. As anticipated, the reconstruction workflow description is done in a declarative manner, while all the underlying optimisations and topological sorts are done transparently by the DPL. An instrumental feature provided by multi-process workflows is the possibility to plug different workflows together. Each detector will have its own reconstruction workflow, composed by many devices and capable to scale the number of its DataProcessors, depending on the requirements to accommodate diverse requirements. The work presented in this thesis has been integrated in the inner tracking system (ITS) reconstruction workflow in its serial version. DPL supports data offloading to dedicated accelerators such as GPUs. Therefore also the GPU reconstruction workflow for the ITS will be supported. Anticipating what will be explained in section 2.5 and chapter 7, there is a dedicated GPU reconstruction framework, also built in the form of a DPL workflow, that will integrate all the GPU-supported reconstructions, therefore the GPU ITS primary vertex reconstruction has been developed in a temporary standalone implementation. The

possibility to distribute the workloads related to the development of different pieces of the O^2 provides not negligible advantage in sharing the effort for developing different modules. Not only is possible to segment different areas of competence for different fields, allowing experts in each field to independently work on self-consistent pieces of software, but also build components of a modular schema that can be moved or replaced by other components.

For instance, taking as an example the inner tracking system reconstruction workflow, it aims to process raw input data and reconstruct primary interaction vertices and the trajectories of charged particles produced in the collisions. Input data to the workflow, in its early stages, were data read from trees contained into a ROOT file. To develop and test the reconstruction chain it is sufficient and natively supported by the DPL facilities. However, the topology can be extended by including a *proxy streamer*, which is responsible to read data from the same file and stream them into a pipeline that can be used as an input channel for other workflows. Next section will describe better in details the ITS reconstruction workflow as an example of all the phases of the reconstruction from Monte Carlo simulated data.

2.4.1 Reconstruction for upgraded ITS

Reconstruction workflow for the ITS is quite simple, depending on the nature of the input data it can be resumed in three main routines. Each is assigned to a DPL DataProcessor. The current layout for the ITS reconstruction reads the simulation data from files and applies the so-called **digitisation**. Output of the Monte Carlo simulation process are the analytical functions that describe the trajectories of charged particles in the generated events and their hits due to the interaction with the detectors. A that stage the information is not yet provided as measured by the detector itself, in fact the response from, the detector is simulated during the digitisation. Its task is to convert the Monte Carlo geometrical truth in data that are of the same format of the digital output of the data acquisition chips. Digitisation is the first instance after the Monte Carlo truth generation, which is not affected by any efficiency or resolution power of the detectors, where the simulated data are "measured" by simulated detectors. By means of *ad-hoc* configurations different scenarios for the measurement of the signal are reproducible. For instance, one of the pivotal parameters that condition the digitisation is the frequency of simulated acquisition; by setting different duration for the ITS readout frames (ROFrames) it is possible to determine the distribution of data across the acquisition frames. That effect, together with the possible manipulation of the *in-bunch* pileup, allows for producing different data related to measurements that share the same Monte Carlo truth.

Later on in the workflow, *digits* are sent to the **clusteriser**, a device responsible to identify clusters of adjacent pixels corresponding to the passage of a single charged particle throughout the active regions of the ITS layers. The clusteriser is also responsible to extrapolate geometrical information about centroids of the clusters, centre of gravity of the geometrical patterns of each clusters that is converted into a two-dimensional point on the stave reference system.

Ultimately the data flow ends in the actual reconstruction routines: the **vertex finder**, the software implementing algorithms aiming to find the coordinates of the point(s) in which the collision(s) took place, named vertex(vertices), and the **track reconstruction**

(tracker), which is responsible for reconstruct and fit the tracks associated to the trajectories of charged particles crossing the ITS layers and leaving clusters (space-points) as seeds.

2.5 GPU reconstruction in O^2

As anticipated in section 2.1.1 the reconstruction software stack of the O^2 will include a framework with a dedicated section to the General-Purpose GPU computing (GPGPU). The adoption of GPUs is motivated by resource-optimisation goals.

It was already shown in the Technical Design Report (TDR) of the O^2[32] that, for the reconstruction and data compression of the Time Projection Chamber (TPC) data, GPU acceleration represents an efficient technique for boosting the processing speed, thus reducing the computing time. Apart from the faster data acquisition rate, the TPC has not been modified very much in its structure. Reconstruction algorithms, which at that time had a GPU implementation for the High-Level Trigger code, served as a reference to demonstrate the convenience in having, for the software and hardware upgrades in view of Run 3, dedicated GPU resources for the online (synchronous) reconstruction. Modern best GPU showcases as the one reported in figure 2.4 demonstrated a speedup in the execution of the TPC track finding algorithm large enough to compare the efficiency of a modern GPU to 40 CPU cores running at 4.2 GHz. These results encouraged developers

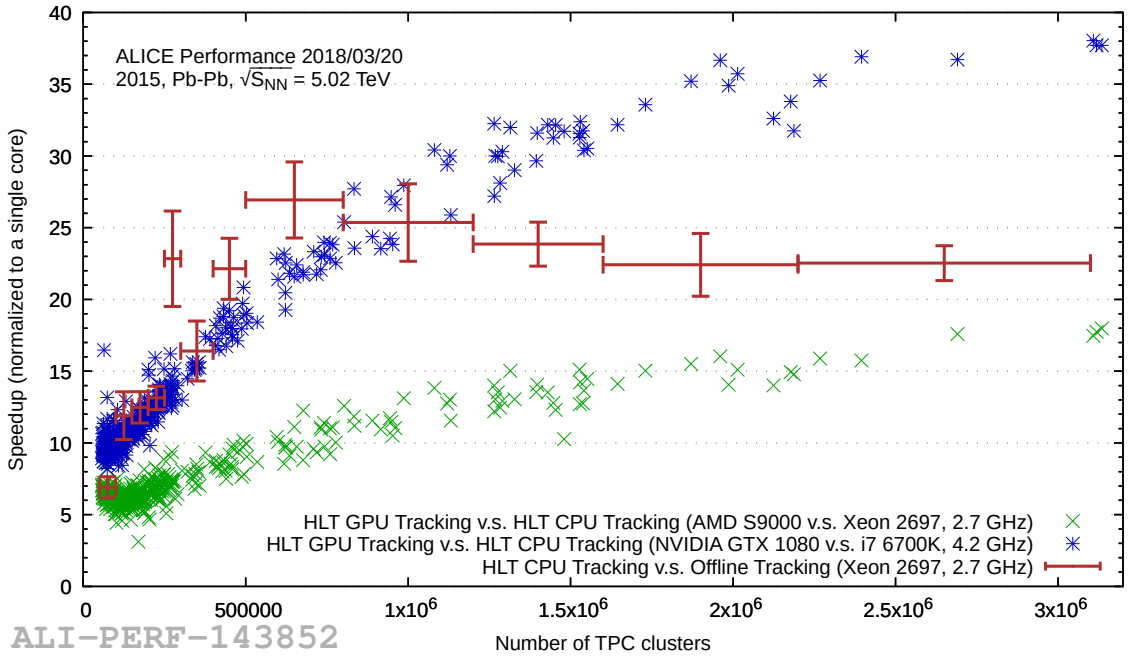


Figure 2.4: Speedup of the ALICE TPC tracking on GPU normalised to a single CPU core.

working on solving similar problems for the upgrades of the other detectors, like the ITS, to evaluate the possibility of implementing GPU-accelerated routines. Among others, there

is the implementation on GPU of the primary vertex reconstruction algorithm presented in this work.

Current state of the art of GPU code base in the Online-Offline platform has grown a lot, with respect to the state declared in the TDR. Other detectors like ITS and TRD have diversely advanced prototypes of workflows based on GPUs. Also, in order not to be vendor locked to a specific provider of graphic cards, different computing languages are supported, notably: OpenCL[76], CUDA[67] and Heterogeneous-Computing Interface for Portability via the ROCm[14] platform.

The final goal is to have a unique DPL-based framework responsible to orchestrate a reconstruction chain based on all the GPU-implemented reconstruction stages limited to those detector that employ them. An implementation of a comprehensive DPL device that includes TPC and TRD GPU reconstruction code is in place. It takes control over the GPU resource by allocating the whole memory and re-implementing a memory provisioning system by intelligently distributing available resource partitions by means of a custom logical memory allocator. This feature is clearly available only for the executions that shares the same logical memory partition with the GPU framework itself. Nevertheless, it also abstracts the access to the GPU in such a way that the code to be written to launch offloaded executions of routines on graphic cards is not bounded to a target architecture (vendor solution). Its interface is a wrapper around the aforementioned GPGPU libraries that automatically detects the available driver and libraries on the hosting machine and produces the corresponding binaries to be launched.

From the developer perspective the framework is also friendly enough to allow for the inclusion of external GPU code contained in other libraries, exposing the possibility to extend its interface to use the external utilities calling them from the framework instance itself. As an example, for what concerns the work presented in this thesis, no time has been spent in integrating it with the framework. It means that the memory management is left to the external GPU library compiled for the ITS tracking. However, the execution can be steered by calling methods of an object that is created by the GPU framework. Next step in the development will be the final integration of the memory management with the DPL device that will host the GPU reconstruction, in order to organically include it in the main workflow.

Concerning the ITS GPU tracking algorithm, it is in a similar situation as the ITS GPU vertex reconstruction (sometimes "vertexer"), with a standalone implementation that can be steered by the GPU reconstruction device.

Chapter 3

General-Purpose GPU computing for the O^2

A graphic processing unit (GPU) is a specialised device originally designed and used to only rapidly manipulate large amounts of graphical pixels. Historically, GPU were conceived to operate with heavy graphics-driven applications requiring computationally expensive image renderings. If compared to a Central Processing Unit (CPU), the design of architecture of a GPU foresees a higher number of cores per volume unit (density of cores) thus to address the execution of many processes in parallel. The focus in designing this kind of accelerators is not only in having fast-frequency cores to quickly perform computations, but also to optimise them to be very efficient in performing in parallel the same operations on different pieces of data. This translates in high *throughput* computations by performing on each core the same sequences of operations following the most deterministic paths possible. To do that GPUs spawn a large number of (hardware) *threads* that act singularly at the same time on a small partition of a homogeneous input data buffer. This paradigm is also known as the "**Single Instruction on Multiple Threads (SIMT)**"[60], and it is considered an extension of "*Single Instruction on Multiple Data (SIMD)*" computing paradigm well known from the *Flynn's Taxonomy*[45] in the parallel computing domain. Modern CPUs, on the other hand, focus on the fast execution of *few* non deterministic tasks: they are optimised for context switching. For this reason in order to improve CPUs performance it is preferable to have faster cores instead of a very large number of them. Over the last decade there was a very active research and development to address a broader set of computing problems, not only directly related to image processing or rendering with the use of graphic cards. A sizeable effort was dedicated to computing intensive workflows where most of the time is spent in memory and computing operations instead of input-output (I/O). In order to exploit the possibility to use GPUs for such computing efforts, there have been built interfaces to interact with codes not related to graphical purposes, for example for linear algebraic manipulations. The General-purpose GPU computing or **GPGPU** computing is the term that denotes the adoption of a GPU to address general-purpose scientific and engineering computing problems, by means of the execution of intensive parallel tasks which, in their logical flux and structure resemble very closely the ordinary workflows executed by graphics cards for which they are optimised.

3.1 Architecture of a graphic card

Architecturally, a CPU is built upon few cores with lots of cache memory that can handle few software threads at a time. The design of CPUs evolved to minimise the latency in processing not predictable inputs such as devices plugged to a workstation. A non predictable workflow has a lot of switches or conditional statements in its logic to make it really hard the optimization of its execution with techniques like *branch-prediction*. This explains why software and hardware in CPUs must cope with every possible evolution of the execution. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously. These threads are not supported by an architecture optimised for unpredictable conditional statements and does not aim at reducing latency. In figure 3.1 it is depicted a schema that compares structural differences among typical modern GPU[i] and CPU. It is important to notice the different proportion between similar components on two different setups. Core components are similar and they are hereafter

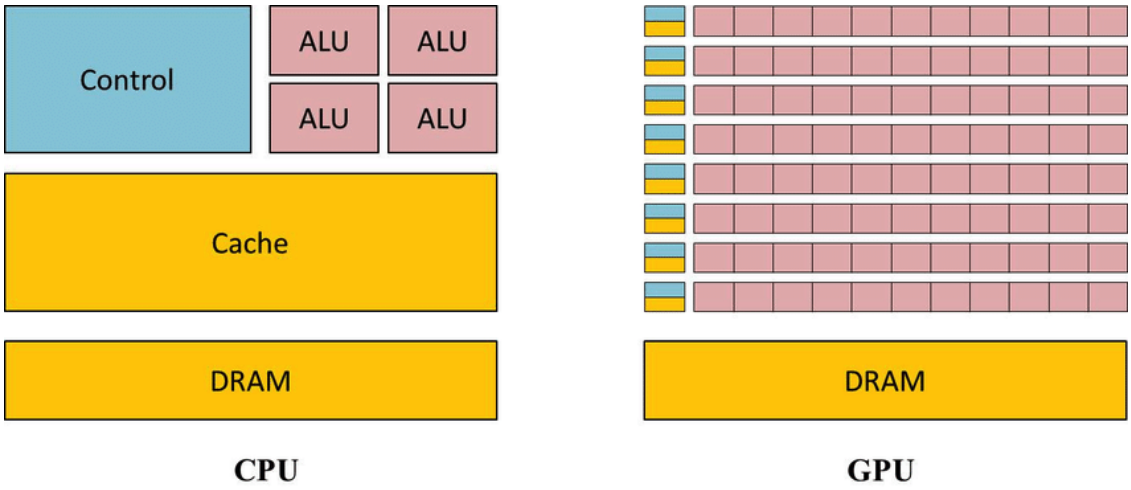


Figure 3.1: Comparison between CPU and modern GPUs structures. The Control Units (CUs) are reported in light blue. The yellow boxes represent the DRAMs and the caches and the ALU/FPUs are shown as pink boxes. It is relevant the different proportions of dedicated resources in the two devices.

described.

- Control Units (**CU**s): in charge of the management of the processor operations. They are responsible to tell the device memory, arithmetic and logic units and input and output devices how to respond to the instructions that have been sent to the processor.
- Arithmetic-logic or floating-points units (**FPU/ALU**s): these units carry out arithmetic and logic operations on the operands in computer instruction words.

[i]Starting from 2006 with the release of the Nvidia GeForce®8 series, processing units on graphic cards became more general-purpose from an instruction-set perspective (not anymore including only graphic-related operations), making them more suitable for introduction of GPGPU computing.

- Dynamic random-access memory (**DRAM**): the virtual memory to store temporary data used by applications. GPUs have their own dedicated memory implemented in distinct hardware not related with CPU.
- **Cache**s: hardware components that store data so that future requests for same data can be served faster than the DRAM. Usually, they present a hierarchical structure in levels. Levels are ordered by capacity, smaller buffers provide quicker responses. CPUs have up to three levels of caches whereas GPU usually only two.

One can also observe that on GPU the replication of many minimal setups similar to CPU is present even if it is characterized by smaller caches and a larger number of processing units. This represents a crucial difference between them: a larger number of processing units with smaller cache lines reflects a different target use case for GPUs.

The smaller caches are shared across many ALU/FPUs, since they are designed to enhance performance in SIMT paradigm implementations. This means there is no need, in their usual workflow, to reserve much space on cache to store parameters that are occasionally used, depending on the direction of the workflow. Since the software running on GPUs is designed to be focused on performing many operations all at the same time on many different cores in parallel, one needs to optimize it to efficiently use the cached memory.

3.1.1 Hardware anatomy and concepts

At present, the market of GPUs is mainly dominated by two competitors: Advanced Micro Devices (AMD®)[40] and Nvidia®[66]. Chips from AMD present differences in the inner components, but they share a lot of similarities in design pattern. A notable example to notice differences and similarities between *Navi* from AMD and *Turing* from Nvidia is reported in figure 3.2. Both designs show a *tiered* structure in the organisation of each component. Navi GPU (figure 3.2 on the left) is mainly divided in two blocks called, in AMD jargon, Shader Engines (SEs). A single block is then partitioned into another two areas called Asynchronous Compute Engines (ACEs) and each one of them comprises five blocks, named after Workgroup Processors (WGPs). A WGP is made of 2 Compute Units (CUs).

For the Turing design (figure 3.2 on the right), names and quantity of each component may vary, but the operational hierarchy is very similar. It presents six Graphics Processing Clusters (GPCs), each with six Texture Processing Clusters (TPCs). Each TPC is built up of 2 Streaming Multiprocessor (SM) blocks. GPUs perform much more work for every unit of energy than CPUs. This feature makes GPUs a key element for supercomputers that otherwise would over-saturate the availability of electrical power.

RDNA vs CUDA computing units, a difference of execution. AMD and Nvidia use a different approach in their unified shader units, even though often the terminology used is the same. Nvidia's execution units, the **CUDA cores**[ii], are scalar in nature:

[ii]The Compute Unified Device Architecture (CUDA) acronym is inherited from the Nvidia official software framework for GPGPU, better described in section 3.2.2.

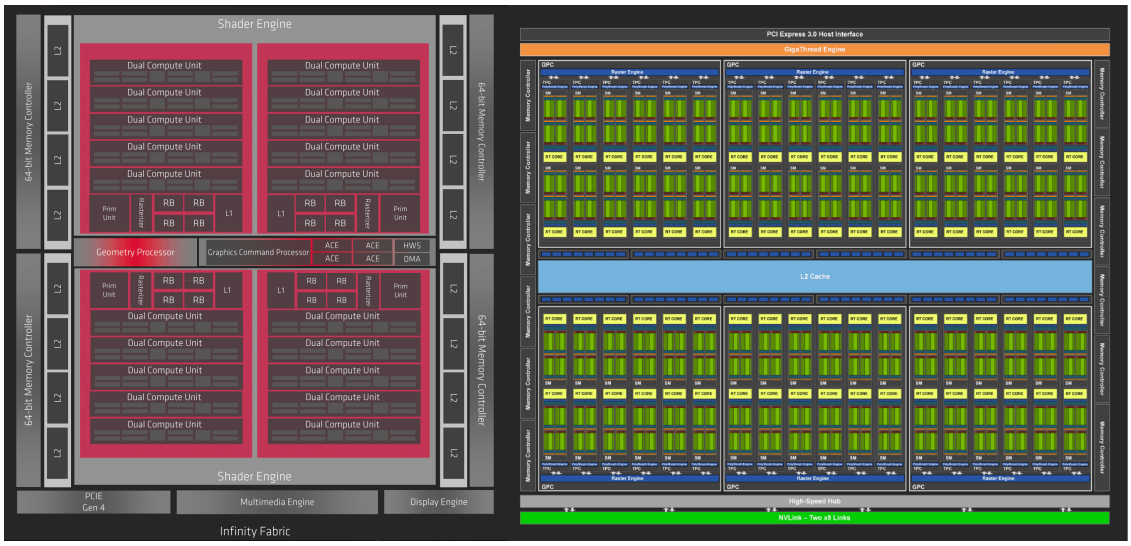


Figure 3.2: Comparison of two architectures available on the market: Navi from AMD and Turing from Nvidia

one unit carries out one mathematical operation on single data component. The most significant change in recent years of development involves how they are arranged and sectioned other than the increase of number of computing units. One CUDA streaming multiprocessor contains 4 processing blocks, each containing

- 1 Instruction scheduling and dispatch unit
- 16 IEE754-compliant FP32 scalar arithmetic-logic units (ALUs)
- 16 INT32 scalar ALUs
- 2 Tensor cores: accelerator for Artificial Intelligence applications brand new addition in this architecture, not covered nor used in this work.
- 4 Special function units (SFUs)
- 4 Load/Store units: handle cache read and write

CUDA core can perform one float and one integer instruction per clock cycle on one data component and the scheduling units organise them into groups such that, the result appears to the developer to be the equivalent of a vector operation. In contrast, AMD's units, **Stream Processors**, work on vectors: one operation is performed on multiple data components. For scalar operations, the Navi architecture has one single dedicated unit. Each Navi RDNA[41] computing unit contains two sets of

- 32 Streaming processors: IEE754-compliant [**IEE754**] FP32 and INT32 vector ALUs
- 1 Special function unit (SFUs): perform very specific math operations (e.g. trigonometry and square roots)

- 1 INT32 scalar ALU
- 1 Scheduling and dispatch unit.

The aim of this comparison between Navi and Turing was to illustrate how both vendors nowadays provide devices with execution units offering reasonably similar feature but they use a different approach on how to process such features. The necessity in being to some extent more homogeneous is a consequence of the need to comply with the requirements of software that runs on top. From a programming perspective, the usage of a graphic card either for image rendering or GPGPU computing, at a certain point in the logical structure of the program execution requires a unique interface to communicate with the accelerator. Such an interface is represented by the "driver", a piece of software that exists at the level of the kernel of a computer Operating System.It is responsible to translate the calls received by the Application Programming Interface (APIs), invoked by the program, into machine instruction for the hardware. Drivers from AMD and Nvidia are different at many levels, but from the perspective of a user there must be a certain layer that abstracts the usage of any GPU regardless the vendor or the model. Especially for what concerns the usage as graphical processor, high level libraries are present and well established, such as developer of graphical tools or video games can program against any GPU via generic calls and very rarely need to access lower level, model or vendor-specific features. In the next section the programming model for GPGPU computing will be described giving emphasis to the comparison between the two frameworks adopted by AMD and Nvidia.

3.2 Programming model for GPGPU computing

A general-purpose GPU (GPGPU) pipeline is a parallel workflow between the CPU and one or more GPUs. The GPU, operating at lower frequencies but at larger scale in terms of number of cores, processes data by running "kernels": single streams of instructions executed each core. In the meanwhile, the CPU is busy executing independent tasks, possibly completely unrelated to what happens on the GPU, among them the orchestration of the graphic accelerator itself. The organization of data in graphical forms (i.e. vectors, textures, matrices) performed with GPGPU is aimed to have large speedups and optimised processing. The definition of speedup in parallel computing as a function of the scaling of the number of cores comes from the Amdahl's law. It was first formulated in [15], and an adaptation that takes into account its scaling as a function of the number of cores, is reported in equation 3.1. Being r_p the fraction of a program execution that can (indefinitely) benefit of a scaling as the number of parallel executions increase, and n the number of parallel resources provided to the execution, the speedup function $\mathcal{S}(n)$ is defined as

$$\mathcal{S}(n) = \frac{1}{(1 - r_p) + \frac{r_p}{n}} \tag{3.1}$$

Term $1 - r_p$ represents the so-called "serial part" of the program, that cannot be split in parallel subroutines. GPUs are very valuable in solving "high-throughput" problems where large volumes of data are mostly independently processable by large amounts of subroutines. The processing of graphic workloads drove the design of the first hardware and software solutions both for AMD and Nvidia. In few years the generalisation of the

graphic hardware allowed for using those additional processors onboard of many computers, to solve highly parallelisable computations. At the beginning, the ambitious idea behind the adoption of the GPGPUs was to overcome the saturation predicted by the Moore law[iii]. Moore predicted the evolution of the chips density (number of transistors per volume unit) as a function of the time describing very accurately the increase of computing power happened from the creation of first microprocessor up to the first ten years of 2000s.

Over last decade, however, such a trend saturated because of reaching physical limits in the scales of microprocessors down to the $7nm$ architectures. Both algorithm development and device design started to investigate larger scale architecture with large amount of core more focused on the throughput than on the frequency of each core.

On the software side different frameworks and suites of profilers and debugging tools to develop GPGPU programs appeared on the scene. Schemas presented are similar: there is a driver that exposes the hardware device to the operating system, then libraries are created to communicate with the driver and offload instructions to the GPUs via Application Programming Interfaces (APIs). Hereafter an overview of the status of the art in terms of software development will be presented and a particular attention will be given to those used in the present work: CUDA e HIP[iv].

3.2.1 Layout of a GPGPU program

As anticipated, the hardware structure of GPUs from different vendors are different in the distribution and balancing of the number of inner components and for some unique features and capabilities that uniquely characterise some aspects. At the same time their design is close, they will eventually need to comply with most popular usage patterns dictated by main use cases, for instance the willing to have uniform high-level graphic, mathematical and algorithmic libraries to ease the software development.

These similarities are resembled also on a logical representation, where it is found that the resource abstraction and the body of a GPGPU programs are very similar in their schema and workflows. Both software layers used in this work are programmatically accessible via standard programming languages such as the C++ which is also the language used in the O^2 framework. The used computing paradigm foresees two entities: the *host* which is the computer that "hosts" the GPU and the *device* which represents the abstraction of the dedicated discrete piece of hardware to which offload GPGPU code. Following previous definitions it will be referred to as host or device code, that part of the program which is executed on the former or the latter. GPGPU programs are made by the combined interaction between these two kinds of execution, but more importantly it has to be noted that the host has the responsibility of generally steer the device execution. Operations, like memory transfers from and to host and device and calls for kernels executions, are always invoked as host directives (APIs) to manage the device. Computational payloads are coded in the so-called *compute kernels*: this term is shared across different frameworks and with a large class of devices containing for example the Field Programmable Gates Arrays and

[iii] first time presented in this article: [63]

[iv] the Heterogeneous-compute Interface for Portability (HIP) framework from AMD will be extensively introduced in section 3.2.3.

represents a single stream of instructions that operate on a large number of data parallel work-items. Kernels are similar to *inner loops* in the sense that their scope is usually associated to a single instance of an iteration assigned to a specific index. They may be specified by a separate programming language that, in the case of this work, is a *dialect* of the C++, enriched with some specific keywords. Parallelism and high-throughput is achieved by having many kernel instances running at the same time on a GPU possibly partitioning a complex problem into small pieces that can be processed in parallel. To accommodate this redistribution of the workload on the hardware architecture described in section 3.1.1 it is rendered a logical representation of the resources that can be used for general-purpose tasks. The "smallest" entity is a **thread**, which is responsible to run the execution of a single kernel. Threads are organised in such a way that those with the same instructions are grouped together, typically into collections of 32 for Nvidia and 64 for AMD's Graphics Core Next (GCN) architectures. This regroup is still performed at hardware level. AMD calls this collection a **wave**, whereas Nvidia call it a **warp**. In both designs, wave and warps are independent, meaning there is not a mandatory synchronisation point in which, to start the next execution, a warp/wave necessarily needs to wait the end of other warp executions. Threads are also logically (from the developer perspective) organised in **blocks**, and this is for a better data mapping and consequently a better performance. Blocks can host a number of threads which is limited by the architecture. At present, modern GPUs have blocks that can contain up to 1024 threads. In the CUDA implementation, threads on the same block run on the same *stream multiprocessor*. They can benefit of a communication via **shared memory**, and have collective primitives such as *barrier synchronization* and atomic operations (operations that does not permit other instruction to interfere with resources involved in its execution until it terminates). Once a thread block is launched on a multiprocessor (SM), all of its warps are *resident* until their execution terminates. Therefore, new blocks are not launched on an SM until there are sufficient resources for the new block such as free registers and free shared memory. Threads are indexed following a tridimensional order that is delimited by the size of the containing block. Usually their index along specific dimension is noted as `threadId.{x,y,z}`. Block size is determined at the kernel launch and is represented by a triplet of numbers regrouped in a `dim3` structure. The global volume of a block is calculated as the product of its non-null dimensions; when the third component of its dimension is null it is considered to be a *bi-dimensional* block, unidimensional in case `blockDim.y` is null too. Blocks cannot access each others *shared memory*, which is privately available only to threads. They can however have access to the *global* and *constant* memory. Access to this memory is usually slower because they are mapped to the DRAM of the GPU, whereas the local and shared memories are spread on the different levels of cache. A **grid** is a *superset* of blocks. Blocks inside a grid have their own unique indexing that resembles exactly the logic used for threads. Grids are tridimensional too, in their maximum extension by volume. In figure 3.3 there is a summary of the logical structure used by the developer to abstract GPU computing model. On the left side of the schema: a single kernel is launched from the host (the CPU) to be executed by many threads on the device (the GPU). Threads are sorted in three-dimensional structures called blocks, which are, in turn, organized in three-dimensional grids. The dimensions of blocks and grids can be decided in the runtime. On the right side there is the memory hierarchy: threads have access to data from many different memories with different scopes. Registers and local memories are private for each thread. Shared memory let threads belonging to

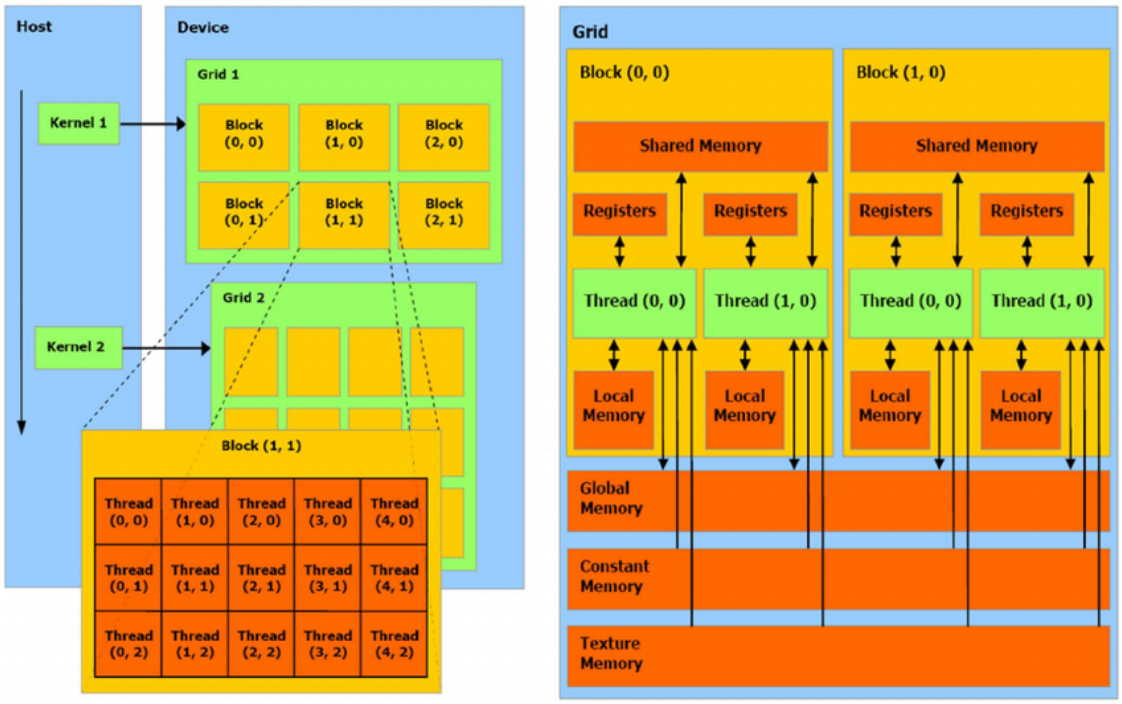


Figure 3.3: Schematic representation hierarchy of threads. This schema is general enough to be valid for CUDA and HIP.

the same block to communicate, and has low access latency. All threads can access the global memory, which suffers of high latencies. On some new architectures such latency is reduced by further cache lines. Texture and constant memories can be read from any thread and feature a cache as well.

3.2.2 Nvidia's Compute Unified Device Architecture

Nvidia develops and maintains the Computing Unified Device Architecture (CUDA)[67] software stack: a parallel computing platform that expose an APIs model. This platform provides a software layer that gives direct access to the parallel computational elements and virtual instruction set of an Nvidia GPU. Such a programming platform is accessible to software developers via CUDA-accelerated libraries, compiler directives such as OpenACC[87] and extensions to standard programming languages such as C/C++[54]. It is oriented to GPUs Nvidia branded and requires an Nvidia driver.

3.2.3 HIP: Heterogeneous-compute Interface for Portability

Heterogeneous-compute Interface for Portability (HIP) is a C++ runtime API and kernel language that allows developers to create **portable applications** which can run both on GPUs from AMD and Nvidia. The goal of the platform is to find the lowest-common-denominator logical level between different architectures and provide an interface which enables the usage of essential and dedicated hardware features (generic GPU) on a

not vendor-locked code. HIP has been introduced and promoted first by AMD to reduce the gap with the *de facto* mainstream and more established platform from Nvidia: the purpose is to allow the developers to write GPU applications to a common C++ syntax and to abstract specific APIs behind a C++ interface, where both host and device code can be written in the same syntax and the specification for different platforms is done automatically later during the compilation phase, the moment where the machine language is finally produced both for CPU and GPU. HIP source code can use a rich set of C++ features including templates, lambdas... The resulting C++ code can be compiled with AMD's Heterogeneous Compute Compiler (HCC)[v] and NVidia CUDA Compiler (NVCC), using optimal compilers and tools on the corresponding hardware. AMD's claim is that HIP delivers performance similar to what can be achieved by coding directly in the native APIs. On top of that it las implements architecture-specific optimizations and APIs through conditional compilation for either platform, leaving the larger part of the code portable and allowing for specialisation wherever required. This will allow the platform to be compatible with latest or unique features released by both vendors, by leaving the possibility to directly call them from within the code. For the sake of this work the HIP platform has been used only to write code for AMD GPUs, leaving the native CUDA version separated to reduce the amount of workload in making the compilation uniform.

The "hipify" executable. An interesting facility provided by AMD is a conversion tool to "hipify" already existing CUDA code. As a matter of fact, CUDA and HIP languages are semantically very close: in most of the cases it is possible to map one-to-one each API call from one language to another . In case of in the need for some not shared feature (not mapped into a general interface) on a specific supported architecture, there is the possibility to use it anyway by explicitly calling it in the native dialect. Compiler front-end, called HIPCC, will be flexible enough to forward specific blocks of code to the subsequent compilation step driven by the platform-specific compiling tool (HCC or NVCC).

3.2.4 ROCm: AMD umbrella software for gpu-accelerated computing

The hybrid portable interface HIP from AMD, constitutes the core technology for a larger platform that has as a target to become a universal platform for gpu-accelerated computing. The Radeon Open Compute platform (ROCm) is an open source software with support for multiple languages that covers all the aspects: the drivers, the ToolChains (compilation-related tools) and the libraries. In figure 3.4 it is presented summary of the ROCm ecosystem for the compilation of HIP code on a GCN architecture (AMD). On left side, circled in white, the full cycle of production for an executable that runs on hybrid CPU+GPU architecture is represented. The `hipcc` compiler frontend selects and dispatches parts of the code to be compiled using dedicated compilers to the GCN or the low level virtual machine (LLVM)[58]. GCN (or RDNA in latest AMD GPUs) assembly and CPU Instruction Set Architecture (ISA) are produced after executing all the optimisations operated by LLVM. Eventually, these two components will be merged into

[v] eventually it will be phased out in favour of `hip-clang`

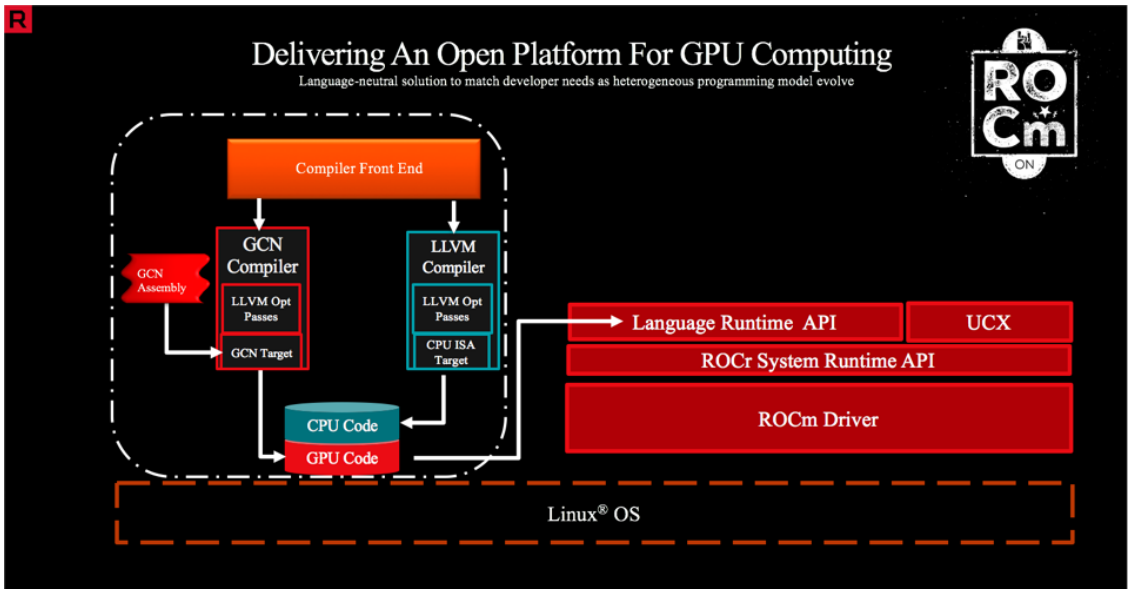


Figure 3.4: Sketch of the ROCm platform. This picture covers a case where the generated program is targeting a GCN architecture (predecessor of the RDNA architecture presented in 3.1.1)

a single program that will communicate with the ROCr System Runtime API using the Language Runtime interface. The described structure is very similar to Nvidia ecosystem. One possible scenario foresees that a frontend compiler can produce *fat binaries*[vi] to be later compiled with the NVCC compiler for Nvidia GPUs.

ROCm platform is a requirement to compile the HIP code used in the ALICE Online-Offline framework for the upgrade.

3.3 CUDA and HIP integration in the O^2

The ALICE Online-Offline (O^2) software is designed to be a unique stack which can be deployed on heterogeneous systems, from the local workstation or laptop of the scientist or software developer to the computing clusters such as the EPN. Its installation procedure is therefore required to be flexible and to provide all the available features depending on the underlying architecture.

The hybrid computing parts, as anticipated, are managed by a dedicated framework conceived for orchestrating the whole GPU reconstruction routines from the same entry point. The compilation and the deployment of the libraries that can be run on the targe computing node is achieved with a combination of automatic software and hardware detection tools and conditional statements that enables specific compilation rules. Specifically, it uses CMake[55] and dedicated modules to inspect the host to find current status. The

[vi] executable program which has been expanded with code native to multiple instruction sets, thus can run on multiple processor types.

default behaviour is to enable the creation of dedicated GPGPU libraries only in case some software requirements are met. The hardware presence is not really required at compilation time, for neither of the two GPGPU framework. This feature helps a lot many cases where the code need just to be proven to compile, for instance during the tests that ar run in the Continuous Integration (CI).

3.3.1 Continuous integration for GPU code

The O^2 release cycle includes a continuous integration (CI) step. Contributions are done using the collaborative tool called git[46], users create personal replica of the central repository, add their contributions in the form of git "commits" and then propose their addition to the community of developers and experts. Together with more on-the-topic discussions standard tests are run to check some general requirements and for the integrity preservation of the latest software. The code is compiled in an isolated environment that resembles the supported operating systems (OSs) in terms of software requisites and versions of libraries. The final goal of this procedure is to check whether external contributions to the central code, hosted on a web repository[35], are conform to standards and test whether the addition is a beneficial improvement. The process is automatic and based on the deployment of multiple Linux Containers (LXC), one for every request of code integration. A LXC is a lightweight virtualisation layer to provide custom environments to be used for each compilations. Their deployment is very agile on the dedicated cluster, but requires an orchestrator which among other tasks is responsible to coordinate the whole execution of the software release validation.

Both CUDA and ROCm libraries can be installed without the corresponding drivers to be actually installed on the host. The continuos integration, to be scalable, is deployed on multiple nodes of a dedicated cluster, which are not equipped with GPUs. This does not represent an issue since containers are available with CUDA and ROCm installed. The O^2 *autodetection* triggers on specific requisites installed in containers and the compilation can be tested. As by-product of the development of this work it has been produced the first version of the container used by the CI. Its inclusion in the testing process is not fully covered by this work but the structure is heavily inspired by the work described in the Appendix A which represents the results of a side project carried out alongside the main PhD work presented in this thesis. Later this test has been replaced by another one more seamlessly integrated with the CI ecosystem, where images of containers are built using a software called Packer[49] in combination with Docker[61]. It uses a descriptive language to enlist into a *recipe* all the requirements and operations to be performed by the container build workflow. Packer is capable, using different engines depending on the final product, to build different environments using diverse technologies. Also the Packer recipe has been produced as a side product of this work, with the intent to deal not only with the aspects related to the software execution but also with the procedure to introduce new pieces in the O^2 framework.

3.3.2 Integration of a GPU-based piece of software in O^2

The main core of this PhD project, which will be presented in the next chapters, is related to the development of the reconstruction code for the Upgraded ALICE Inner

Tracking System (ITS). The work includes also on the accelerated implementation of the software on GPUS: both Nvidia, using CUDA and AMD via HIP. It has been initially developed as a standalone tool integrated in the compilation of the Online-Offline framework. Later, also as a consequence of the introduction of the dedicated GPU framework its integration has been moved to use common interfaces. A full migration to the mainstream method is currently ongoing.

Chapter 4

Online reconstruction of primary vertices with upgraded ITS

During one data taking period, dense clusters of hadrons (protons or ions[i], typically ^{208}Pb nuclei), called *bunches*, are accelerated within the beampipes and then forced to collide at the experimental sites, where the two beams circulating in opposite direction cross one other. Each bunch contains in average 1.15×10^{11} *hadrons* that have a probability to interact with others in the upcoming approaching bunch. The probability to have a collision in a bunch-crossing depends on geometrical and kinematic factors, notably the total interaction cross-section, which depends on the colliding system, and the spatial density distribution of a bunch. It is therefore possible to have more than one collision per bunch-crossing because of different pairs of nuclei colliding at the same time. This phenomenon is called *in-bunch pileup*. It is a stochastic process and the average number of events $\mu(t)$ is computable as a function of the instantaneous luminosity $\mathcal{L}(t)$, the cross section of the process σ, the revolution frequency f and the number of colliding bunches n_b in equation 4.1

$$\mu(t) = \frac{\sigma \mathcal{L}(t)}{f n_b} \tag{4.1}$$

In general, depending on the pace in the detector data acquisition compared to the interaction frequency, it is also possible to have a pileup effect due to data from collision happened in two different bunch-crossing, this is called *out-of-bunch pileup*.

The **primary vertex** is the position in the tridimensional space of the interaction of two colliding particles (e.g. protons or Pb nuclei). In the scope of this thesis, unless otherwise stated, with "primary vertex reconstruction" it is intended the measure of the

[i] As per ions are in this case intended "completely ionized" atoms: only the nuclei of the accelerated elements. Complete ionization process is part of the preparation of the matter before the bunch creation and acceleration chain. All electrons are previously subtracted since the acceleration would anyway separate them, during the process and they would create electromagnetic noise.

vertex position obtained using data recorded from the ITS detector. Such an estimation is included as a step of a larger reconstruction workflow and produces a required information for the next step in line: the ITS tracking that aims to reconstruct the geometrical paths left by charged particles travelling across the detector. Later in the reconstruction workflow, the vertices position will be drastically refined as a result of the integration of all other pieces of information and matching with other results measured with different detectors. This will be the *global vertexing* phase, where the complete information of reconstructed particle trajectories is used to recompute the primary vertex position to the best precision available, ready to be used in physics analyses. Usually, these higher-resolution refits are performed *asynchronously* with more sophisticated techniques that often require a lot of computing time to produce results without looking for a trade-off between speed and precision or efficiency. However, a reasonably precise measurement of the vertex position is necessary at data-taking time (*synchronously*) not only as a starting point for the ITS tracking but also to monitor the beam position. While precision constraints are required by the tracker are relatively permissive in terms of spatial resolution, it is also true that the vertex reconstruction must be done as soon as possible, letting other online algorithms more time to run. According to the O^2 design guidelines, it has to be performed as promptly as possible to reach an efficient online data reduction and to enable successful physical triggers that will ease later data analysis. The final goal is to have a primary vertex estimation representing the optimum between the elapsed time in the calculation and the precision reached.

An efficient reconstruction is able to find the position of each collision, also in challenging circumstances with a lot of pileup, where collision data are related to the same bunch-crossing or possibly in the same acquisition *frame*: the minimal segment of data usually limited by a time duration whose inverse is the data acquisition rate. The objective is to identify the best trade-off between computing speed and precision by keeping the best efficiency possible. This motivates some *a priori* assumptions and simplifications, which are listed and described in the next sections, and an extensive description of the algorithm will follow.

4.1 Understanding the input data

In the ITS reconstruction chain, the primary vertex reconstruction is the first step of the workflow. Its main purpose is to provide a good measurement of the position to the tracking reconstruction algorithm: an iterative routine aiming at the identification and connection of all the clusters on consecutive layers that belong to the same track, to refit the trajectory and obtain an information on particle tracks based on ITS information only. The underlying idea in designing the vertex reconstruction process, is to find an algorithm conditioned by a small number of constraints that would either introduce biases or would affect the goodness of the final result. Then, depending on few *a priori* assumptions and boundary conditions, such an algorithm needs to be efficient and performing in terms of time spent in the computation.

The only primary vertex finder input is a stream of 3D positions of the signals left by charged particles on the sensors of the ITS, regrouped in clusters. Their position is referred to the global reference system illustrated in paragraph 1.3.2. In figures 4.1 and 4.2 two

distributions of the coordinates of the signals are presented, both the transverse and the longitudinal section of the inner barrel of the ITS, to better visualise their arrangement. The distributions extracted from a dataset containing 200 Pb–Pb collisions. It is interesting to observe the disposition of the cluster in the turbo-geometry of the ITS. The cross section

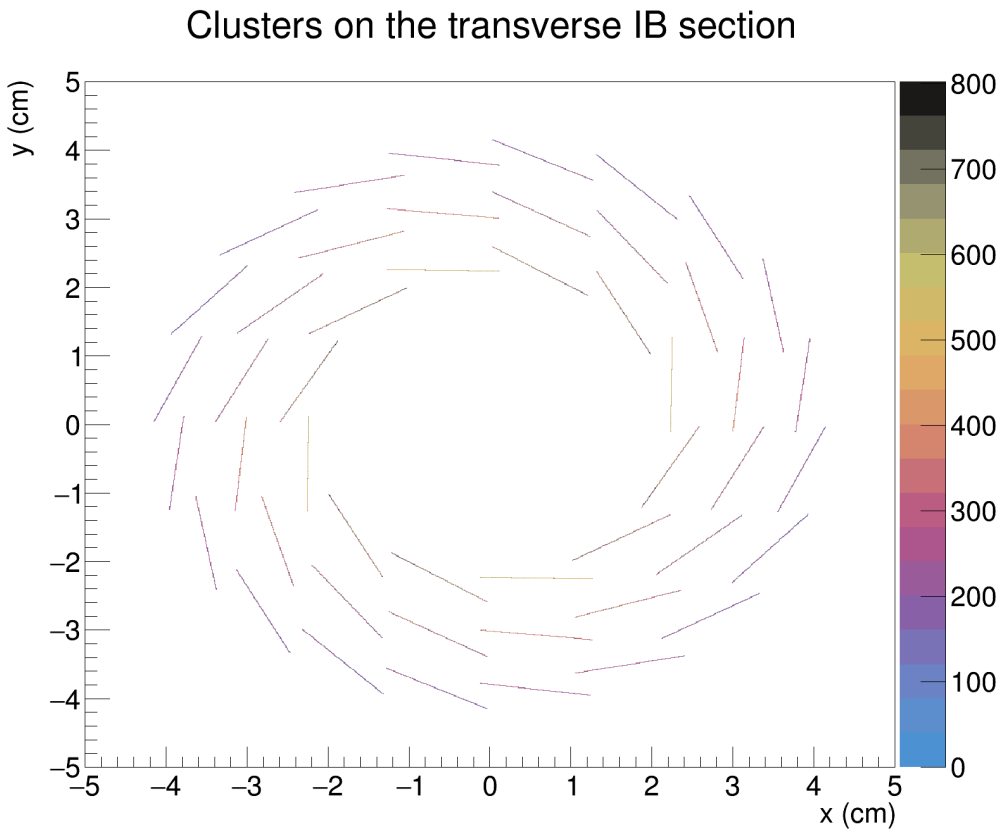


Figure 4.1: Distribution of the clusters position on the transverse (xy) plane. The *turbo geometry* is clearly visible from data.

of the active[ii] region of the staves is visible as the x, y coordinates of clusters distribute only there. The figure on the right, related to the longitudinal plane, has been produced by computing the radial coordinate of all the clusters, resulting in an integrated distribution with respect to the ϕ angle.

To develop a strategy for matching *correlated* clusters, i.e. corresponding to the same particle trajectory, some details related to the experimental conditions must be considered. Detectors are electronic devices and are subject to stochastic noise and malfunctioning in the case of some pixels which can be not operational or always providing a fake signal.

[ii] active region of a detector is its sensitive part, in this case it measures the passage of a charged particle.

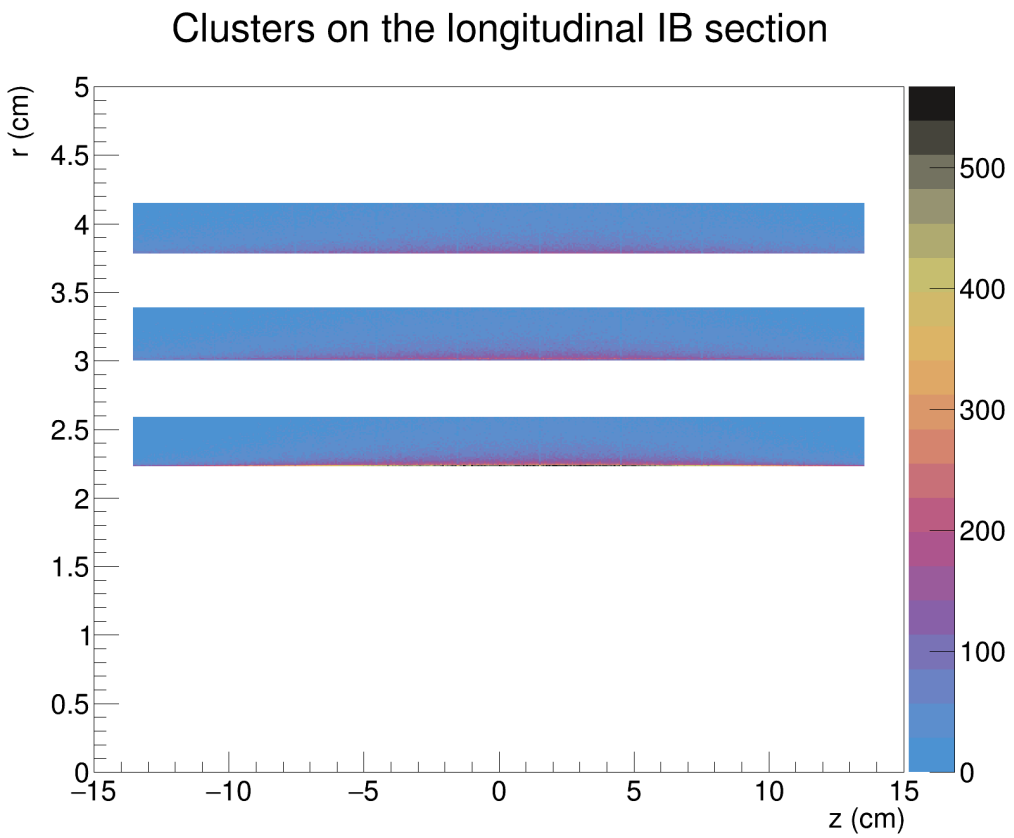


Figure 4.2: Distribution of the clusters position on the transverse (rz) plane, integrated over the azimuthal angle ϕ. The distributions of the clusters on the three different layers show an accumulation along the z coordinate centered in the nominal position of the interaction diamond (0,0,0).

While the random component of the noise is not detectable, hence it cannot be systematically removed, the map of *dead* pixels is compiled and updated in periods when there is no beam. That contribution is therefore subtracted to the input signal to be reconstructed to remove fixed noise. Another type of contribution to the goodness of the signal concerns the correction of the detector misalignment that takes into account all the small deformations of the experimental setup to correct the signal in such a way it is really referred to the ALICE reference frame. Based on these information, a complete set of selection strategies and variables could be devised, as explained in details in the following sections. It is relevant to take also into account both the Physics of the particles involved and their interactions with the detector. Among the most important there are:

- the **geometry of the detector**, as it constraints the measures to be placed on a specific volume in the space corresponding to its active region. As shown in previous figures 4.1 and 4.2, the detector is able to provide only tridimensional space points that correspond to the intersections of the trajectories of the particles with the active detecting volumes;

- the **symmetry** of the collisions and the **dynamic** of the particles produced, including the fact that they can decay. Knowing in advance the typical behaviour of a Pb–Pb collision is extremely useful to decide criteria to select clusters based on the expected topology of the particle propagation;
- the interaction with the experimental apparatus called **multiple scattering**. As shown in figure 1.9, multiple types of material constitutes the ITS, hence charged particles passing through interact with them via Coulomb scattering. The neat result is a deviation of the ideal helicoidal trajectory that, in the case of a silicon pixel detector like the layers of the ITS, it is measured as a displacement of the reconstructed tridimensional clusters of pixels. This effect is added to the systematic uncertainty related to the intrinsic resolution of the pixels themselves;
- the bending of the trajectory in the magnetic field caused by the **Lorentz force**. In the ALICE central barrel, particles travel through the magnetic field of 0.5 T. They are bended on the transverse plane with an orientation that depends on the sign of their charge. The curvature radius of the trajectory is proportional to the inverse of the transverse momentum (p_{T}), the projection of the momentum vector on the transverse plane.

Each of these factors plays a role in decision making procedure of designing the *pattern recognition* algorithm for the ITS primary vertexer. The approach to the first reconstruction step is to apply few geometrical selections, driven by the knowledge on the geometry of the detectors and symmetries of the ion collision.

Input data inspection. All the data used to develop and test the vertexer have been simulated using the new simulation program developed for the O^2 suite. At the time this work has been carried out, there was a limitation on the number of events available for the simulation. This is because the simulation tool was under intense development, and in the time window where it was needed there were some hard constraints on the memory used. Maximum number of events generated was 150 events with no selection on the impact parameter, for the Pb–Pb collision system. However, for the sake of this work it was enough to inspect the main scenarios, including the ones with multiple piled-up events. Larger simulation sample will in future be beneficial to provide a complete overview of the overall performance and behaviour of the algorithm also in less probable corner cases. For instance it will be interesting to inspect pile-up of many different of collisions with different particle multiplicity, for instance the superimposition of a high-multiplicity event, characterised by a large number of yielded particles, and an ultra-peripheral one, where the number of particles produced is much smaller and the capability to neatly distinguish between them and reconstruct both, is relevant. The purpose of the work is to port the execution on different architectures, keeping results consistent across implementations.

As previously described, particles are generated and propagated through the model of the experimental apparatus, simulating every physical interaction happening to the particles during their lifespan. Data acquisition from detectors is simulated as well as the digitisation of data and the reconstruction of clusters. The output of the simulation chain will eventually have a common format, shared with the one of real data. The final software

implementing every reconstruction algorithm will run on simulated data as it will on real ones. This has evident advantages in terms of consistency and code maintainability.

It is therefore useful to analyse the content of input data, to make some preliminary considerations. It helps not only in discussing algorithmic decisions and implementation strategies but at the same time to understand how the *turbo geometry* are manifested in the data. A set of 150 Pb–Pb events with no selection on the impact parameter were used, produced using Pythia8[75] generator included in the O^2 Monte Carlo simulation tool. The contribution from low pt QED electrons coming from the electromagnetic interaction of the Pb nuclei is added on top of these simulated events. Having extremely low transverse momentum, the signals of the QED electrons resemble the one of the noise, as their trajectories cannot be reconstructed using the approximation to tracklets. The (x, y, z) coordinates of the vertex were randomly extracted from gaussian distributions centered in $\mu = 0$ and $\sigma = 50$ μm for the position on the transverse plane, whereas with $\mu = 0$ and $\sigma = 5$ cm for z coordinate. Particles created by the event generator are transported through the detector geometry by the GEANT4[12] transport code. In this phase the interactions of particles with the detector material is simulated. At this stage, also the energy deposits of charged particles in the sensitive volume of the detector are computed. After the simulation, data are then processed by the digitisation workflow and the cluster reconstruction. The first is responsible to reproduce the moment when the ITS measures the energy deposit and produces a digital signal from simulated pixels is the first part of the reconstruction chain and processes the digitised signal to group adjacent pixels, the actual input of the reconstruction for vertices and tracks. In order to be sure to be working on detected particles, only the ones that interacted with the innermost layer leaving at least a cluster were considered, meaning that the detecting efficiency of ITS is assumed to be perfect. Using as reference the information available in the Monte Carlo truth, the charged pions p_{T} distribution of the tracklets that can be reconstructed is shown in figure 4.3.

In the subsequent three figures are shown some features of the data used as input for the development of the algorithm. Since the actual physical content of the data is something that can be decided and tuned at Monte Carlo configuration level, the aim here is only to provide a visualisation of some of the main aspects of the cluster data, neglecting every physical consideration. The three azimuthal distributions on three layers of the inner barrel are represented in figure 4.4. The number of entries decreases as the layer number increases because the three layers of the ITS have the same length but increasing radii. Therefore the innermost layer will see more particles since its coverage in terms of solid angle acceptance is larger. Another contribution to this reduction is caused by the QED contributions by electrons that typically do not reach the third layer because of their very low momentum (down to 1 MeV/c, with a corresponding curvature of ~ 6.6 cm). Usually, the density of such a contribution scales $\propto r^{-2}$ causing an average reduction of the integral of the distributions. It is important to understand the shapes of the distributions: particles are expected to be produced according to a uniform distribution, as expected from the cylindrical symmetry of a collision integrated to 150 events. This is almost the case, apart for interesting regular spikes in very narrow angular regions, repeated periodically, with different periods depending on the layer number. This comes straight to the geometry of the detector: the turbo geometry. By counting the number of spikes it turns out that they correspond exactly to the number of staves for every layer of the inner barrel [16]. The staves have very small regions of overlap in the sensitive area of the sensors, therefore a

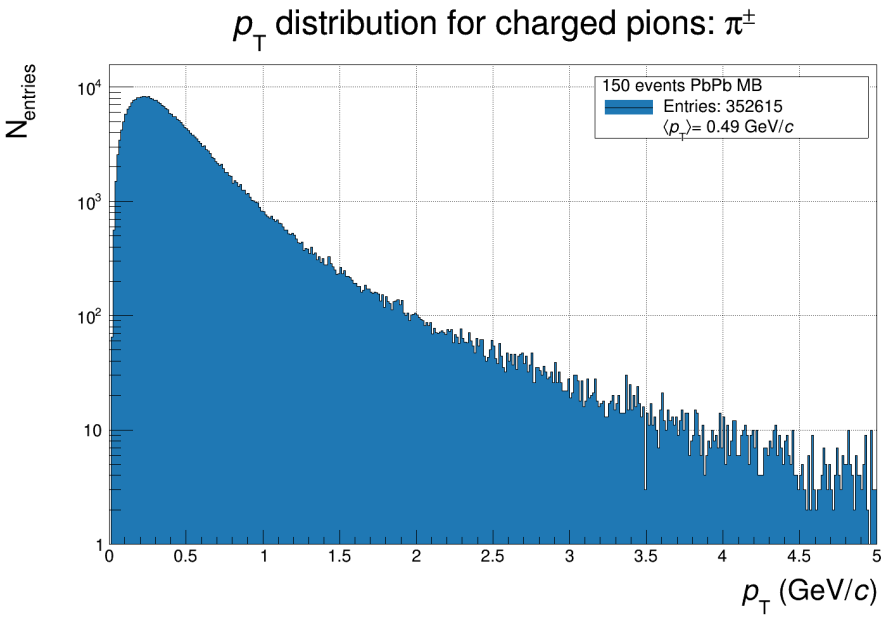


Figure 4.3: The p_{T} distribution for charged primary pions detected for simulated events. The $\langle p_{\mathrm{T}} \rangle = 0.49$ GeV/c is similar to the aforementioned measured value on real data by ALICE detector.

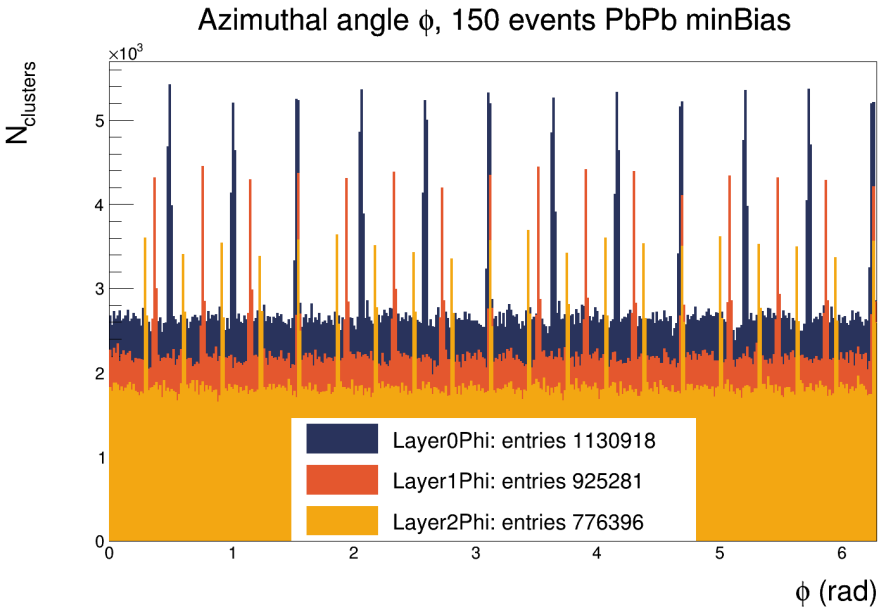


Figure 4.4: Azimuthal angle distributions of clusters for three layers used in primary vertex finding. The spikes represent the region of overlap between staves of every layer, typical of the *turbo geometry*.

particle with medium-high momentum is possibly detected twice, on the same layer. In addition, comparing the height of those peaks, it is clear how they are the double of the average value of the base uniform distribution, which suggests that almost all particles in those azimuthal regions have been detected twice.

The longitudinal distribution of the clusters (along z) is shown in figure 4.5. The

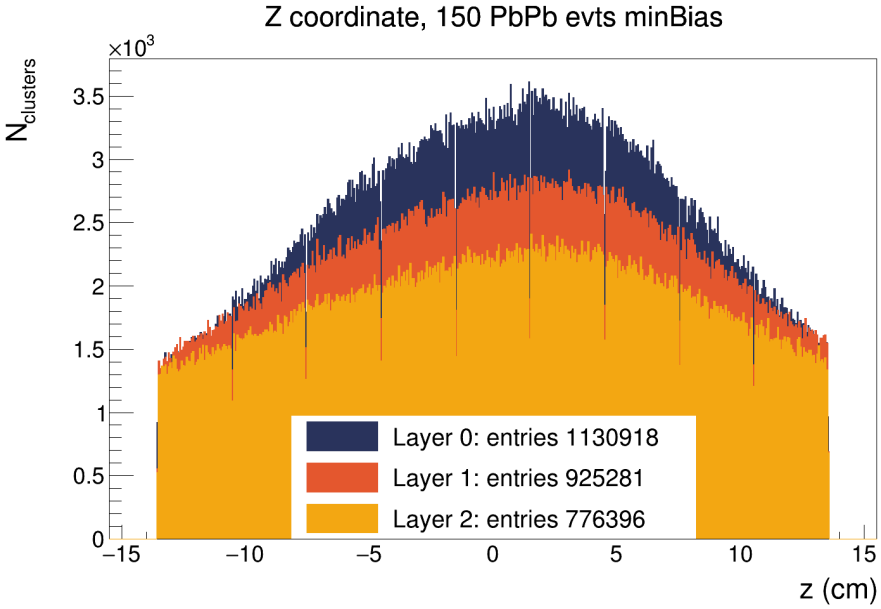


Figure 4.5: Cluster z coordinate distributions for the three layers of the inner barrel. The peaked distributions is in agreement with the majority of particles produced at midrapidity. The position of the maximum is aligned across three layers. Eight gaps are visible, in correspondence of the junctions between HICs bounded to the same stave.

distributions have a broad peak around $z = 0$ and that is because the largest number of particles is yielded at midrapidity. The slight asymmetries are due to a combination of the random generation of the z of vertices in a large region and the relatively small number of event integrated. The dataset is made of minimum bias events: they are taken from Pb–Pb inelastic scatterings, with no restriction on the impact parameter, the distance between the centres of the two colliding nuclei. The contribution from every event is therefore not the same and with a low number of events this may lead to shifts on the distribution. These distributions are expected to be very symmetric with larger statistics.

Another feature directly caused by the geometry of the detector that is clearly visible in figure 4.5 is the presence 8 gaps in the distributions, regularly spread along the *staves* and aligned between the different layers, corresponding to the insensitive regions of the 8 Hybrid Integrated Circuits (HICs) constituting each stave of the inner barrel.

Finally, figure 4.8 a two-dimensional plot of the radial coordinates of clusters versus their azimuthal angle is shown. The typical curved and periodic patterns are caused by two main factors:

- the clusters are on the plane surface of the staves, that in section can be represented by segments, neglecting their thickness;

- the segments are placed according the turbo geometry of the ITS described in 2.3; spanning around the azimuthal angle every stave will be encountered, and the cylindrical symmetry of the detector gives the periodic pattern.

Each stave is tilted with respect to the plane orthogonal to the radius that connects the centre of the ITS geometry to their midpoint as can be seen in figure 4.6. Considering

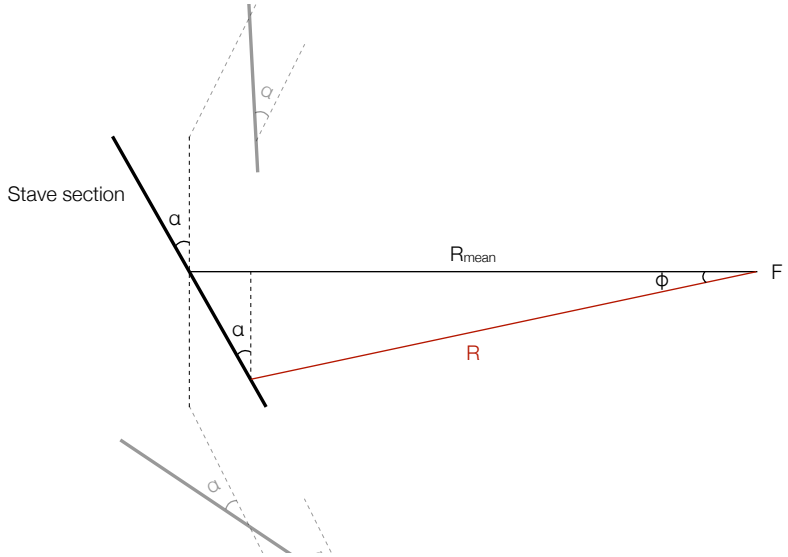


Figure 4.6: Each stave is tilted with respect to the plane orthogonal to the radius that connects the centre of the ITS geometry to their midpoint. The red radius connects clusters on the stave to the centre of the IB geometry. To different inclinations (ϕ) correspond different radii.

figure 4.7 as a reference, which represents a closeup on a stave of the inner barrel, α is the tilting angle of the stave. In this configuration, each layer in the IB is delimited by a r_{min} and a r_{max}, determined both by the thickness of the staves and the tilting angle. In figure 4.7 it is observable the phenomenon related to the variation of the radii of clusters that are separated by the same angle ($\Delta\phi$). It can be demonstrated that for each stave, radii of clusters detected by the staves are function of the azimuthal angle. Taking as a reference figure 4.6, the R segment represented in red can be evaluated as a function of R_{mean} and the angle they form (ϕ). Such a relation is:

$$r(\phi) = \frac{R_{mean}}{\cos\phi + \sin\phi \tan\alpha} \tag{4.2}$$

In figure 4.8 there are also accumulations of clusters in correspondence of lower radii for each layer. It is due to the binning of the histogram and to the fact that there is a dependency of the radii function $r(\phi)$ on the azimuthal coordinate that makes it not to uniformly distributed.

The development of the algorithm presented in this work focuses on the Pb–Pb collision system. It represents the greatest challenge both from a computational perspective in terms of number of operations and time spent in the reconstruction. Central events represent the

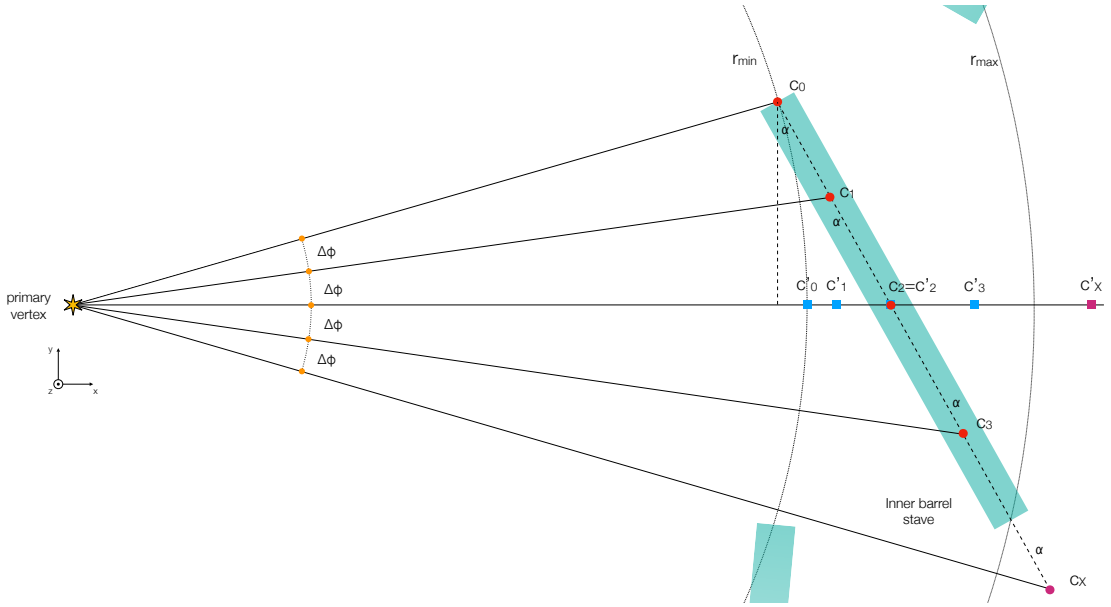


Figure 4.7: This schema shows how clusters on a stave tilted by an α angle, interleaved by the same azimuthal interval have radial coordinates that are function of the tilting angle. To be better visualised the radii are reported on the same line and are represented by square markers. Further clusters, closer to r_{max} are at a distance that is $\propto \sin\phi \tan\alpha$

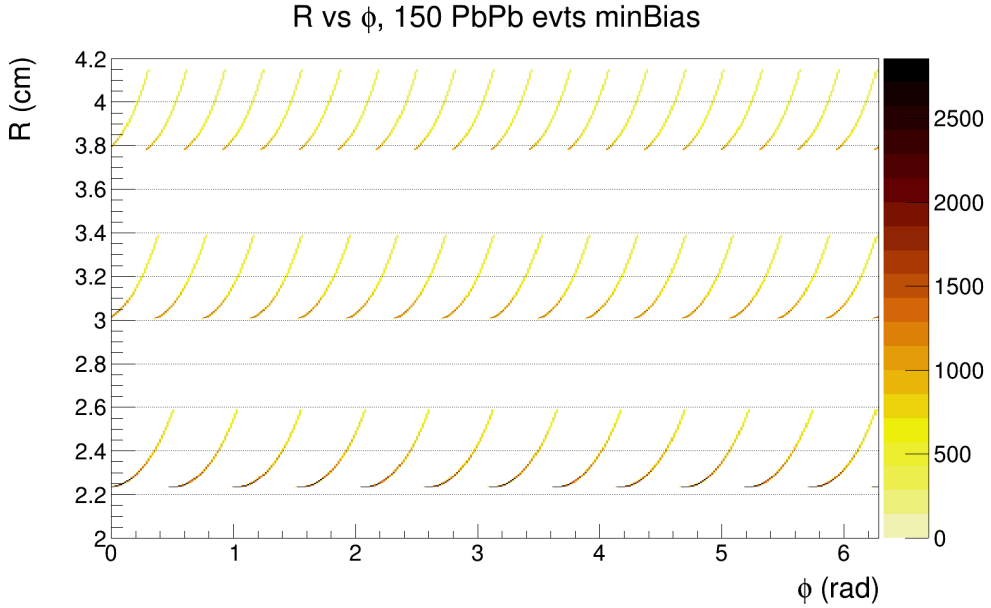


Figure 4.8: Distributions of R vs ϕ of the clusters for the three layers of IB. Every distribution decreases with the radius because of the stave inclination in the *turbo geometry*. The side closer to the z axis has greater angle acceptance.

most challenging example: the high multiplicity of yielded particles requires fast execution and agile management of the combinatorial complexity inherently related. With a ground-up approach in modelling the algorithm, it is important to understand the size of the data to be processed and to estimate the requirements for solving the problem. In other words, the known parameter expected to vary from frame to frame is the number of space points related to particles crossing the ITS, because the number of particle itself is subject to change across different interactions. The number of particles yielded in a collision is not uniform with respect to their direction starting from the primary vertex, in fact it follows a differential distribution with respect to the (pseudo-)rapidity (eq. 1.8): $\mathrm{d}N_{\mathrm{ch}}/\mathrm{d}\eta$. In general this also means that detectors displaced at different angles will record in average different number of produced particles. However, a rough estimation is provided by the weighted average $\langle \mathrm{d}N_{\mathrm{ch}}/\mathrm{d}\eta \rangle$ and it is sufficient to compute the scale for the algorithm. Such a quantity is the expected number of charged particles produced in Pb–Pb collisions with an energy in the centre of mass of $\sqrt{s_{\mathrm{NN}}} = 5.02$ TeV in the midrapidity region: $|\eta| < 0.5$. It is computed as the weighted sum over all the centrality classes and is the average number of charged particles produced considering the complete inelastic cross-section of the Pb–Pb collisions per rapidity bin.

Using results measured by ALICE[6], with W_{class} being the centrality binwidth and $\langle \mathrm{d}N_{\mathrm{ch}}/\mathrm{d}\eta \rangle \,|_{class}$ the corresponding estimation for that centrality class the value is:

$$\langle \mathrm{d}N_{\mathrm{ch}}/\mathrm{d}\eta \rangle_{\mathrm{tot}} = \sum_{classes} W_{class} \, \langle \mathrm{d}N_{\mathrm{ch}}/\mathrm{d}\eta \rangle \,|_{class} \simeq 611 \tag{4.3}$$

This number is relevant to get the expected average occupancy of the detector, and drives both the design of the algorithm and the selection tuning.

The upgraded Inner Tracking System has been designed to match the acceptance of outer detectors of the ALICE apparatus (TPC, TOF, EMCAL), covering an interval of $|\eta| < 1.22$. Although the maximum number of produced particles is in the midrapidity region, a safe assumption is to expect to measure $\sim$ 2500 charged particles per single collision at minimum bias, also including possible pileup.

Particles generated during the collision do not have an uniform p_{T} distribution. In fact, as stated in [5] their distribution is not uniform. Also, $\pi^{\pm}$ is the most abundant yielded species within the rapidity region $\mathrm{d}N_{\mathrm{ch}}/\mathrm{d}\eta \simeq 486$ with an average transverse momentum $\langle p_{\mathrm{T}} \rangle \simeq 0.532\mathrm{GeV}/c$ measured in minimum bias collisions. The design of a process to measure the interaction vertex in Pb–Pb, relies heavily on features mentioned above to apply some simplifications that aim at accelerating the routine.

4.2 Linear approximation of particle trajectories

The fundamental choice in the adopted approach is to work with a linear approximation of the early stages of particle trajectories, when they are still close to the interaction point. A numerical support to this approximation will be presented in section 4.2.1, by comparing the scale of the curvature radii of particles and the distance between pairs of clusters measured on different layers of the IB. In section 4.3.2 it will be described how the linear approximation establishes a fundamental hypothesis for the design of a tracklet-based primary vertex reconstruction algorithm starting from clusters of pixels. As a side note,

for different collision systems like *pp*, the algorithm will be potentially suitable under the same hypotheses, with some caveats. Some typical features, for instance the low particle multiplicity will present some more challenges of different nature. Small colliding systems present a sensibly lower number of particles produced ($\mathrm{d}N_{ch}/\mathrm{d}y \simeq 4.1$ per rapidity bin, at midrapidity) and lower average transverse momentum ($\langle p_{\mathrm{T}} \rangle \simeq 0.46\mathrm{GeV}/c$) that implies different selections and algorithmic choices where possible. The scope of this thesis does not fully investigate those systems, although it is planned to use the same rationale also for *pp*.

4.2.1 Charged particles curvature in a magnetic field

Charged particles generated in each collision, mostly pions, travel through the Inner Tracking System and their trajectories are bended on the transverse plane by the presence of the magnetic field. As anticipated, its field lines are parallel to the beam direction (z axis), in the rapidity region covered by ITS.

Hereafter is derived the curvature radius R_c on the transverse plane of a particle of charge q as a function of its transverse momentum p_{T}. Let F_{T} be transverse component of the Lorentz force as a function of the transverse velocity v_{T}, normal to the magnetic field lines of $\vec{B}$:

$$F_{\mathrm{T}} = q v_{\mathrm{T}} |\vec{B}| \tag{4.4}$$

The force can be expressed in terms of the centripetal acceleration as:

$$F_{\mathrm{T}} = m \frac{v_{\mathrm{T}}^2}{R_c} \tag{4.5}$$

By combining the equations 4.4 and 4.5, the curvature radius can be expressed in terms of the transverse momentum and the magnetic field as:

$$R_c = \frac{p_{\mathrm{T}}}{q|\vec{B}|} \tag{4.6}$$

This equation of the curvature is expressed in terms of p_{T}, that means it is not necessary to know the mass, consequently the type of the charged particle. The relation between R_c and p_{T} is useful because allows the algorithm to perform some selections on the possible p_{T} associated to the tracklets constructed during the process, based only on geometrical considerations, for instance the width of an opening span in the azimuthal angle. This observation is instrumental, because the actual p_{T} of a reconstructed trajectory is an observable computed later in the reconstruction process, by the track finder and is therefore not available at this point.

It is important to demonstrate that the scale of the inner barrel of the ITS, compared to the scale of the curvature of the majority of yielded particles ($\pi^{\pm}$ at 530 MeV/c), allows for approximating most of the detected trajectories to **straight lines**. In other words, their average p_{T} is high enough to neglect the curvature effects of magnetic bending at a distance from the interaction vertex of the comparable with radius of the third layer of the inner barrel.

Considering a charged pion ($\pi^{\pm}$) with transverse momentum equal to the $\langle p_{\mathrm{T}} \rangle$ of the distribution related to Pb–Pb inelastic scattering[6]: 0.532 GeV/c, by means of equation

4.6, its expected corresponding curvature radius in the ALICE magnetic field $B = 0.5\text{T}$ is

$$\left\langle R_c^{\pi^\pm} \right\rangle_{\text{Pb–Pb}} \simeq 3.55 \text{ m}.$$

Figure 4.9 presents a sketch of the scenario: the red line represents the trajectory of a charged particle from the primary vertex outward, travelling throughout the ITS and the blue circles are the clusters it left in the detecting surface. Pairs of clusters on two adjacent layers of the inner barrel can be connected by means of tracklets.

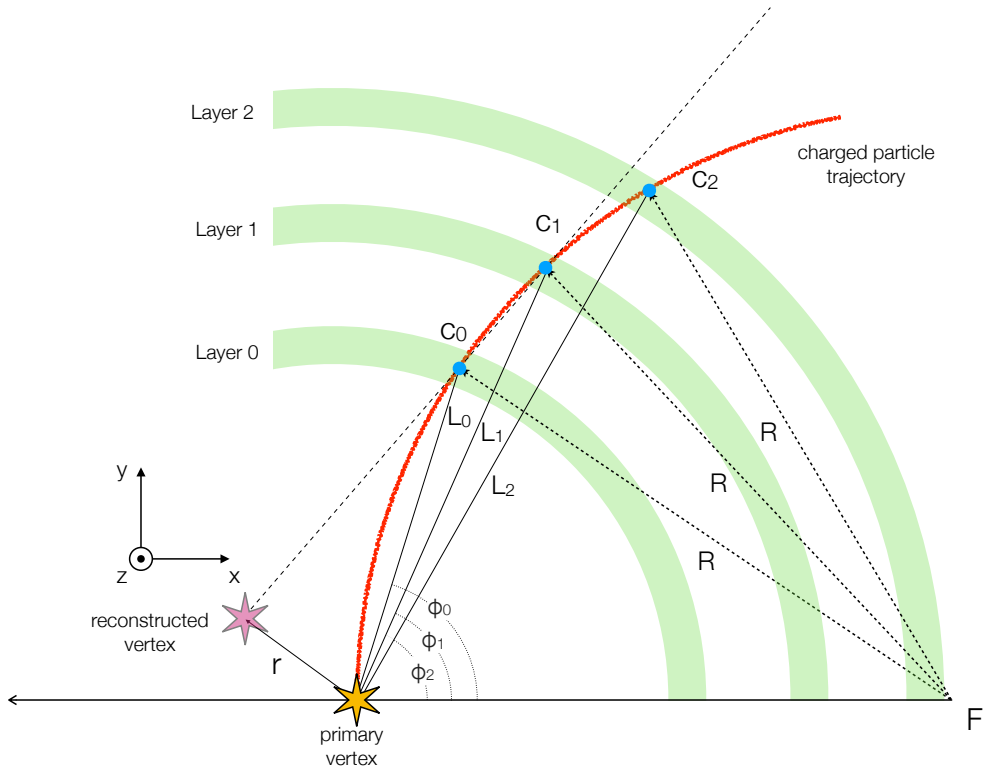


Figure 4.9: Charged particle crossing the three silicon layers of the inner barrel with a curvature radius R. Points c_0, c_1, c_2 are the clusters left on the detecting surfaces, their azimuthal coordinates are expressed by ϕ_0, ϕ_1, ϕ_2. Radii of the three layers are L_0, L_1, L_2. The distance on the transverse plane for the reconstructed vertex from the primary vertex is represented by vector $\vec{r}$

Considering schema in figure 4.9 the first step is to compute the azimuthal difference for clusters generated by a particle with a curvature radius $\langle R \rangle \sim 3.5$ m. Being L_i the radius of the i-th layer, the azimuthal coordinate of the cluster on that layer, as a function of the curvature radius is obtained by the trigonometric relation:

$$\frac{L_i}{2} = R \cos \phi_i, \tag{4.7}$$

from which it can be obtained the expression for the ϕ_i angle. For large curvature radii that satisfy the relation $R \gg L_i$, compared to the scale of the inner barrel, the expression

for ϕ_i can be expanded to the first order:

$$\phi_i = \arccos \frac{L_i}{2R} \underset{R \gg L_i}{\simeq} \frac{\pi}{2} - \frac{L_i}{2R} \tag{4.8}$$

The azimuthal coordinate difference $\Delta\phi_i$ for clusters belonging to the same particle trajectory and placed on consecutive layers is

$$\Delta\phi_i = \frac{L_{i+1} - L_i}{2R}. \tag{4.9}$$

Considering now $\Delta\phi_0$ that involves the first two clusters, known the radii of the first two layers ($L_0 = 2.34$ cm, $L_1 = 3.14$ cm), their difference is $\Delta L_{01} = 8 \times 10^{-3}$ m. Therefore, $\Delta\phi_0$ is

$$\Delta\phi_0 \sim \frac{8 \times 10^{-3}}{7} \simeq 1.14 \text{ mrad}. \tag{4.10}$$

This consideration enables the design of a *tracklet-based* vertex reconstruction and introduces the **tracklets**, or track candidates: straight segments connecting two clusters of pixels, representing the shortest linear path linking them in tridimensional space. In the end, these approximations will characterise the best the efficiency and accuracy level reachable by the vertex reconstruction. If now the tracklet connecting the first two clusters is prolonged inwards with a linear path, it will have a distance of closest approach (DCA) on the transverse plane from the primary vertex. Such a DCA represents the minimal spatial displacement of the two vertices, and is an indicator of how large is the error in using a linear approach with tracklets. It is important to consider that the final position of the vertex will be computed out from the average contribution of multiple tracklets, and the estimation will be smoothed by contributions that will have a uniform azimuthal distribution (resembling the symmetry of the particle production in Pb–Pb collisions). In the limit of $L_i/R \to 0$ the propagated line and the radius of the detector L_0 are parallel. The angle $\Delta\phi_0$ and the angle formed by vector that starts from cluster c_0 and points the primary vertex and the segment of the track that connects the cluster c_0 to the reconstructed vertex, are alternate internal, therefore equal. Finally, the module of the DCA vector that separates primary and reconstructed vertex can be computed as:

$$|\vec{r}| = L_0 \sin \Delta\phi_0 \approx 26 \ \mu\text{m}. \tag{4.11}$$

This quantity represents the error made in propagating a straight line from two clusters instead of computing the circle connecting them together with the one used to validate the tracklet. It is important to notice how the formula in 4.12, derived from 4.9 by substituting the curvature radius with its expression according to the Lorentz's law,

$$\Delta\phi_i \simeq \frac{0.3|\vec{B}|(L_{i+1} - L_i)}{2p_{\text{T}}} \tag{4.12}$$

provides a connection between the transverse momentum of a charged particle trajectory and the azimuthal displacement of its clusters on two cylindrical layers. Such an equation is also useful to correlate the azimuthal selections on $\Delta\phi$ of clusters and the minimum p_{T} allowed to be included in the vertex reconstruction, as will be later discussed. Compared

with the resolution on the primary vertex required by the tracker it is two orders of magnitude smaller, therefore the contribution of this approximation to the final estimation is negligible.

The choice of which clusters are used to create the tracklets is done using the aforementioned selections as a discriminating value. Tracklets sharing the same middle cluster are then compared to validate each other according to an alignment selection, performed later. The process is explained in detail in next section, for the moment is only interesting to assume that the innermost tracklet has been validated and its linear prolongation is used as input for the vertex position estimation. It can be demonstrated that a similar approximation is valid also for the reconstruction of *pp* collisions. In this kind of events the average transverse momentum of yielded pions is lower than in Pb–Pb: $\langle p_{\mathrm{T}} \rangle = 459$ MeV/c[5], but still high enough to guarantee the effectiveness of a straight line approximation. The curvature radius associated to that transverse momentum is: $R_c \simeq 3.06$ m and the ratio

$$\frac{\langle R_{L_2} \rangle}{\langle R_c^{\pi^\pm} \rangle_{pp}} \simeq 1.2 \times 10^{-2} \tag{4.13}$$

4.2.2 Contribution from detector resolution

For both collision systems it is necessary to evaluate whether and possibly how much the resolution of the detector can affect the approximation done. A good estimator is the error on the azimuthal angle ϕ associated to the measure of a cluster. The error on the angle is obtained by from the error associated to the single pixel size, divided by the radius of the innermost layer, to compute the largest contribution possible. Considering the size of a MAPS pixel rounded to $25 \times 25 \mu\mathrm{m}^2$, the associated error on the position measurement is $\sigma_p = 25/\sqrt{12} \simeq 7.2 \mu$m. The tangent of the angle is therefore approximated to the angle itself and its value is reported in equation:

$$\tan\phi \sim \phi = \frac{\sigma_p}{R_{L_0}} = 3.1 \times 10^{-4}. \tag{4.14}$$

This value, compared with the error associated to the straight line approximation computed in 4.10 is a negligible source of uncertainty to be considered.

4.3 Primary vertex reconstruction algorithm

This section will study and discuss the algorithm for the primary vertex reconstruction. Its development has been tuned for the reconstruction of trajectories of charged pions generated in Pb–Pb collisions. More precisely, it will rely on all of those charged particle that are generated with a $\langle p_{\mathrm{T}} \rangle$ intense enough, compared to the scale of the involved parts of the ITS that are used for the measurement, to approximate their trajectories through the innermost section of the ITS to straight lines. It will be contextually shown how, because of the impossibility to associate clusters on consecutive layers that belongs to the same *track* (trajectory), it is not obvious to have correct matches between consecutive clusters. Strategies to perform those selections will be discussed and benchmarked, also presenting some ideas that had been studied and discarded whenever it is interesting to investigate the motivations. Finally, also the complexity of the proposed algorithm will be studied, with

an eye on possible approaches that allow for further optimizations and implementations on diverse architectures like Graphic Processing Units (GPUs).

4.3.1 Input data pre-processing and access interface

The primary vertex reconstruction begins right after the computation of the coordinates of centres of mass of the *clusters* of pixels. Those positions are referred to the *tracking frame*, a special coordinates frame local to the sensor that is obtainable from the global frame (the one of the laboratory or accelerator) by means of a rotation about the z axis and a translation that moves the x axis normal to the sensor plane. Different transformation between inner barrel and rest of ITS will be used, because of the turbo geometry. The whole ITS reconstruction, consisting by primary vertex finder and tracking, uses the global coordinate frame, therefore clusters coordinates are transformed to this frame. The second operation, right after the transformation, is to compute their position in the cylindrical coordinate frame such as every cluster structure stores its coordinates in both cartesian and cylindrical global frames. The third operation is to distribute the clusters across seven data *containers*[iii], one for each layer of the ITS, in a temporarily unsorted fashion. As it will be better illustrated in section 4.3.2, during the *trackleting* step it will be necessary to perform many lookup operations to find matches between clusters on adjacent layers. It means that to find a specific cluster in the container, any routine will likely be running through all the clusters testing some specific targets with binary operation on them, one by one. Within a contiguous dataset of size N, this is an operation of complexity $O(N)$, meaning in best case it will find the match at first test, in the worst it will iterate on all the N clusters in the container, right before finding the correct match. To ease the lookup process, they are sorted inside each container according to two parameters, corresponding to their z and ϕ coordinates. It will be possible to quickly access them using a discrete index, described more into details in equation 4.16. Its value keeps into account the bi-dimensional disposition of the clusters into a one-dimensional container. The third radial coordinate is not considered during the sorting, since there will not be any lookup nor selection based on that parameter. Considering the azimuthal angle and the z coordinate of the clusters, data can be represented as a set of bi-dimensional points located on the surface of a cylinder whose radius can be chosen as the average radius of the clusters on a specific layer. Since the cluster container is a unidimensional *array* of data, an effective strategy is to introduce an intermediate discrete grid, hereafter called an **index table**, to serve as a proxy interface to access rectangular discrete subset on the surface of the bi-dimensional representation. Let's suppose to unfold the cylindrical surface corresponding to data of layer N, with length L_N to a rectangular $\Sigma(z, \phi)$ surface and to partition it using a discrete grid of $Z_{bins} \times \phi_{bins}$. Each cell of the grid is $\Delta z = L_N / Z_{bins}(cm)$ long and $\Delta\phi = 2\pi/\phi_{bins}(rad)$ high. The goal is to distribute across all the cells in the index table, the information about clusters sorted according indices corresponding to the indexing used to list row and columns on the grid. To compute the row and column integer indices (i, j)

[iii] Different container implementations might have different names and layouts in memory; for the moment the terminology tries to be as computing language-independent as possible

of the cell σ, in which a generic cluster c is located into, the following relations are used [iv]:

$$\begin{aligned} \sigma_i &= c_z \bmod Z_{bins} \\ \sigma_j &= \lfloor c_z / \phi_{bins} \rfloor \end{aligned} \tag{4.15}$$

For simplicity, in the algorithm lookups two indices are employed, but in most of the computational architectures the underlying array is unidimensional, as anticipated. In other words, every lookup operation accesses a cell in the table using two indices, but then it is required to transparently translate the query to a lookup over a one-dimensional array. With l being the index of the array position, the definition is reported in relation 4.16

$$l = i + j\phi_{bins}. \tag{4.16}$$

These definitions are used also to perform selections of subset of clusters delimited by rectangular Z, ϕ selections, using single indices on a two-dimensional interface. These selections are also called *regions of interest*.

After cluster sorting, the index table is compiled by filling each cell with the index of the first cluster that fits in. Since the clusters are sorted, each of them that would fit in the cell c_n has an associated index greater or equal to the index stored in cell c_n and lower than the index stored in the cell c_{n+1}. In case the cell is empty due to the lack of associated clusters the index stored in the previous adjacent cell is repeated automatically. Index tables are ultimately designed to provide delimiters for range-based iterations[v], which means in general, that ranges are always delimited by the indices contained in a continuous sequence of cells between two delimiting bounds. This approach is very effective in partitioning the data sample, but does not allow accessing the single cluster, only ranges.

It is straightforward to select subsets of adjacent index table cells and to perform *neighbour* lookup as a function of two indices on the table. A two variables function to converts lookups performed using two indices into an the array is required. A table with Z_{bins} columns and ϕ_{bins} rows, has dimension $\mathcal{D}$:

$$\mathcal{D} = Z_{bins} \times \phi_{bins} + 1 \tag{4.17}$$

The first cell always contains the index 0 which is the first element of the array of clusters, whereas the last bin is manually added, outside the rectangular area and stores the index of the last cluster. The schema in figure 4.10 describes how the index table is filled, starting from a sorted array of clusters. The grid grows in two directions: the abscissa is the z coordinate in the global frame and covers the $[-16.3, 16.3]$ cm interval, whilst the ordinate spans the azimuthal angle ϕ in the interval [0,2π) (rad).

The subsequent task is to match couples of clusters on consecutive layers, according to some criteria deriving from the selection the algorithm is going to perform. More precisely, geometrical selections on the surface of the cylinder are applied, to test all the clusters that belong to a delimited area.

[iv] The symbol $\lfloor x \rfloor$ corresponds to the *floor* operation, that always rounds down decimal part of x.

[v] For instance, any *for loop*

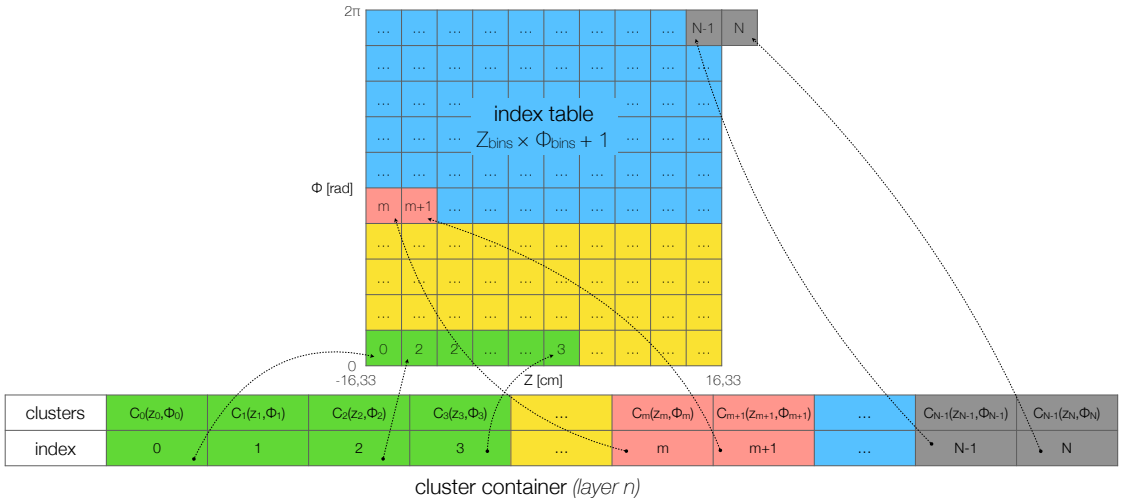


Figure 4.10: The index table grows in two directions: the abscissa is the z coordinate in the global frame and covers the $[-16.3, 16.3]$ *cm* interval, the ordinate spans the azimuthal angle ϕ in the interval [0,2π)[rad]. The first cell contains 0, which is the first element of the array of clusters, the last bin stores the index of the last cluster.

4.3.2 Tracklet finder

To identify all the possible straight lines available in every iteration of the vertexing workflow there are two main steps: called *tracklet finding* and *tracklet selection*. A tracklet is a pair of clusters connected by a segment that represents a track "candidate". As previously demonstrated, in the specific case of the primary vertexer all the tracks treated as straight lines. The vertexing procedure aims to process the tracks to find if there is one ore more –in case of pile up– geometrical points to be considered as the origin of those tracks. Those points are the candidates to be the **primary vertices**.

The vertexing routine starts with a **tracklet finder**: the sequence of operations to computes all the possible tracklets, according to geometrical selections. In this work, the tracklet finder runs only on clusters that belong to two consecutive pairs of layers, $0-1$ and $1-2$. It starts from clusters on the $Layer1$, hereafter also called *middle layer*, and iterates over all the *target clusters* on the adjacent layers, searching for a match and create a tracklet. This operation is repeated two times: one for the innermost layer and one for the outermost layer of the inner barrel. From the azimuthal coordinate of the middle cluster, it is computed the ϕ bin on the index table of the adjacent layer, to identify the region of the detector where to find the clusters to be matched with. In the specific case of the primary vertexer there is no constraint on z for the selections, that is every target cluster on a row corresponding to the considered azimuthal bin is checked.

If figure 4.11 is used as a reference, the two grids are the projection of the index table to the unfolded surface of the layers. Three clusters on $Layer1$ are to be matched with all the available clusters on $Layer0$ or 2, delimited by a binned area. Given a $\Delta\phi$ selection, it is considered, on the target grid, the smallest area circumscribing the angular interval around the starting cluster position. In this way it is ensured that all possible matches that

could satisfy the condition are considered, where $\Delta\phi$ is the difference of the two azimuthal cluster coordinates.

$$|\Delta\phi| < \Delta\phi_{max} \tag{4.18}$$

Typical cases are there represented.

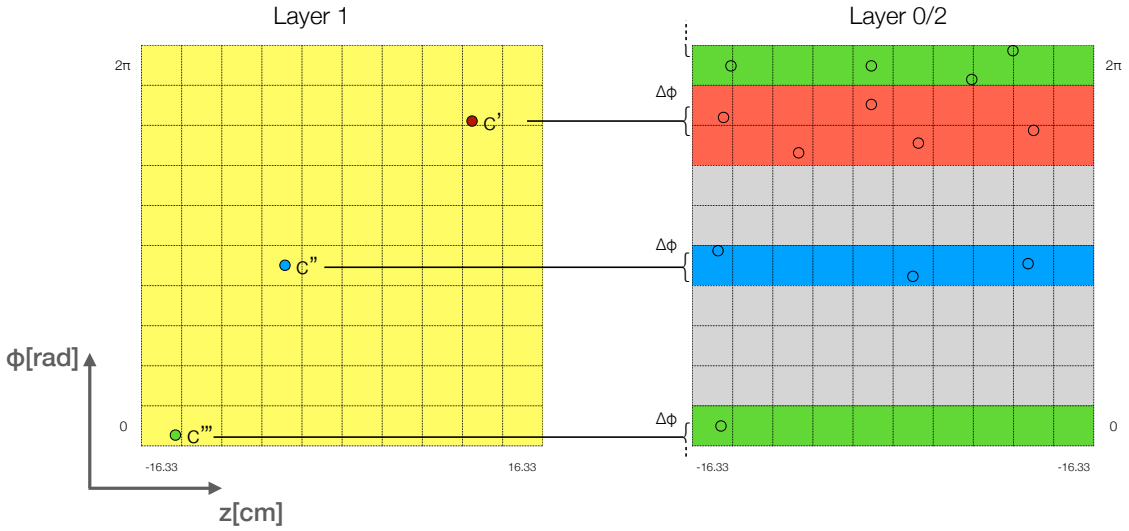


Figure 4.11: Cluster matches: if the distance of the middle cluster to the border of the index table cell is lower than $\Delta\phi_{max}$, also the adjacent row along the ϕ coordinate is included. If the middle cluster is close to the border of the table at a distance lower than the required cut, the periodicity of the table is automatically managed.

- Cluster c' is close to the grid border, by a gap that is smaller than the $\Delta\phi_{max}$ selection: to not lose correct matches with other clusters also the adjacent row is considered on $Layer0/2$.
- Cluster c'' is centered in azimuth and the $\Delta\phi$ cut si small enough, compared to the grid step, to allow for matching with clusters in just one row on the $Layer0/2$.
- Situation for c''' is similar to c', the access pattern to the index table has to consider the periodicity of the grid, to perform correct matches.

To validate a matching between two clusters, their separation in azimuth is therefore required to be below certain selection or, equivalently, that their offset in the transverse plane is small enough to consider them to belong to the same straight line. In chapter 5 it will be described how such a parameter has been optimised. It is quite straightforward to evaluate the complexity (better explained in equation 4.20) of the single trackleting step. Each execution iterates over the clusters on the intermediate layer only once. Then, for every cluster it is performed a selection of possible matches on the adjacent ones that generally depends on the ratio between the cut applied for the ϕ selection and the cell width. The latter is generally configurable in terms of azimuthal and longitudinal (along z axis) width, meaning the distance of the indices stored in adjacent cells of an index table

can be varied. For the azimuthal selection $\Delta\phi$ a number of cells on the target index table is chosen to be sure to iterate over all the clusters that are ideally stored in a given $\Delta z, \Delta\phi$ partition of the layer. In case the selection are larger than the cell size, multiple cells are considered, and iterations are performed over all the clusters subtended by the cells. The resolution on the cluster selection is therefore approximated to a multiple of the cell sizes. This is optimal because it is easier to perform the check on cluster parameters one by one instead of creating a too granular index table, perhaps with a step proportional to the applied selections, eventually nullifying, in case of too fine selections, any advantage in using the index table. Considering N as the total number of clusters spread across three layers (inner barrel only) and k_i the number of clusters on layer i is:

$$N_i = k_i N, \quad k_i \leq 1 \tag{4.19}$$

where k_i is the fraction of clusters belonging to the i^{th} layer and $\sum_{i=0}^{3} k_i = 1$. For each pair of layers, all the clusters in the middle layer and a fraction of the clusters on the second layer have to be considered. The number of the clusters to be matched on $L_j, j \in (0,2)$, is N_j/s_j where $s_j \geq 1$ is a factor that represents the equivalent share of the clusters corresponding to the index table selection, and grows proportionally to the inverse of the selection applied.

Now the complexity $T(N)$ of the tracklet finding can be calculated as

$$T(N) = \sum_{i=0,2} N_1 \frac{N_i}{s_i} = \sum_{i=0,2} \frac{k_1 k_i}{s_i} N^2 = O(N^2) \tag{4.20}$$

A similar calculation is done to estimate the eventual memory occupancy of the obtained tracklets. Keeping into account that not al the N_j/s_j clusters found a match, since some have been possibly discarded in case of iterations on clusters on the boundaries of the cells, it can be further defined a coefficient f_j: $1 \leq f_j \leq s_j$ to describe the result of selection imposed by equation 4.18. Best case scenario it is equal to s_j, although in general this is not true. The memory occupancy $S(N)$ is described by the formula:

$$S(N) = \sum_{i=0,2} g_i N^2 \times \text{sizeof}(\textit{tracklet}) = O(N^2) \quad \text{where} \quad g_i = \frac{k_1 k_i}{f_i} \tag{4.21}$$

In total, the tracklet finder runs twice for every ROframe one time for matching layer 1 with layer 0 and one to match layer 1 with layer 2. The estimated duration is similar and scales with the square of the number of clusters. Found tracklets are then stored into unidimensional data containers described by two lists of indices that store the number of found tracklets for each cluster. It is later possible to access all the tracklets associated to the same cluster, both in the inner and the outer pair of layers. This solution is ideal to accommodate both serial and parallel implementations of the tracklet selection step.

4.3.3 Tracklet validation

Tracklets obtained are the result of a combinatorial process that just considers pairs of clusters displaced in the same azimuthal acceptance window. The validation of the accepted tracklets is done via the **tracklet selection** step, that aims at selecting tracklets

found on layers 0 and 1, by validating them with the tracklet found on layer 1 and 2 that shares the same cluster on layer 1. Validating a tracklet requires three constraints on the two adjacent tracklets:

- they must **share a cluster on the middle layer**, to enforce the continuity. This is easy to check, since there are two index lists that are used for tracklets bookkeeping. Accessing the i-th position of the index list container provides boundaries to where to find the associated tracklets with cluster c_i
- they must have a **relative inclination on the transverse plane below** $\Delta\phi_{max}$ threshold. The cut applied is the same used to validate pairs of clusters on the transverse plane. This ensures that matches assigned to tracks with transverse momentum lower than the one determined by the $\Delta\phi$ selection for clusters are avoided.
- they must have a **relative inclination on the** r, z **plane below** a given threshold. On this section the magnetic field has parallel lines, and does not bend the trajectory, so strict selections can be applied to ensure only pairs of tracklets laying on straight lines, neglecting the multiple scattering, are considered. Tracklets are also considered coplanar on r, z: $\phi_{c_0} = \phi_{c_1} = \phi_{c_2}$. Comparison is then computed counting the tangent of the angle of a tracklet with the z axis: λ such as:

  $$\tan\lambda = \frac{r_{c_{i+1}} - r_{c_i}}{z_{c_{i+1}} - z_{c_i}}, \quad i = (0,2)$$

  and the condition is requested in terms of tangent of that angle as

  $$|\Delta\tan\lambda_{01\to12}| < \Delta\tan\lambda_{max}$$

  to facilitate the comparison.

By accessing the index list of created tracklets, every tracklet from layer 0 to 1 (t_{01}) and from layer 1 to 2 (t_{12}) is accessible at the same time. Validation is done via two nested iterations: the first is over every t_{01} tracklet and for each of them a second iteration is carried out on t_{12} tracklets. The validation procedure described above is performed on each pair of tracklets, and the inner iteration is interrupted as soon as a positive match is found. Every t_{01} that shares a common cluster uses the same t_{12} to check the validity. It might happen that the same outer tracklet is use as "validator", but there is no objective criterion to chose one among the other. This leads in general to have the same cluster that is shared by multiple tracklets, and at this level this cannot be avoided with an objective criterion. The schema reported in figure 4.12 represent a typical iteration for the single t_{01} tracklet. Tracklets t'_{12}, t''_{12}, t'''_{12} are considered in sequence, with their corresponding angles. The first that matches the selection stops the iteration. If all t_{12} are evaluated and no match is found t_{01} is rejected.

It is worthwhile doing two considerations at this point: the first is that there is no need to look for the t_{12} tracklet that has minimum $\Delta\tan\lambda$ deviation from the t_{01} tracklet investigated. In fact, the only relevant check is the existence of one tracklet that matches the selections, to confirm t_{01} is a good candidate. Secondly, there is no objective consideration that allows the algorithm not to use the same tracklet t_{12} to validate two or more different tracklets t_{01}.

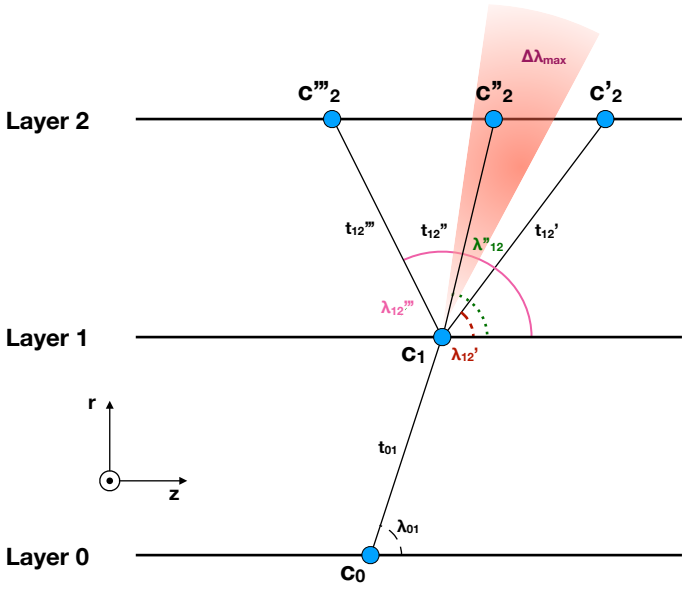


Figure 4.12: Starting from a tracklet t_{01} the algorithm iterates over t_{12}. The first validated tracklets interrupts the iteration and the next innermost tracklet is processed.

The complexity of this section is driven by two nested iterations over fractions of the pre-computed tracklets. Let M_i^j be the number of tracklets found in the *tracklet finding* phase associated to the i-th cluster c_i with a match on j-th layer and

$$k_i^j, \quad j = (0,2)$$

the fraction of the total tracklets from Layer j to Layer 1 associated with that cluster. The coefficients are such that $\sum_{N_{clusters}} k_i^j = 1$. The number of found tracklets is proportional to the square of the number of clusters, even if the number is heavily reduced by the selections. By increasing the number of clusters N on two layers the amount of tracklets grows as N^2 because the distributions are uniform, therefore $M \propto N^2$.

The complexity function $T(N)$ for tracklet selection is not analytically solvable, because the product of two coefficients k_i^0, k_i^2 is not known for each term of the sum. A safe assumption is to consider the majorant of the sum, which is N. Therefore the complexity function $T(N)$ is

$$T(N) = \sum_{i=0}^{N_{clusters}} M_i^0 M_i^2 = \sum_{i=0}^{N_{clusters}} k_i^0 M^0 k_i^2 M^2 \leqslant O(N^6) \tag{4.22}$$

In this work, as it will be described in section 6.2, tracklets to be validated store a minimal representation of the link between two clusters. Validated ones are used to construct the actual *lines*, to be used to fit the primary vertex position. The lines occupy larger memory, because of the larger information stored, such as the starting cluster and the three direction cosines. If αk_i^0, with $\alpha \leq 1$, represents the fraction of the tracklets

associated to the i-th cluster on layer 1 and a clusters on layer 0 that *had been validated* by the algorithm, the memory occupancy function $S(N)$ for lines scales as

$$S(N) = \sum_{i=0}^{N_{clusters}} \alpha^0 k_i^0 \alpha^1 k_i^1 M^2 \times \text{sizeof}(line) \leqslant O(N^4). \tag{4.23}$$

Each element in the sum is proportional to the square of the number of tracklets, therefore with the fourth power of the number of clusters. In the worst scaling scenario each coefficient is equal to 1, therefore their sum is N. It can be assumed N^4 Complexity functions and memory footprint estimations are generic expressions to quantify theoretical scaling of the problem. In real life, the selections applied in the various reconstruction phases, represented by coefficients k_j^i, and the maximum number of charged particles yielded in the single collision, proportional to N, neglecting the efficiency of the detector, can sensibly reduce the theoretical scaling of these functions.

4.4 Vertex finder

The final step is the **vertex finding**, that is the process in charge of extrapolating the information about the spatial position of the primary vertices on the processed ROframe. Previous phases already constructed tracklets by selecting clusters with certain characteristics, based on the previous physical and statistical knowledge of collision geometry and kinematics. Cluster matching and tracklet selection are not enough to ensure that all the selected tracklets are actually segments of particle trajectories. Even more importantly, it must be taken into account there can be more than a single collision occurring in the same ROframe, and the final workflow must be able to recognise them. During the development of this work different approaches had been evaluated, implemented and tested. Finally, two of them are available to the vertexing purpose, both have pros and cons that will be discussed in chapter 5. This type of algorithm falls into the category of *clustering problems* and aims to identify, aggregate and label clusters of entities in a given parametric space, by defining a *distance* function to compare pairs of entities. In the case of vertex finder the entities are tracklets or clusters of tracklets to be used to derive the primary vertices position from them.

4.4.1 Cluster-finding based method

First being implemented, this approach relies on the identification of *clusters of lines* that are pointing to the same direction, namely the primary vertex position by comparing lines to each others. It is worthwhile to mention that in this section the word "cluster" can refer both on the aforementioned clusters of pixels that identify a space point on an ITS layer and a group of lines that are matched together according some distance criterion in the tridimensional space. A schema of the workflow is reported in figure 4.13. The algorithm is based on a triangular loop, that is two nested loops in which the outermost loop spans over all the lines, from the first to the last. The inner iteration starts from the first line in the list and compares it with all the others. If the distance of closest approach (DCA) of the two line is below a threshold, a cluster is created, if not the iteration continues to next candidate. The **centroid** of the cluster is defined as the medium point of the segment

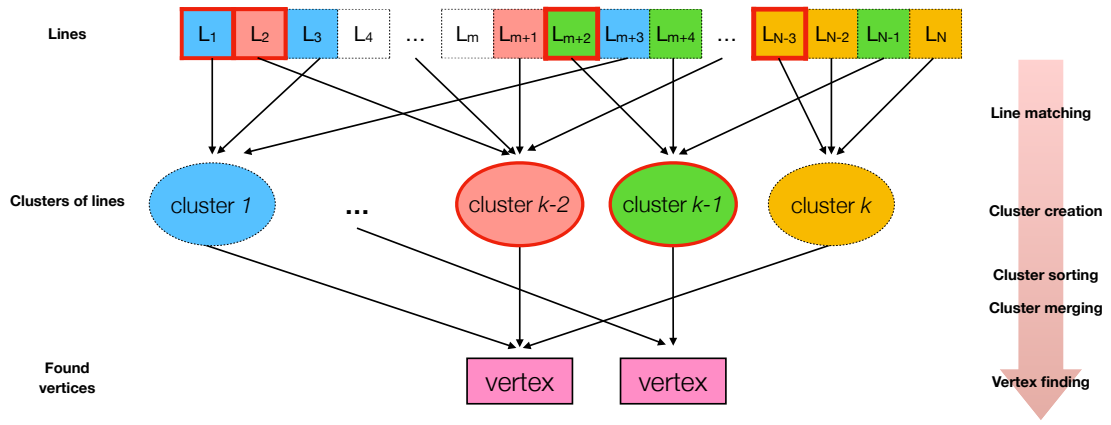


Figure 4.13: Schema of vertexer using cluster finding algorithm. In the case of lines, the red-squared boundaries highlight the starting lines of three different clusters. Lines are added to the first highlighted ones with an iterative schema. Finally, clusters of lines are sorted according the number of contributors and merged, starting from the first position (red-circled) and iteratively trying to merge all the others. Finally, centroids of the clusters are promoted to primary vertices.

representing the DCA vector. The accepted lines are labelled as "used" to be not counted in later iterations. A third loop runs on not labelled lines, testing if their minimum distance with respect to the centroid is below the same threshold used for matching couples of lines. If the test is positive the line is labelled as *used* and the centroid is updated as the centre of gravity of the cluster. The obtained cluster is stored and the primary loop moves to the next available lines. All lines not added to any cluster are discarded. Generally, more than one cluster can be constructed, depending on the cuts applied. The next step is to skim the clusters and merge them, wherever possible. They are sorted in descendant order, according to the number of lines contributing (*contributors*), then a triangular loop is used to iteratively compare clusters. Two clusters are merged if the projection on the z axis of their distance is lower than a threshold called *cluster cut* and the tridimensional distance is below another threshold called *pair cut*. The merging process consists of appending all the lines of the second considered cluster to the main one. The clusters are sorted to minimise the number of iterations over the contributors of the following clusters. Whenever a line is added to the main cluster, the centroid position is updated. At the end of the process either clusters are all merged together or more than one survived. Another selection is then performed on the remaining clusters: if only one cluster is reconstructed, it is promoted automatically to be the one where the primary vertex is extracted. In case of more than one cluster is reconstructed only the first one and the ones with a number of contributors greater than a selection on the number of contributors are kept. Finally, the centroids of each cluster are promoted to **primary vertices**.

Complexity of this step is driven by two triangular loops. The outcome of both line clustering and cluster merging is not predictable, so the worst scenario is assumed, where all the iterations of the outer and the inner loops of both parts are performed. Let N be the number of lines in input and M be the number of formed clusters, complexity functions

$T_{\mathrm{LC}}(N)$ for line clustering and $T_{\mathrm{CM}}(M)$ for cluster merging are defined as

$$T_{\mathrm{LC}}(N) = \frac{N(N-1)}{2} = O(N^2) \qquad T_{\mathrm{CM}}(M) = \frac{M(M-1)}{2} = O(M^2) \tag{4.24}$$

A positive aspect of this approach is related to the memory occupancy of both sections. For what concerns the line clustering, the size in memory of intermediate cluster artifacts increases just for the size of the information that points to the line that has been added, every other parameter is just updated upon line addition. Same is for cluster size after another cluster is merged, only labels referring to the used lines are forwarded. It must be also pointed that every line can contribute only once, with β representing the fraction of lines included in a cluster, only such as $\beta_i N \leq N$ with $i \in (LC, CM)$, depending on the step considered. Therefore, the memory occupancy functions for two previous steps are:

$$S_{\mathrm{LC}}(N) = \beta_{LC} N \times \mathrm{sizeof}(label) = O(N) \quad S_{\mathrm{CM}}(N) = \beta_{CM} N \times \mathrm{sizeof}(label) = O(N) \tag{4.25}$$

4.4.2 Histogram-based method

Due to the heavy serialisation that the previous approach introduced, better discussed in chapter 6, it has been necessary to investigate an alternative approach that was more suitable for a parallel approach. The rationale behind this technique here is different: there are more assumptions made on the scenario to be reconstructed and hypotheses made out from prior knowledge. The most important one concerns the final resolution on the position of the primary vertices. The core idea is to split the seek for the primary vertices into two parts. First is estimated the position of the beam on the transverse plane for the current ROframe, then it is used that information to select the line candidates to compute the vertex position. The first peculiarity of this technique is that in case of pileup the resulting position on the transverse plane would be the same for all vertices. In fact, the vertex position on the transverse plane is given by the estimation of the beam position. The possibility to distinguish among different vertices is therefore only based on their separation on z axis. This is mainly motivated by considering the size of the beam radius, which is usually narrow enough not to allow a vertex finding based only on the ITS to resolve that fine structure. As previously extensively explained, the goal of the whole primary vertex reconstruction is to provide a seed (a reasonable approximation) to subsequent reconstruction processes. This means that the position of vertices is eventually re-fitted by more accurate algorithms, based on more refined and complete information available. In the end, it is sufficient to demonstrate that the primary vertex finder is sufficiently precise to allow the rest of the reconstruction workflow for determining the correct results. The other observation comes from the comparison of the distribution on the transverse plane of the primary vertices coordinates with respect to how they are placed along the Z axis.

Starting right after the tracklet selection execution, this approach is completely interchangeable with the one presented previous section 4.4. I starts with two nested loops that run on the lines, evaluating the centroids for all possible line combinations. Such an execution is also considered a "triangular loop" since for the inner iterations the starting index boundary depends on the index of the outer, that increases at every outer iteration. Therefore, at each inner iteration the loop decreases its length. Two different one-dimensional

histograms are filled with the x and y coordinates of every centroid respectively. These are used to compute separately the position of the beam on the transverse plane. Maxima on both the histograms are found. Then the analytical coordinates are evaluated as the average of the position of a fixed number of bins symmetrically distributed around those maxima, weighted with their content. This allows for moving from a binned coordinate, the centre of the maximum bin, to a smoother and better refined estimation. The two-dimensional point is also defined as "*pseudo* beam position". Then all the distances of lines from the pseudo beam are calculated, and for all those lines whose the distance of closest approach (DCA) is below a certain selection specifically set and tuned for this approach, the z coordinate of the centroid is put into a third one-dimensional histogram. As a difference with respect to the transverse plane (x and y coordinates), where the simplification made is that all possible vertices are supposed to share the same x and y coordinates[vi], here the goal is to be able to separate more than one local maximum, in case of pileup. Therefore, an infinite loop is run on the histogram. Every iteration computes the maximum of the histogram, if it is greater than a predefined threshold it considers a symmetric binned window around it and evaluates the weighted analytical z coordinate the same way it is done for the transverse position. Then it sets all the considered bins to zero and it moves to next iteration. The value of each bin in this histogram corresponds to the number of lines that located a centroids in that specific bin, therefore the same value for the selection used in previous method for the contributors is applied. If no maximum is found with enough contributors the infinite loop breaks and the process terminates. In the end the process is made of a triangular loop plus a normal loop, the complexity as a function of N, number of lines in a pure serial implementation is

$$T(N) = \frac{N(N-1)}{2} + N = \frac{N(N+1)}{2} = O(N^2) \tag{4.26}$$

For the memory footprint function $S(N)$ receives contributions from three histograms, which are modeled as one-dimensional containers with a fixed size set by the number of bins, B. Therefore, regardless of the number of entries for each bin, the memory occupancy is rather a constant

$$S(N) = B \times \text{sizeof}(\textit{float}) = O(1) \tag{4.27}$$

4.4.3 Boyer-Moore majority vote algorithm: vertices purity estimation

Algorithm described in this section is used to measure the goodness of the reconstructed vertices and is fundamental for the comparison of the reconstructed results with the Monte Carlo data. The pileup, the background originated from wrongly matched clusters and the noise contribution, pollute the data used for final vertex fitting. To serve this purpose, using this algorithm introduces a system to assign a score to a reconstructed vertex that goes from 0 to 1. This parameter is defined as **purity**. To evaluate how much a single

[vi] Because of the impossibility of separating *piled-up* events on the transverse plane, it is just a matter of binning in x and y to improve the resolution on the coordinate of the vertices

vertex is "pure" a label assigned to every line used to fit it is considered. A process of line validation is performed by means of comparing the Monte Carlo truth linked to the clusters used to create them. A line is then *Monte Carlo*-validated if the two clusters constituting the tracklet have been generated by the same particle. If a line has been validated, the ID number of the event is extracted from the labels and assigned as identifier to the line. Otherwise, a special value is assigned to those lines which have not been validated. After this *tagging* phase, in the vertex finding final step, during the construction and validation of the vertices in output, all tags are used to vote the event ID of each estimated vertex. The final ID will be the most frequent in the sequence of line tags used in vertex finding. Assigning it to a reconstructed vertex enable the possibility to relate it with the Monte Carlo truth. Purity is therefore defined as:

$$P = \frac{N_{winner\ tag}}{N_{all\ tags}} \tag{4.28}$$

and defines a parameter to quantify the goodness of a vertex. Voting is done via the Boyer-Moore[28] algorithm, additionally the information of the vote is saved and used to calculate the purity. The Boyer-Moore algorithm is a simple routine meant to evaluate the more frequent element in a sequence, called *majority*. In this case, if it is a majority it always finds it, otherwise it is a tie, and the result is discarded or considered not pure. The pseudo code reported below describes a version that also keeps information related to the weight of a label (in this case an event ID).

Algorithm 1: Boyer-Moore majority voting algorithm pseudo code

Result: Returned parameter of the algorithm is usually just *label*. In this implementation also *votes* and *total_votes* is stored to weight *purity* of the result.

Initialise: $label = 0$, $votes = 0$, $total\_votes = 0$, $id\_map = \{\}$;
foreach $x \in \mathcal{I}$ **do**
  if $votes == 0$ **then**
    Assign: $label = x$ and $votes = 1$;
    Update: *id_map*;
    Assign: $total\_votes = total\_votes + 1$;
  else
    if $label == x$ **then**
      Assign: $votes = votes + 1$
    else
      Assign: $votes = votes - 1$
    end
    Update: *id_map*;
    Assign: $total\_votes = total\_votes + 1$;
  end
end
return *label*

The original version routine aims just at finding the winner whose percentage of votes is greater than the 50% of the total votes. This allows for having a very agile algorithm, with an $O(n)$ complexity scaling and a $O(1)$ memory scaling. However, in this context it

is relevant to compute also the purity of the result: this is done by adding a bookkeeping storage to save all IDs that appear in the process. Then the purity is computed by normalising the number the winning ID appeared to the total counter of votes as described by equation 4.28. In this way the complexity remains the same and the memory becomes $O(m)$, where m is the number of different IDs that are hit by the calculation.

More importantly, it enables effective testing on real scenarios, with continuos readout enabled, pile-up and fragment of events information spread across readout frames. The estimation of the efficiency is more complicated in a context of continuous readout with respect to a triggered mode, where the evaluation of resolution and efficiency is done on separate event data and with complete information. With the continuous readout, to compute a similar efficiency the whole reconstruction must be run and the single collisions reconstructed. Otherwise efficiencies only related to quantities included in the ROframe can be defined but usually they are related to parts of the reconstruction, as it is done in chapter 5.

4.4.4 Extend the tracklet-based approach to the *pp* collision system

As anticipated, while the primary vertex finder has been developed focusing on the most challenging Pb–Pb scenario, it can be demonstrated that the *pp* collision system, under similar hypotheses previously discussed, can be addressed by using the same tracklet-based approach. For the purposed algorithm to have at two good[vii] tracklets is sufficient to identify the position of the vertex with the required precision. This is valid since the precision is correlated to the amplitude of $\Delta\phi$ selections that guarantee straight line candidates only. As long as the vertexer can identify two good candidates it potentially will find a vertex. While this is most of the time the case for Pb–Pb collisions, it is more difficult to always guarantee two good tracklets in the optimal p_{T} window in *pp*. The lower multiplicity in proton collisions can however be also an advantage for the estimation of the severity of the applied selections. Considering the azimuthal angle selection, it has generally a double role: first is to ensure that the match of a hit with one on the next layer is done in a ϕ acceptance range that forces the R_c associated to the arc connecting them to be greater than a minimum value. Secondly, it is important to reduce the probability of wrong matches: hits that are not related to the same particle trajectory but are matched by this selection rule. The narrower is the selection and the best is the purity of the matches, until efficiency start decreasing too much. For *pp* collision the effect caused by wrong matches is mitigated by the low multiplicity. Therefore, to improve the efficiency in this system a preliminary consideration is to adopt wider selections in ϕ trying to cover a larger p_{T} window. The straight line approximation will produce results with worse resolution, but at the same time it must be pointed out that the track finder process can also work without an extremely precise vertex position, provided that at least a meaningful seed is produced by this step. The estimation on the primary vertex position will be re-computed after the whole track reconstruction (including tracking done by other detectors)

[vii] with "good" it is intended a tracklet correctly connecting clusters of pixels generated of the same particle

is completed. In that case tracks are reconstructed without any linear approximation, hence they provide much finer resolution on the position. The fast primary vertex reconstruction presented in this work is a mandatory input to more precise reconstruction algorithms. The *pp* reconstruction can therefore benefit from the same primary vertex reconstruction approach, with larger selections. Much critical scenarios, for instance where only one good tracklet is reconstructed, may require minor dedicated adjustments in the technique, to improve the overall efficiency of the vertexer, but are not covered by this work.

Next section continue to address the Pb–Pb case and will cover the tuning of the selections and will study the behaviour of performance as a function of the selections. Although it was performed on specific implementations (software), most of the discussions is kept independent from the technologies actually used. Treating the topic in a way that is mostly unrelated from technical implementation details makes it easier to focus on the numerical and statistical aspects. Details related on the performance on different hardware platforms, programming strategies and timing performance are presented in chapters 6 and 7.

Chapter 5

Calibration of the algorithm: selection variations

The algorithm has been developed by using Monte Carlo simulation data. The first collisions system that was used to test the algorithm is the Pb-Pb one, because it is more challenging in terms of time performance to be reached and for the high detector occupancy. It is known that a higher number of tracks in the event makes more difficult the reconstruction of tracks because of the huge combinatorial background. The possibility to compare reconstructed data with the Monte Carlo truth is crucial to refine the selections of in order to maximise the final vertexing efficiency and the resolution on the vertex position. From the comparison of the results obtained with the reconstruction with the full information available from simulated data it is possible to evaluate the effect both of algorithmic choices in design phase and the quantitative effects of the selection tuning on the final performance.

In figure 5.1 the distributions of $\Delta\phi$ (on the left) and Δz (on the right) are reported for correlated clusters in Layer 0 (L_0) and Layer 1 (L_1) of the ITS. The distributions are obtained by analyzing 150 minimum bias Pb-Pb events, same dataset is used for the tests described later and that were done to benchmark performance and selections.

5.1 Tracklet finder optimisation

The tuning of the $\Delta\phi$ selection translates in setting a minimum value for the p_{T} of the tracks. Depending on the range of $\Delta\phi$ selected, the ratio between the number of *fake tracklets* and the number of *real tracklets*[i] varies. It is useful to define a placeholder for the *trackleting efficiency* as the ratio

$$\varepsilon_{trackleting} = \frac{N_{real}}{N_{MC}} \tag{5.1}$$

[i] real tracklet: matching of two clusters corresponding to the same particle track. A fake tracklet is a spurious match of clusters with different Monte Carlo id.

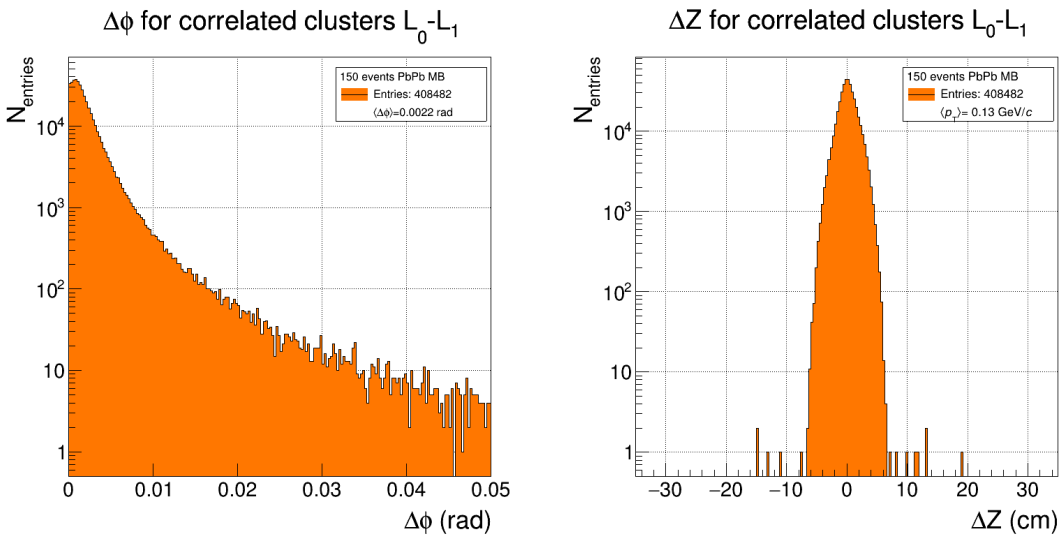


Figure 5.1: Distribution for $\Delta\phi$ and Δz for correlated clusters taken on adjacent layers. Distribution the left is very similar to the p_{T} distribution for pions, the most abundant species yielded, tail on the right is populated by charged particles with low momentum, such as electromagnetic contributions. The z distribution is peaked around the centre of the interaction diamond.

where N_{real} is the number of tracklets reconstructed with the tracklet finder, validated with the Monte Carlo (MC) truth. Hence, N_{MC} is the number of the matches artificially constructed by counting all possible cluster matches between clusters on consecutive layers associated to the same track identity. The latter quantity represents the maximum number of correct associations an algorithm, with no previous information on reconstructed tracks, could find. As long as a selection on $\Delta\phi$ is applied, the trackleting efficiency is never expected to be 100%, because of the automatic exclusion of some p_{T} interval from the matches. On the other hand, the *straight lines* approximation is effective to select valid tracklets only if a tight selection on $\Delta\phi$ is applied. The selection in ϕ is not sufficient to isolate the "good" candidates, namely to separate the signal from the combinatorial background. A further selection on the z coordinate of the cluster needs to be applied, in order to constrain the correct alignment for matched clusters. This effect clearly visible in both plots reported in figure 5.2. In particular, the distributions are reported with a red continuous line. Those distributions represent the number of tracklet validated with the MC check normalized to the number of tracklets found with different selections. It is clearly visible the low efficiency for such a set of tracklets because the ratio goes to zero for almost the whole $\Delta\phi$ range.

It can be noticed that by applying a very tight selection, the ratio rises up just to the 5%. The bars in light blue and the brown ones show the efficiency in detecting validated tracks when the tracklet finders is used between Layer 0 and 1 and between Layer 1 and 2, respectively. They starts dropping as the signal arises, meaning that selections are becoming too penalising. From these distribution is clear that to release the selection is

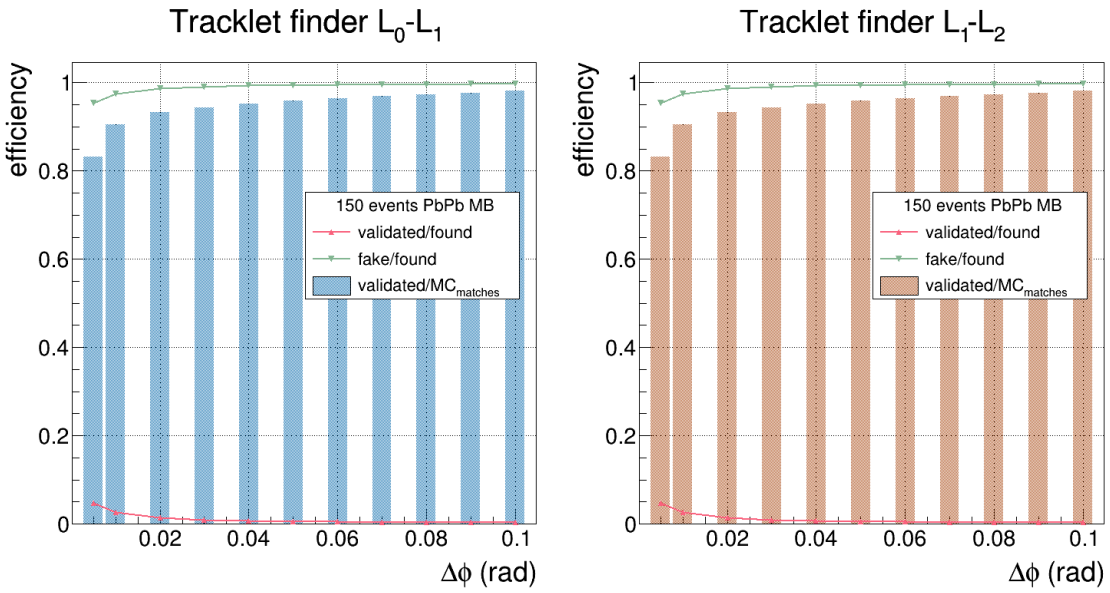
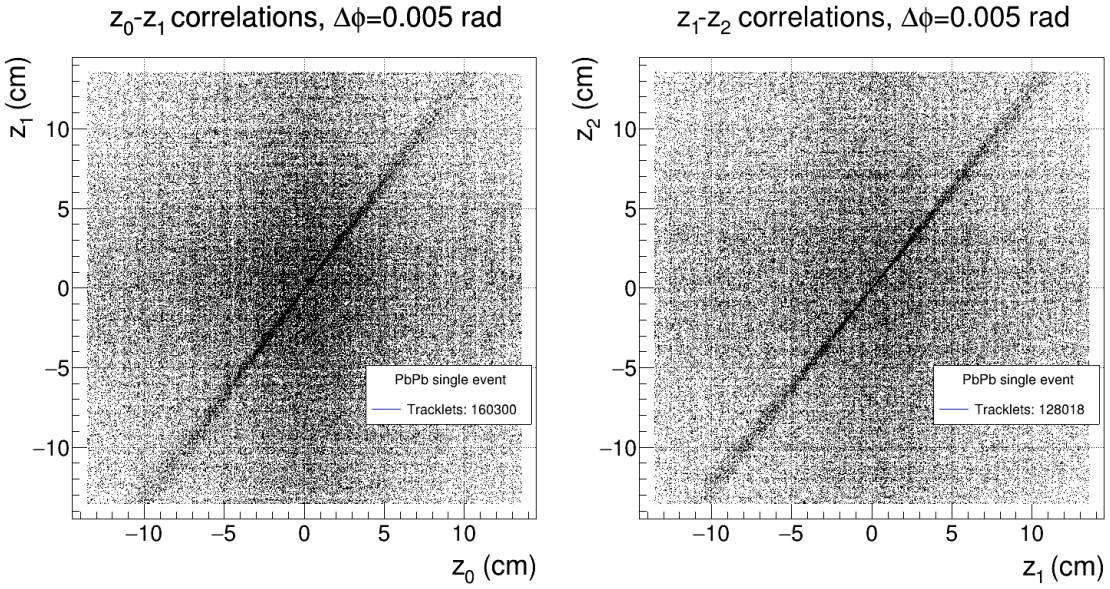


Figure 5.2: Efficiencies for tracklet finding applied to three innermost layers: figure on the left reports results for matches L_0 and L_1,figure on the right reports results for matches L_1 and L_2. Results are based on 150 minimum bias PbPb events. They look very similar and it is expected because the uniform distribution in ϕ cuts on both terms of the efficiency calculation.

not a good strategy because the efficiency start to drop dramatically. The combinatorial background is still very high, because no selection on the spatial prosecution of the tracklets is performed at this point. On the contrary, all possible matches in an acceptance window large $\Delta\phi$ and long as the whole active region of the inner barrel are considered. The green line represents the fraction of fake tracklets compared to the whole number of reconstructed tracklets. Coloured bars are the fraction of good tracklets found divided by all validated obtainable not applying any selection. After this first study performed with the tracklets, the best selections are determined by studying the efficiency and the resolution obtained for the whole vertex reconstruction process. Once it is chosen, the same selection will be also applied to the subsequent phase: tracklet selection, to further skim tracklets through a validation process. There is no a strong constraint in using the same selection, in principle they could be different and would be easy to add one more parameter to the algorithm. For the moment no particular issues that could be fixed by using two separate selection was encountered. As first approach the range $\Delta\phi = 0.005$ (rad) radians is selected and this corresponds to reject tracks with p_{T} smaller than 200 MeV/c. The aforementioned selection represents a good compromise between keeping a high (more than 80%) efficiency for the tracklets reconstructible and the reduction of the combinatorial background, which translates in a faster execution.

In figure 5.3 two scatter plots obtained after applying the $\Delta\phi$ selection to the data contained in one readout frame are reported. In such ROframe only information related to a single event (collision), hence no pileup is present. Specifically, in figure on the left 5.3a the

z coordinate on Layer 0 (z0) vs the z coordinate on Layer 1 (z1) for the pairs of reconstructed tracklets is reported. Figure on the right 5.3b shows the z coordinate on Layer 1 (z1) vs the z coordinate on Layer 2 (z2) for the pairs of reconstructed tracklets. The diagonals, due to



(a) Scatter plot for z_0 vs z_1

(b) Scatter plot for z_0 vs z_1

Figure 5.3: Scatter plots taken from the same ROframe with no pileup. In figure on the left are the z coordinate on Layer 0 (z0) vs the z coordinate on Layer 1 (z1) for the pairs of reconstructed tracklets are reported. The diagonals, representing the correlations, are visible only after the application of a selection in $\Delta\phi$.

the prevalence the correlated pairs of points, emerge from the combinatorial background only after the $\Delta\phi$ selection. This is a good visualisation of the existence of the signal embedded in a lot of uniform combinatorial background. By not applying any selection in azimuthal coordinates would have not been possible to highlight its presence. The slope of each diagonal depends only on the ITS geometry and it is determined by using the radii of each pair of layer in the following equation:

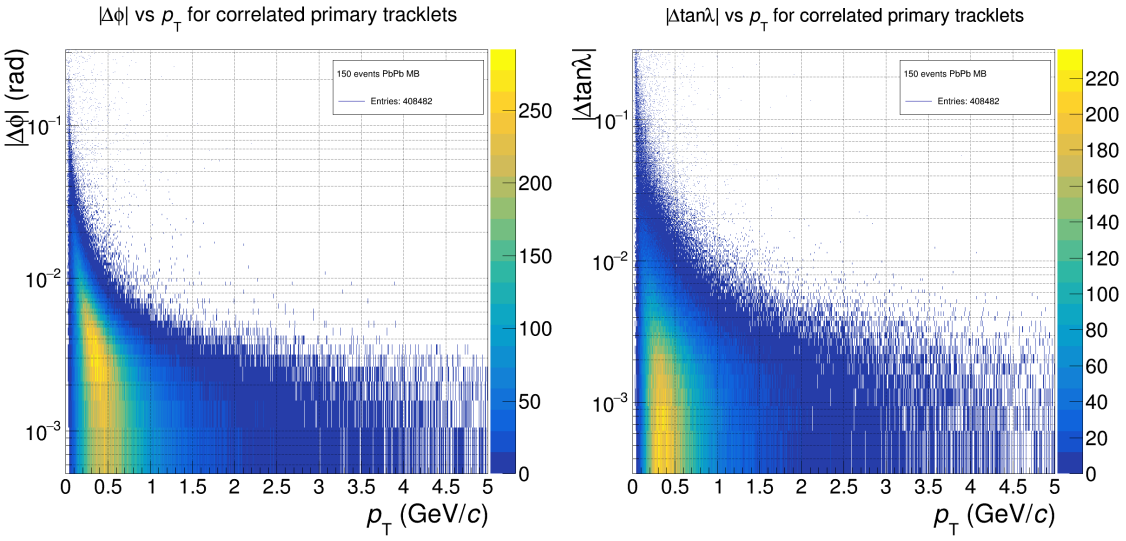
$$\theta_{ij} = \tan^{-1}(r_i/r_j) \quad \text{where } i, j \text{ are layer indices}$$

From inspecting those scatter plots it is already possible to see how a reasonable azimuthal selection can be $\Delta\phi = 0.005$ rad, although its impact is more tangible in section 5.2.

5.2 Tracklet selection optimisation

In order to further improve the efficiency in the signal selection a constraint on the tangent of the relative angle between two tracklets is imposed. Monte Carlo data have been used to analyse the distribution of those angles in order to determine the minimum value of the angle for which the efficiency is still satisfactory. The figures 5.4a and 5.4b

report the distributions, these are also used in cut selection to qualitatively visualise where the selection tested are going to affect in terms of p_{T} values exclusion.



(a) $|\Delta\phi|$ distribution vs p_{T} for correlated primary tracklets

(b) $|\Delta tan\lambda|$ distribution vs p_{T} for correlated primary tracklets

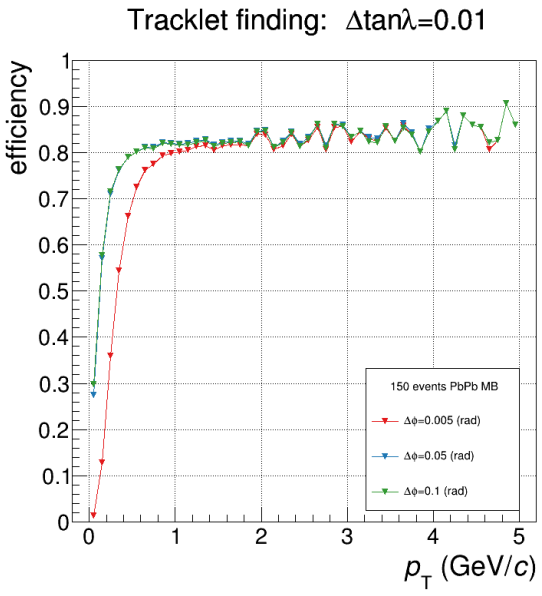
Figure 5.4: Distributions of parameters on which selections are applied vs transverse momentum for correlated primary tracklets. This is helpful to understand how the parameters related to the signal (tracks with a $\langle p_{\mathrm{T}} \rangle > 500$ MeV/c) are distributed, to tune future selections.

The relative angular deviation between correlated tracklets decreases when the transverse momentum of the particle trajectory they belong increases. This phenomenon is related to the multiple scattering with the Layer 1, that changes the direction of the particle. The probability to observe a larger deviation increases with the momentum of the particle decreasing. In figure 5.4a it must be kept into account the relative deviation is also due the presence of the magnetic field.

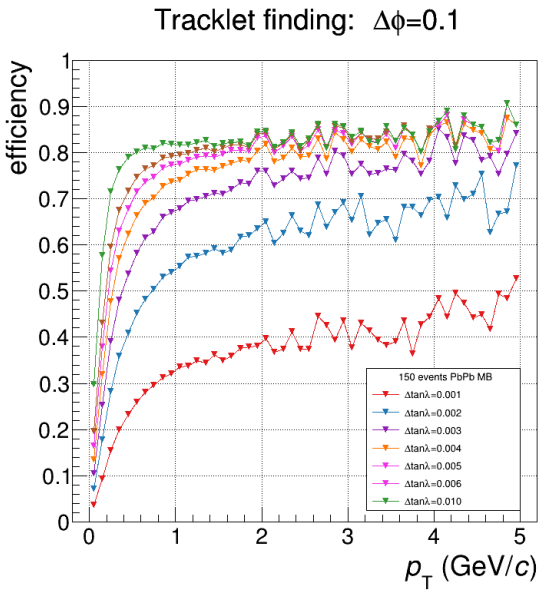
The selections are optimised by looking to the effect of the different set of cuts on the overall *tracklet selection* efficiency and on the signal purity. The first corresponds to the ratio between the number of correct associations of two tracklets and the total number of primary tracks with hits in the ITS. The latter is the ratio between the number of the correctly matched tracklets and the total number of found tracklets. The contribution of each cut to the performance in terms of efficiency has been evaluated by changing one cut at a time (while realising the other cuts) and looking to the obtained selection efficiency. This procedure is adopted not to sensibly affect the efficiency considering that one can have correlations between two different cuts. In figure 5.5a the efficiency as a function of the transverse momentum is reported for different value of the selection on $\Delta\phi$.

As expected, by narrowing the range of the selections, only higher transverse momentum particles are selected, namely straight particle trajectories.

Looking to the efficiency distribution reported in figure 5.5a and obtained by applying a cut in $\Delta \tan\lambda = 0.01$ and a cut on $\Delta\phi = 0.005$ rad it can be noticed that for transverse



(a) Tracklet selection efficiency obtained by varying the selection on $\Delta\phi$ and applying the selection $\Delta \tan\lambda = 0.01$. Over two orders of magnitude for the $\Delta\phi$ selection a drop on the efficiency is seen at low p_{T}.



(b) Tracklet selection efficiency obtained by varying the selection on $\Delta \tan\lambda$ and applying the selection $\Delta\phi = 0.1$. In this case the efficiency is very sensible to the selection over a single order of magnitude span.

Figure 5.5: Track selection efficiency: ratio between correctly matched pair of tracklets and primary charged tracks detectable in the ITS.

momentum of 500 MeV the efficiency is slightly higher than 75%. This is a good result and complies with the algorithm design pattern, aiming to reconstruct trajectories of particles populating that p_{T} region. Overall, a variation of two orders of magnitude in $\Delta\phi$ causes an efficiency reduction in the relevant region of less than 10%. From $\Delta\phi = 0.05$ to $\Delta\phi = 0.1$ rad performance is almost identical. Looking to figure 5.5 it is possible to see that the efficiency is more sensitive to the variation in the selection of $\Delta \tan\lambda$. In both figures points at high p_{T} are not stable. This is because at higher transverse momentum are produced less particles. Considering the distribution for pions shown in previous chapter 4.3, it decreases exponentially increasing the p_{T}.

The best pair of cuts must guarantee the highest efficiency in order to have a good performance also for events with a lower number of produced particles. At the same time the algorithm must provide the lowest value of fake matches (background rate) to select the purest set of candidates to be used in the vertex finding step. In the end, it is needed to find a good trade-off of these two conditions and possibly work on critical situations like events with few particle (e.g. *pp* events or ultra-peripheral collisions) or high-multiplicity collisions where it is harder to identify the signal over the dominant combinatorial background.

A second round of tests has been performed to measure how the signal over background ratio changes as a function of one variable when the other one has a fixed value. It is important to observe how the selection in $\Delta \tan\lambda$ is strictly correlated to the one in $\Delta\phi$.

The latter forces three connected clusters to be on the same plane. In order to ensure this condition, the selection on $\Delta\phi$ needs to be tight enough to assume that he projection of a considered tracklet on the (z, r) plane is as parallel as possible to the tracklet itself. Figure 5.6 reports a summary of the efficiencies in the tracklet selection. In this particular configuration the azimuthal selection has been set to a loose value $\Delta\phi = 0.1$ and the selection on the alignment in $\tan\lambda$ has been variated on a scale that goes from 0.001 to 0.01. Three plot are shown: the violet bar plot represents the efficiency in validating correct matches divided by the number of primaries detected by the ITS. There are also two graphs, the green one, on the top of the plot, represents the ratio between the number of validated "fake matches" (i.e. coming from combinatorial background) over the number of selected tracklets, whereas the red one on the bottom is the ratio between correct matches divided by the total number of selected tracklets. The two efficiencies are complementary and their sum is 100%. The red plot in figure 5.6 confirms that is not very effective to impose the constraint on the alignment on the (z, r) plane for the consecutive tracklets, with a loose selection on the azimuthal alignment. As expected, the efficiency is reduced without any significant reduction of the background. To be noticed that the tracklet selection is a stage that comes always after the tracklet finder. The tracklet finder that precedes the benchmarked tracklet selection in figure 5.6, is carried out with $\Delta\phi = 0.1$, as anticipated the selection is shared across the two phases. Generally for benchmarks presented hereafter every run of the first two stages has been configured with the same azimuthal selection in both steps. Concerning the violet bar plot, it is expected to obtain such an high efficiency for loose selections, since very small fraction of p_{T} region (in the lower values) is discarded. Aiming at suppressing the noise rate, it is therefore useful to adopt more strict selections, as it has been already concluded also for the tracklet finder. A reduction on all the efficiencies is expected to be observed because of the suppression of low p_{T} signal. Eventually it will be important to find a working point where the signal over background rate allows for good results in terms of efficiency.

Concerning the impact of the azimuthal cut, it is more evident how this selection has a great impact in the background reduction. Taking into account that the value of the cut is shared with trackleting phase and that every measurement is done by re-running the reconstruction, every reported value or bar in the graph is consistent with the same selection. Figure 5.7 shows the efficiencies of the tracklet selection as a function of the azimuthal angle cut. The alignment cut in the (z, r) plane is set to a large value to reduce the correlation effects and inspect all the efficiency windows.

The violet efficiencies are evaluated, as done before, by computing the ratio of the found correct matches over the possible discoverable primary charged particle trajectories. Red graph represents the ratio of correct matches over all the pair combinations whereas the green graph is $1 - \varepsilon_{signal}$. It is interesting to highlight how the violet efficiency starts to critically decrease at lower selection values (0.015 rad) compared with the two other curves. This leaves a relatively high efficiency range (violet) that represents best scenario possible, to find a selection value that does not reduce too much the efficiency. As expected, the selection on the azimuthal angle has a great impact on the resulting efficiency, since it imposes constraints on both the curvature of the approximated track and on the reduction of combinatorial noise produced by bad matches.

Finally, in figure 5.8, the selections are combined in order to maximize the purity of the sample provided to the vertex finder phase. The azimuthal angle cut is set to 0.005

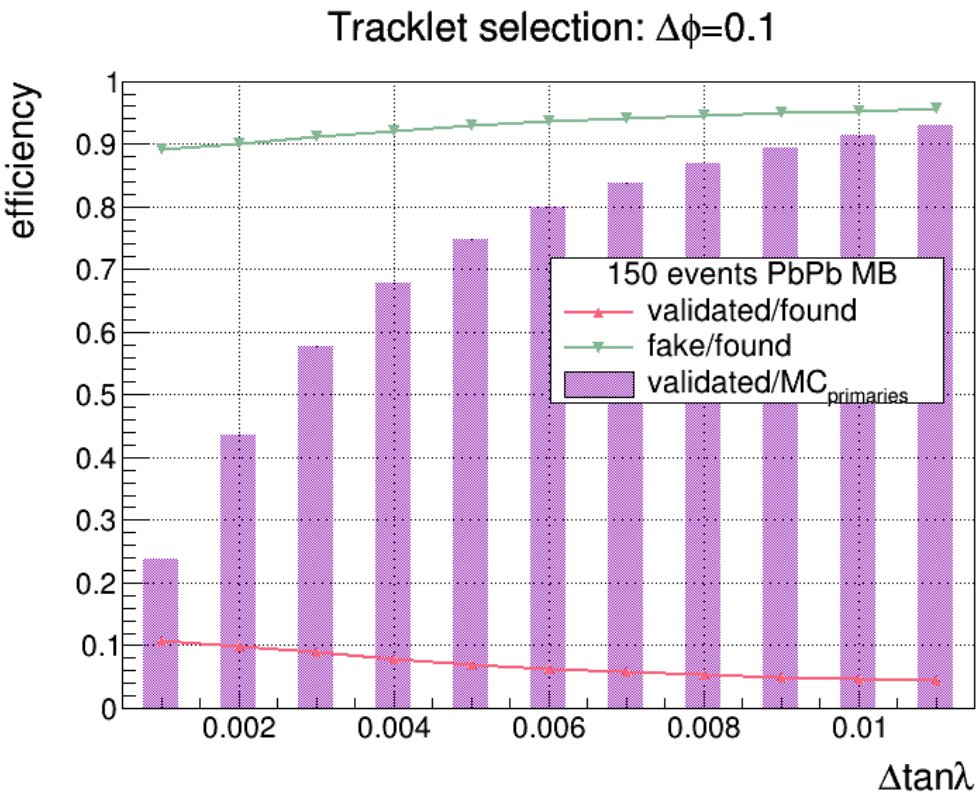


Figure 5.6: The violet bar plot shows the efficiency as a function of the cut on r, z plane. It rapidly drops by a factor 4 from $\Delta \tan \lambda$ that decreases of one order of magnitude. The red graph is the efficiency in selecting valid tracklets over all the found tracklets. The green one is the complementary information, hence the ratio between fake matches validated over all the validated tracklets. The correlation between two cuts is such that the selection on the $\Delta \tan \lambda$ parameter is neither correct nor effective without also imposing a strict selection on the azimuthal angle.

radians and the alignment in $\Delta \tan \lambda$ is varied within the same range used in figure 5.6. It is shown clearly how the high (violet) efficiency is reduced because of the tight cut and at the same time the red graph of the correct matches remains more stable around values greater than 90%. Eventually, the choices for the selections are tuned also considering the final performance of the whole vertex finding workflow. In particular, effects like the *fake pileup* will be discussed in section 5.3.

5.3 Vertex finding optimisation

The vertex finding optimisation consists in tuning the three parameters that interplay in the routine to maximise the efficiencies and the precision in the position estimation. The problem is not trivial, especially for those scenarios in which there is pileup of multiple events, namely the recording of many collisions in the same ROframe. The lack of a

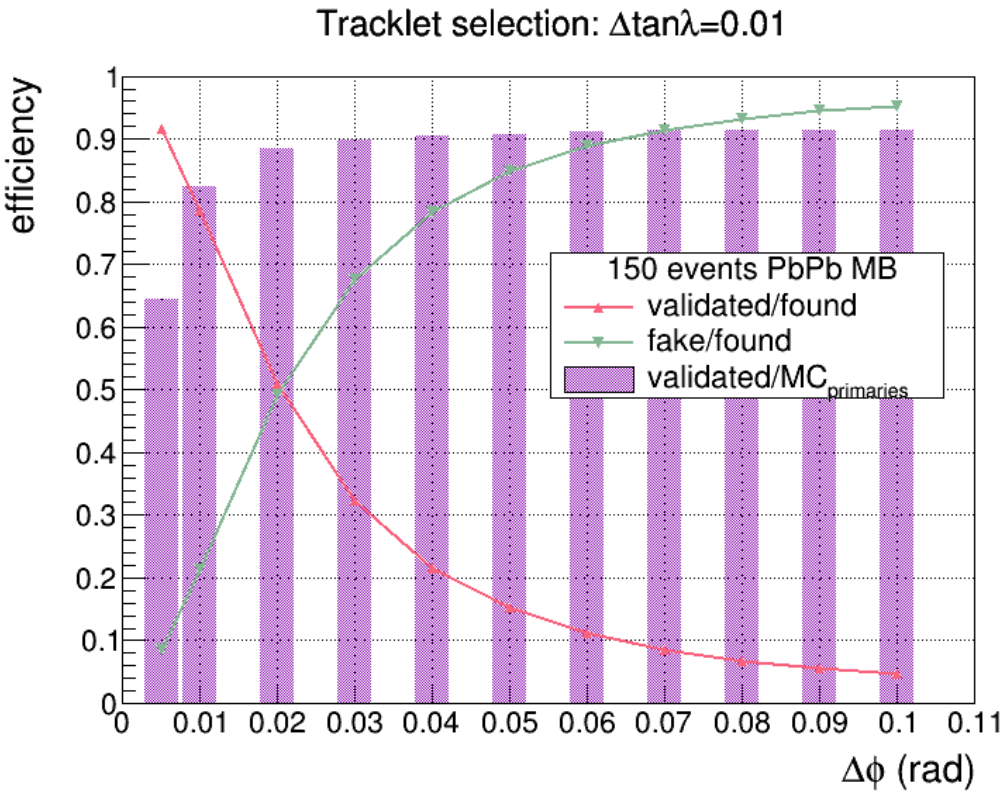


Figure 5.7: The violet bar plot shows the efficiency as a function of the cut on r, z plane. The red graph is the efficiency in selecting valid tracklets over all the found tracklets. The green one is the complementary information, hence the ratio between fake matches validated over all the validated tracklets. To isolate the only contribution of the azimuthal selection it is imposed a relatively large selection on the alignment. Noise fraction rapidly increases in the first 40 milliradians window and it stabilises more later on, around the 80% then it continues to slowly increase.

"triggered mode" for the data, a mode where the single events are logically separated and all of their data are processed in a single run of the algorithm, generally constitutes an additional degree of complexity in estimating the performance. In the following paragraph each step of the last phase of the algorithm will be profiled to measure the achieved performance and to improve the values. The first step of the optimization procedure consists in matching pairs of lines by measuring their distance of closest approach (DCA) and cutting out all those pairs with a distance greater than a given selection. As a preliminary study, the distribution of the DCAs calculated for each Monte Carlo validated lines pair has been investigated. The DCA distribution has been obtained taking into account all the matches between pairs of lines whose *centroids*, the middle points of the DCA vector between two lines, are located at least in the cylindrical volume covered by the beam pipe This is done to force pairs of lines to converge into the region where event collisions can happen. This selection is applied only to the (x, y) coordinates of centroids on the transverse plane, in

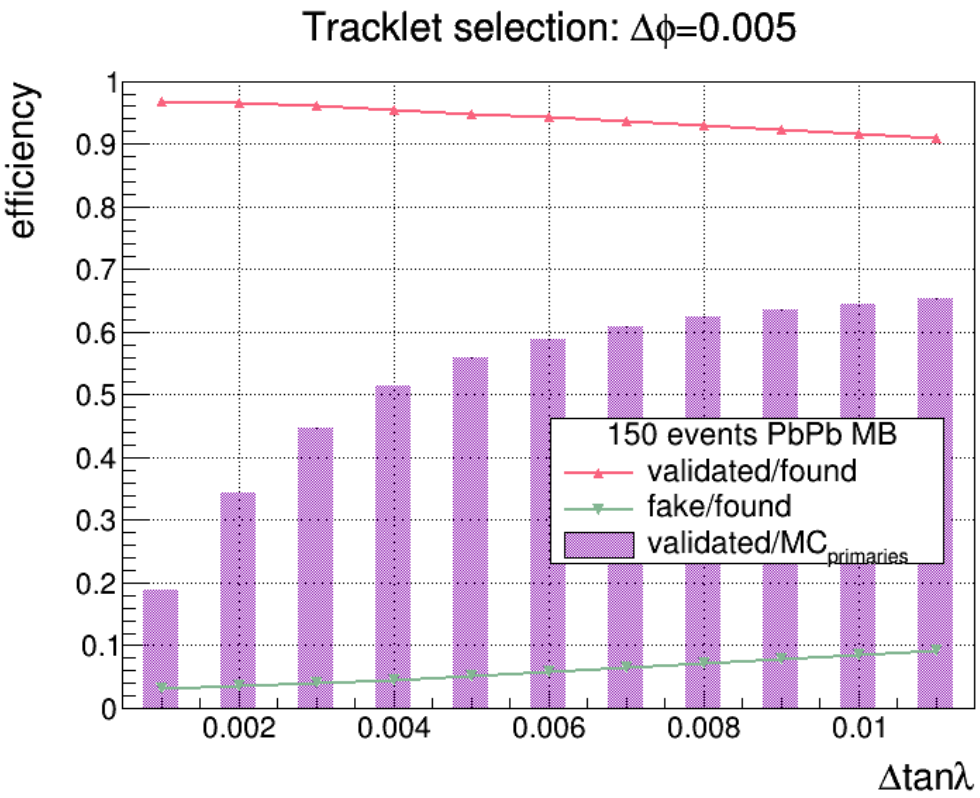


Figure 5.8: Efficiencies as functions of $\Delta \tan \lambda$ with $\Delta\phi = 0.005$. Potentially matchable tracklets acceptance (purple bars) is drastically reduced by the tight selections, in favour of a higher purity (always $> 90\%$) represented by the red graph. The noise rate is reduced down to less than 10%.

order not to impose any further bias on the pileup recognition. Such a selection is important because it is also used when the algorithm runs on real data not yet validated and represents a strong constraint. Plot in figure 5.9 reports the distribution of the module of the DCA vectors of each pair lines validated using the Monte Carlo truth, hence each is positive or at least null. Red line is the gaussian fit with a $\sigma = 220\ \mu m$, that is used to estimate the selection. Tails of the distribution can be fitted with a constant, they represent those low-momentum tracks that passed the selections but for which the linear approximation is not precise enough to reconstruct their trajectory. Their contribution is negligible since it is suppressed by a factor 10^3 with respect to the height of the gaussian. Taking into account the DCA distribution for Monte Carlo events, the selection to select good lines is done cutting at 2σ extracted form the Gaussian fit.

In figures 5.10 the distribution of the selected centroids is reported for each pair of Monte Carlo lines on the left. On the right there is a an enlargement of the same in the x and y range. Simulated events for this plots have been generated with nominal vertex position in (0,0,0). It is therefore expected to observe an accumulation of centroid position on the transverse plane around the origin. Only centroids whose radial distance from

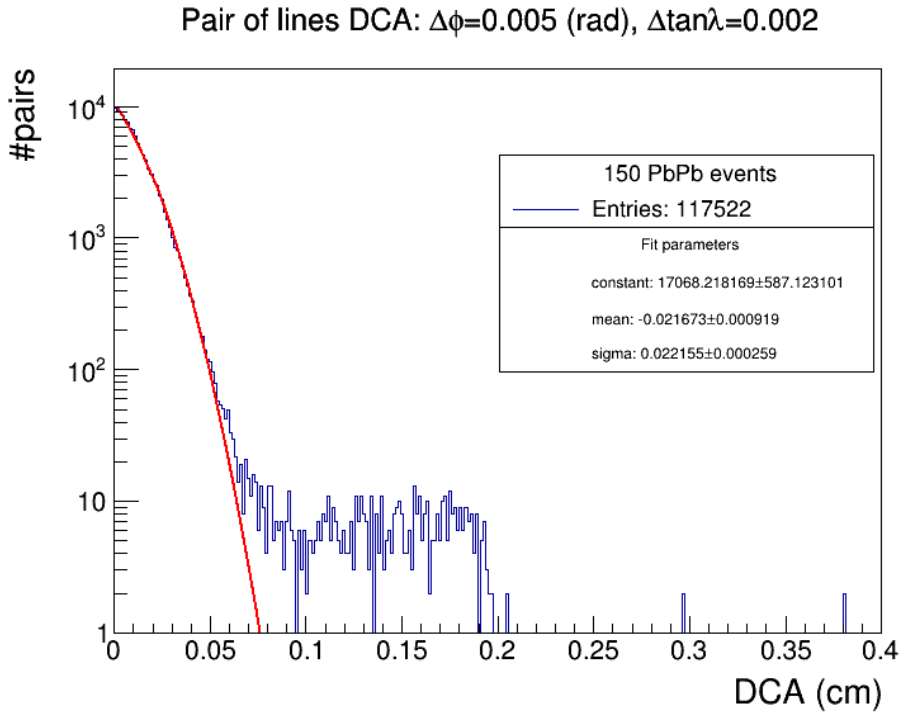


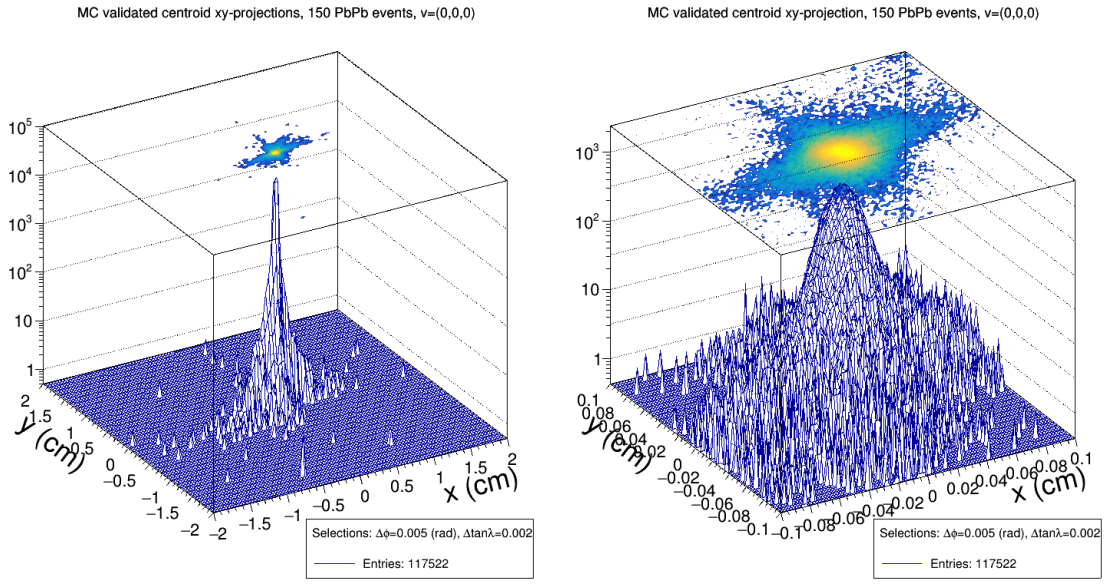
Figure 5.9: Distribution of the DCA of each pair of Monte Carlo validated lines. The gaussian fit to the distributions shows a sigma of $220\mu m$. This is a cooked cut that allows the reconstruction to select the most of correct tracklets.

the origin of the transverse plane is lower than the section of the beampipe[ii] have been selected. The distribution reported in figure 5.10a is constrained in a restricted area, one order of magnitude smaller than the beampipe. This is expected, due to the fact that the simulated vertices are in the origin of the coordinates on the transverse plane. Tails of the distribution present some spikes with a regular pattern. This effect is more evident in figure 5.11a which is produced without Monte Carlo checks. The core of the distribution, in yellow, with a radius of ≈ 0.02cm is more symmetric because all the primary particles are generated from the origin. With collisions generated in the whole interaction diamond the concentration in the distribution of centroid positions would be less narrow because of the displacement of the transverse position of the collision vertices.

Plots in figure 5.11 report the distribution of the centroids obtained from the same dataset used for previous considerations, without applying any validation based on Monte Carlo truth. Distribution on the right (5.11b) is an enlarged representation of the one on the left (5.11b). Considering the figure on the left, a regular pattern in the azimuthal distribution is observable in the tails of the distribution 5.11a. This typical star-shaped pattern is a consequence of two contributions: the turbo-geometry[iii] of the ITS and the

[ii] radius of the beampipe is ≈ 1.98 cm.

[iii] the turbo geometry has overlaps between adjacent staves to maximise azimuthal detector coverage.

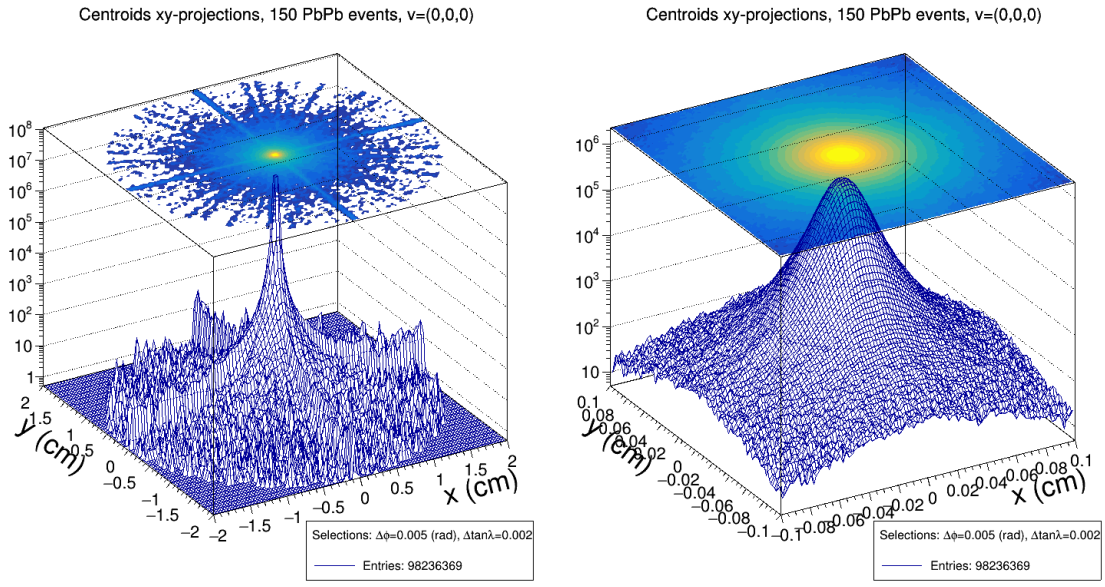


(a) Centroids are located within an area with radius smaller than 0.5cm.

(b) Enlarged view of the distribution on the left: centroids are accumulated around the origin within a circular area of 0.02cm.

Figure 5.10: On the left: Distribution of the transverse position of the centroids for every combination of Monte Carlo validated lines. Figure on the right is an enlargement of the plot on the left to better visualise the accumulation of centroids around the origin. Events are generated with vertex fixed in the origin of the transverse plane. The amplitude of the distribution is proportional to the amplitude of the distribution of DCAs, since centroids are medium point of DCA vectors.

procedure used to find clusters of lines in the vertex finder. An intrinsic implication of the azimuthal selections, centered exactly in the origin of the transverse plane, is that there is the possibility to find "duplicate" tracklets. Starting from the same cluster on Layer 1, it happens that within the same azimuthal selection window it is matched multiple times with different cluster on the adjacent layer. This phenomenon is enhanced in those angular regions where the staves overlap to each others. A particle crossing those regions is likely to be measured by both staves on the same layer, duplicating the signal, that is propagated as track "duplication" but not deterministically removable without introducing any bias. During the clusterisation of lines stage, the accumulation of centroids around specific azimuthal angle values reflects the collection of all possible medium points in DCA vectors found in the process. The duplicate lines are very close each other in the transverse plane, since the relative inclination in ϕ is smaller than the coverage in acceptance of the angular region of overlap of staves. Because of this close distance, their convergence on the transverse plane is not accentuated at all, such as in some cases they can be considered almost parallel. Therefore, when calculating the centroids of each pair, their radial position is more probable to distribute far from the distribution centre, where the centroids of other pairs of converging lines accumulate. The cross pattern corresponding to the transverse



(a) Centroids transverse positions are more spread across the beampipe cross-section compared to the ones from Monte Carlo validated lines. It is interesting to observe the regular patterns shaped as multiple overlapped crosses in the tails of the distribution, caused by matches of tracklets reconstructed multiple times in the angular intervals corresponding to overlaps of staves in single layers.

(b) The region where most centroids are accumulated does not present evident patterns as the tails of the distribution do. Also in this case the concentration of centroids is within a circle of 0.02cm radius. Due to the presence of more centroids coming from matches of background the area around the core is more populated.

Figure 5.11: On the left: Distribution of the transverse position of the centroids for every combination of lines not strictly validated. Figure on the right is an enlargement of the plot on the left to better visualise the accumulation of centroids around the origin.

plane axes is more populated because of the degeneration of this phenomenon. Considering the ITS layers, the first has 12 staves, second has 16 and third has 20, their greatest common divisor is 4. There are four angular directions, where all the three layers stave overlaps are aligned, corresponding to the axes of the transverse plane reference frame. Along these four directions, particles with high momentum (with almost straight trajectories) will encounter multiple times staves of the same layer for every layer, possibly duplicating the number of clusters left. Then the tracklet finder and tracklet selection routines will not be able to distinguish among them and will produce an increased number of tracklets at those directions. The vertex reconstruction algorithm is not able to perform any deterministic disambiguation in the selection a part form impose the constraint that the centroids must lay within the beampipe cross section, which is less probable for almost parallel tracklets. This phenomenon is also visible in Monte Carlo validated runs, due to the fact that the check on the Monte Carlo labels matching is done for each pair of clusters of interest. The removal of this effect in the Monte Carlo validation runs is possible: the most direct

approach is to find a unique match for each cluster of pixels, implementing some logic based on temporary flags, that keeps the information on whether cluster has been already matched or not. Then, in case of possible ambiguity due to duplications, is sufficient to choose the cluster with smaller distance from the origin. Since the linear approximation is used, a smaller segment will always be more precise in approximating a curve line. The reason why the aforementioned effect has not been removed, for the time being, is to keep the quantitative comparison with runs on real data more consistent, because when running on real data it is not possible to perform a deterministic selection on tracklets or lines. The "polarisation" (not uniform azimuthal distribution of clusters in the tail of the transverse distribution) of outer centroids is however an effect difficult to isolate and at the same time it has not a big impact on the overall performance. This is due to the fact that the "polarization" is suppressed of more than three orders of magnitude. Must also be said that the construction of the clusters of lines follows a procedure slightly different from the mere clusterisation of the centroids. In fact, it starts from a validated cluster and iteratively corrects the candidate vertex position by considering every candidate line, automatically discarding most of the fake centroids.

Concerning the unavoidable presence of duplicated tracklets or lines in the vertex estimation, there are some considerations that can be done:

- the main goal of the primary vertex finder is to get a estimation of the position of the vertex, with a precision of the order of the millimetre. The contribution to the resolution of duplicates is really negligible, as long as the required resolution stays within the order of hundreds of microns, still smaller than the requirements;

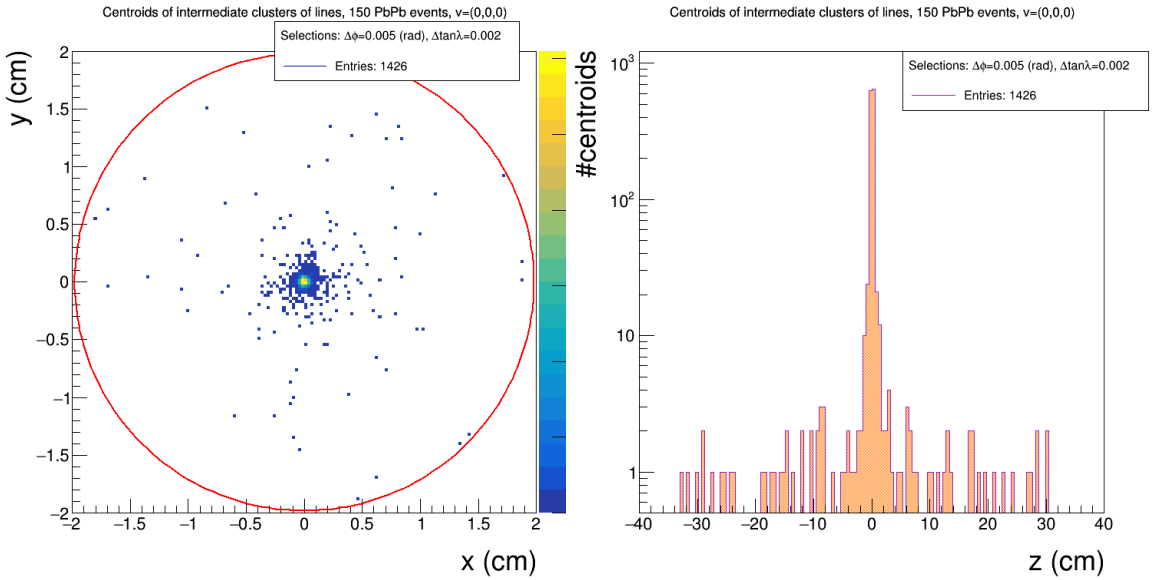
- in events where the estimation of the efficiency is quite difficult due to the fact that the filtered lines are very few, the probability of having mismatches related to the density of clusters is low. Duplicates have the side effect to enhance a correct signal, mainly because the overlap region is small enough to ensure that the particles generating those signals are intrinsically of high p_{T}. It has a positive contribution to enhance the signal in low multiplicity events, in case duplicates are produced in those precise directions. Otherwise it is a negligible effect in high multiplicity events;

- the clusters merging eliminates the largest fraction of outliers centroids, which have been enhanced due to the presence of many duplicates;

- concerning the bias introduced by the inclusion of duplicates, one has to consider that the final primary vertex position will be completely refitted by using the full track information.

Last inspection to the intermediate phases of the algorithm is done on the distributions of the coordinates of the clusters of lines, as they are created in the vertex finder. Figure 5.12a shows their position on the transverse plane and 5.12b their z coordinate. To be recalled that the cluster of lines creation starts from a pair of lines, calculates its centroid, then iteratively attaches each line with a DCA within the same selection used on line pairs.

After all the candidate clusters are created, they are merged if the relative centroid distance is small enough, according to the selections. The goal of this process is to remove all the outliers that do no count enough contributors to be accepted as standalone vertices and



(a) Distribution of transverse position belonging to the centroids of lines clusters. It resembles the distribution of the combinatorial centroids, a part from apart for the narrower core. Red circle is the beampipe cross section border.

(b) Distribution for the z coordinate belonging to the centroids of lines clusters. The distribution is obtained before the iteration done for merging the clusters. Some background is still present (enries far from 0 cm).

Figure 5.12: Partial distributions for centroids of clusters of lines before the merging. Data are from vertices generated in (0,0,0). to better inspect the z distribution.

stabilise the correct information stored by the "correct" clusters by merging their contributions. The centroids of the remaining clusters are promoted to primary vertices. Finally, the performance of the algorithm and its related selections are estimated by checking the quality the reconstructed vertices.

Taking into account the Monte Carlo data sample available at disposal, what has been done so far in terms of selection tuning is enough to determine in which phase the algorithm might fail in reaching the required performance.

5.3.1 Vertices purity measurement

The purity is the rate between the number of Monte Carlo validated contributors to a primary vertex divided by the total number of contributors. In pileup scenario there can be contribution from validated lines that belong to two different events. The association of a primary vertex to a certain event is done computing the most frequent validated ID among all the contributors, i.e. computing a majority using the Boyer-Moore voting process explained in section 4.4.3. The measurement purity of reconstructed vertices is a powerful tool to quantify the quality of filtered lines used to fit the vertex. The estimation is performed on the same dataset of 150 minimum bias Pb–Pb events spread across 545

ROframes, with vertices generated displaced both in z and on the transverse plane according aforementioned gaussian distributions with $\sigma_z = 5\ cm$ and $\sigma_x = \sigma_y = 100\ \mu m$. The selection adopted in the analysis presented in the following are those reported in Table 5.1: Results are presented for 151 reconstructed vertices: as it will be shown in section 5.3.2,

Selections: cluster-based vertexer	
$\Delta\phi$	0.006 (rad)
$\Delta \tan\lambda$	0.002
DCA	0.04 cm
$N_{contributors}$	16

Table 5.1: Selections reported in the table are those used to produce all the distributions related to the cluster-based vertexer presented in the following sections.

only 140 among the entries are related to unique IDs, therefore associated to single collisions. The others correspond to "duplicates" caused by the splitting of one event across two consecutive readout frames. One of the consequences of the continuous readout, because of the absence of a trigger, the impossibility to decide in advance which data correspond to the single event. The "rebuilding" of the information related to event parts is done after the reconstruction ITS-only.

In figure 5.13 there are reported the estimation of the purity for reconstructed vertices. The average value is between 97% and 100%. In this plot duplicates are anyways inserted

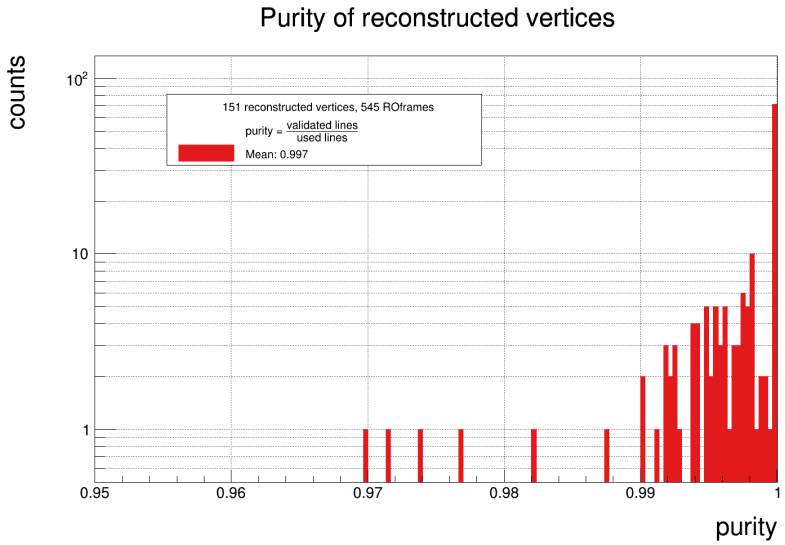


Figure 5.13: Purity of the reconstructed vertices (estimated as the ratio between the validated lines and the number of used lines) for Pb-Pb Monte Carlo events.

to highlight how, also when events are split across different frames, the purity remains greater than 95%. It is also worth to point out that even if the high purity obtained so far is an important achievement, one needs to also analyze the number of vertices not reconstructed because of the tight selections adopted in the analysis.

In figure 5.14 the trend of the purity of the vertices is reported ad a function of the number of contributors. It is clearly visible that when the number of contributors is higher than 10^3 the average purity decreases. The effect is due to the mixing of contributors from the overlap of data belonging to the same readout frame.

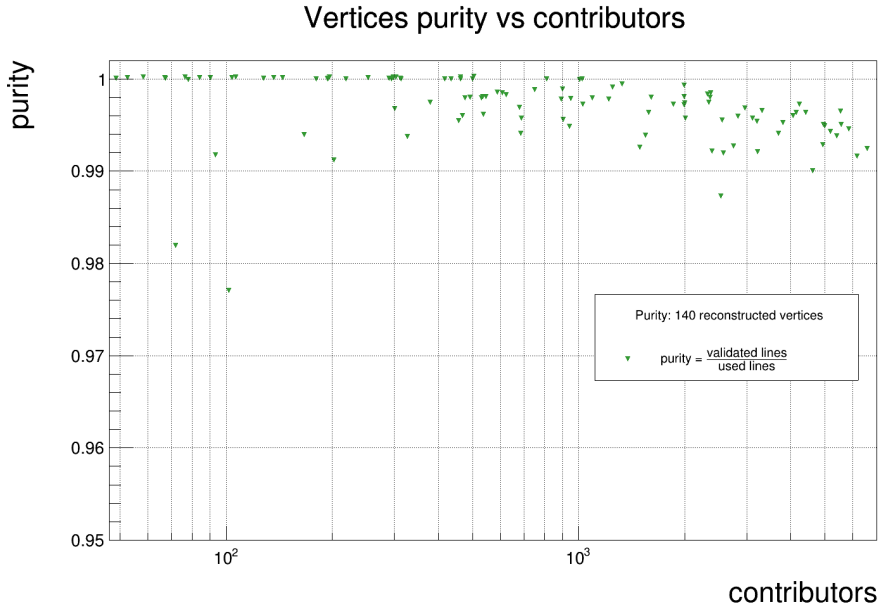


Figure 5.14: Purity versus the number of contributors: the trend shows a decrease of the purity when the number of contributors is greater then 10^3. This effect is due to the mixing of events in the same readout readout frame.

The estimation of the purity is pivotal to validate the lines selection process performed before the vertex finder. Selections in previous plots have been chosen to have no fake vertices found. The high purity is a consequence of the algorithm optimization in terms of reduction of false positives, namely *fake pileup* events.

5.3.2 Vertexing efficiency measurement

The first intuitive definition of efficiency is the number of reconstructed vertices compared to the number of "generated" ones. This definition is perfectly suitable when it is possible to isolate the process of the single event and to decide whether and to what degree of precision its vertex has been correctly reconstructed. In case of the continuous readout, the information on collisions is likely to be spread out over two or more readout frames. In particular, their distribution depends on the acquisition conditions, for instance the readout frequency, but also from the event duration and the tracks propagation time. The scenario is further complicated by the pileup. Within the same ITS readout frame there is a superimposition of partial data from different events. The primary vertex reconstruction algorithm aims at distinguishing vertices from different collisions by separating contributions (tracks) generated from different vertices and to fit their position from data that can also be shared not evenly across frames and to provide them as seed to the ITS tracking

algorithm. The products of the ITS tracker are track objects fitted with ITS-only only data and they are not assigned to a defined "event" until the whole reconstruction, including other detectors, process the *timeframe*[iv]. The event "assembling" is the final product of the whole reconstruction chain, which is enabled by the cross-matching of spatial and time information, provided by different detectors. The efficiency estimation, in the context of this work, needs to be extended to continuous readout situations. During the development of this work it was not possible to access data generated with a triggered mode. On the one hand, he possibility to regroup all the information related to the single event into a single frame is a good approach to evaluate the performance of vertex reconstruction as a function of parameters like the number of initial clusters or the number of *contributors*. On the other hand, continuous readout is the foreseen working condition, therefore tests need to be consistent with that data taking regime. Using the Boyer-Moore voting algorithm presented in section 4.4.3, it is still possible to find which kind of events have been reconstructed in every readout frame. In figure 5.15 the event IDs of the reconstructed vertices are reported. In some cases IDs are reconstructed more than once because of the

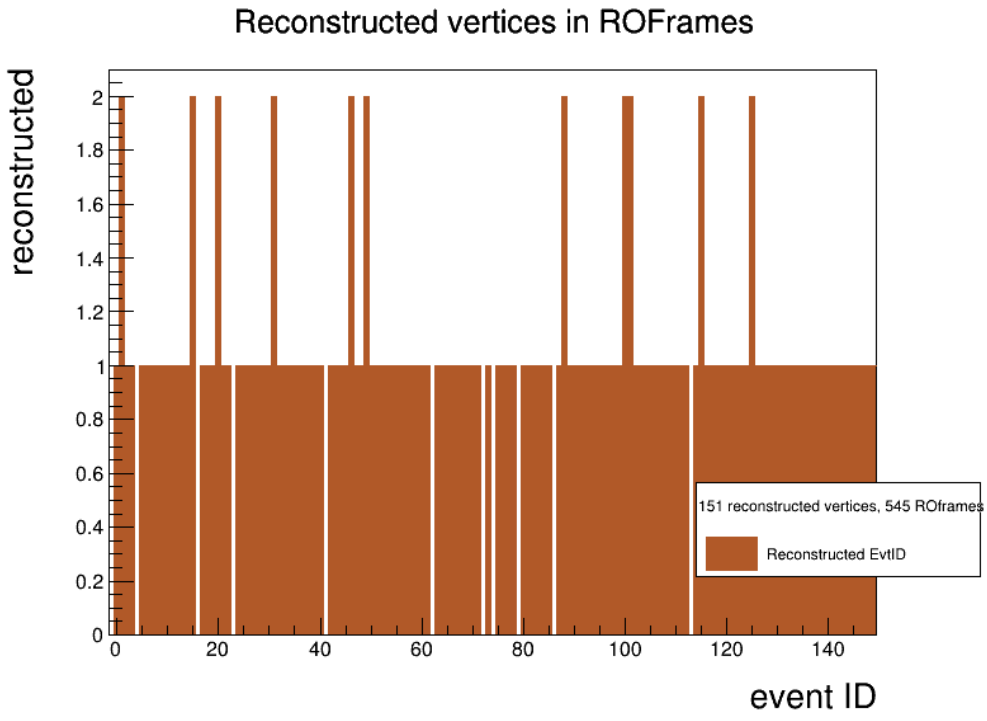


Figure 5.15: The histogram reports the IDs of reconstructed vertices spread across 545 readout frames. Indices are assigned using the Boyer-Moore voting routine. When more than one entry is present for the same ID, it means that the event is shared across at least two frames. White gaps represent missing IDs, hence vertices not reconstructed. No *fake* vertices (ID$= -1$) have been reconstructed in this dataset.

[iv] One *timeframe* lasts $25ms$ and is longer than one ITS readout frame which is around $\simeq 5ms$.

event data splitting over two readout frames. In these cases the reconstruction algorithm succeeded in reconstruct correct vertices with partial data. White holes, 10 in total, are those IDs that have not been successfully reconstructed. To reconstructed vertices with purity percentage $< 50\%$ it is assigned an ID$= -1$. In this specific selection setup it was not found any fake vertex, therefore it is possible to define a simplified version of an "absolute" efficiency. When there are no *false positives* in the reconstruction, the "absolute" efficiency to be able to reconstruct the single vertex is estimated as:

$$\varepsilon_{absolute} = \frac{N_{reconstructed}}{N_{simulated}} \simeq 93.3\% \tag{5.2}$$

By relaxing the $\Delta\phi$ selection, a higher number of missing vertices with a purity percentage $> 50\%$ are then detected, increasing the ratio of reconstructed vertices versus the Monte Carlo generated. On the other hand, as expected also the number of *fakes* increases. It has to be pointed out that finding the optimal setup is not the primary goal of this work, although it is intended to demonstrate that the results are promising enough, also in terms of resolution and purity, to be used in the reconstruction chain. A relevant consideration is that the tracking algorithm does not run on frames that have not a primary vertex associated to them, hence the probability of loosing the events that do not comply models used during the algorithm design is proportional to the inefficiency of the algorithm itself. For those reason the selection are optimized not only for the vertex finder but to maximize the performance of the tracker algorithm in managing fake vertices.

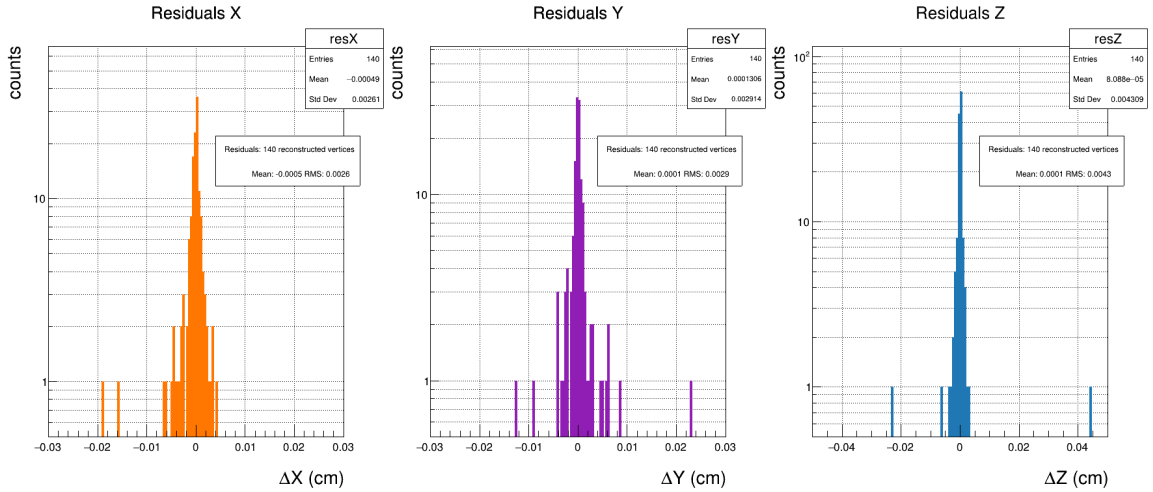
5.3.3 Vertexing resolution measurement

Another important indicator of the quality of the reconstructed vertices is the *resolution*. In this context it is a parameter represented by the RMS of the distribution of the *residuals*: the distance vectors of the reconstructed vertex position from the Monte Carlo one along the x, y and z directions. Comparisons are performed using the ID of found vertices obtained through the aforementioned voting routine. The residuals along three directions are reported in figures: 5.16a, 5.16b, 5.16c. In case of vertices reconstructed multiple times, the entries are taken only once. The RMS on the three coordinates can be used to estimate the resolution of the algorithm with this chosen set of selections.

	Δx(cm)	Δy(cm)	Δz(cm)
mean	-0.0005	0.0001	9×10^{-5}
RMS	0.0026	0.0029	0.0043

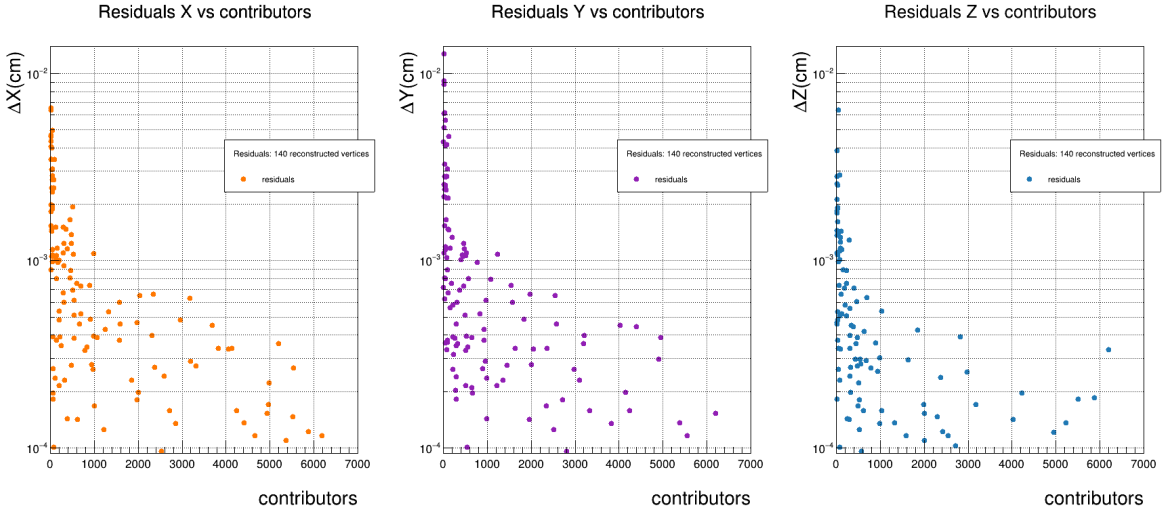
Table 5.2: Means and standard deviations of the distributions of the residuals between reconstructed and simulated positions of the vertices. Results are reported for the three coordinates.

In figures 5.17, as a complementary information, the distribution of the residuals versus the number of contributors of each vertex are also reported. The global trend shows how by increasing the number of contributors the residuals become smaller, especially from 0 to 10^3 the residuals goes down of a factor ten, whilst after that value the decrease is smoother.



(a) (b) (c)

Figure 5.16: Residuals distributions of reconstructed vertices.



(a) (b) (c)

Figure 5.17: From the left to the right: distribution of residuals in x, y and z direction as a function of the number of contributors.

Both signal and fake matches contributions are expected to increase with the growth of multiplicity with different proportions. As estimated by the purity studies, however, the purity remains quite high with the increasing in the number of contributors. In contrast with what has been observed with the purity, the RMS of residuals distributions reduce with the increasing of the number of contributors. The algorithm is therefore stable in keeping a pure signal in high multiplicity events, that helps in improving the final resolution.

5.4 Histogram-based vertex finder optimisation

The histogram based approach described in section 4.4.2 has a different sensitivity to the selections with respect to the cluster-based version. In this section the pros and cons of this method will be discussed and the same distributions shown for the cluster-based method will be presented for a direct comparison. Table 5.3 reports the value of the selections adopted for the tests and all the plots shown in the following are obtained by applying those selections. Some of the parameters used are the same as in cluster-based algorithm version, although their effect on the final results change because of the interplay with other parameters of the configuration of the histograms. The first difference between

Selections: cluster-based vertexer	
$\Delta\phi$	0.005 (rad)
$\Delta \tan\lambda$	0.002
DCA	0.01 cm
$N_{contributors}$	16

Table 5.3: Selections reported in the table are those used to produce all the distributions related to the histogram-based vertexer presented in the following sections

the two methods is that the cluster-based vertexer uses the DCA selection both in pairing lines to build temporary or partial clusters and in selecting lines to be added to existing clusters. In this second case the selection depends on the lines distance from the centroid of the cluster itself. The selection on the number of contributors $N_{contributors}$ reduces the spurious vertices made of few fake lines. However, while in the cluster-based this selection is used to filter out all the vertices found after the first one determined by applying the selection on $N_{contributors}$, the algorithm will return the first vertex found even if its number of contributors does not match the selection. This is not the case of current version of the histogram-based approach, where the $N_{contributors}$ selection is always applied. The rationale behind this choice is strictly related to the strategies in use and it will be described in detail in section 5.5.

The histogram-based algorithm stores the entries related to the contributors to the final vertex when the x,y and z coordinate are available. In table 5.4 the working setup used in this thesis is presented. Boundaries for histograms associated to x, y coordinates are the same, because of the azimuthal symmetry of the events and the circular section of the beampipe. Following the same logic used in the cluster-based vertexer, histograms in x and y are filled only with the coordinates of the centroids whose radial distance on the transverse plane is smaller than the beampipe radius (1.98 cm). In figure 5.18 the ID of reconstructed events with possible repetitions of events shared across readout frames are reported.

Results on the measurement of the vertex purity are shown in 5.19and it is possible to see how the set of selection reported before are particularly effective in terms of purity performance. Almost each of the reconstructed vertices has been using only Monte Carlo validated lines. This set of selections is probably too strict and this is confirmed by the value of the efficiency, that found 146 vertices which is less than that obtained with the cluster-based approach (151). The most critical difference is in the data which contributes

	x	y	z
Boundaries	[−1.98,1.98] cm	[−1.98,1.98] cm	[−40,40] cm
N_{bins}	401	401	4001
Bin_{size}	99 μm	99 μm	200 μm
$Bin_{interval}$	2	2	4

Table 5.4: Settings used for histograms on three coordinates. The number of bins per histogram is set by N_{bins}, the boundaries are equal for x and y and set to the beampipe size. The size of the bin is computed according to previous parameters. The $Bin_{interval}$ parameters are related to the number of bins to be counted on the left and on the right of a found maximum of the histogram to compute a weighted average for the position and smooth the binning effect for the final position of the vertex, as described in equation 5.3.

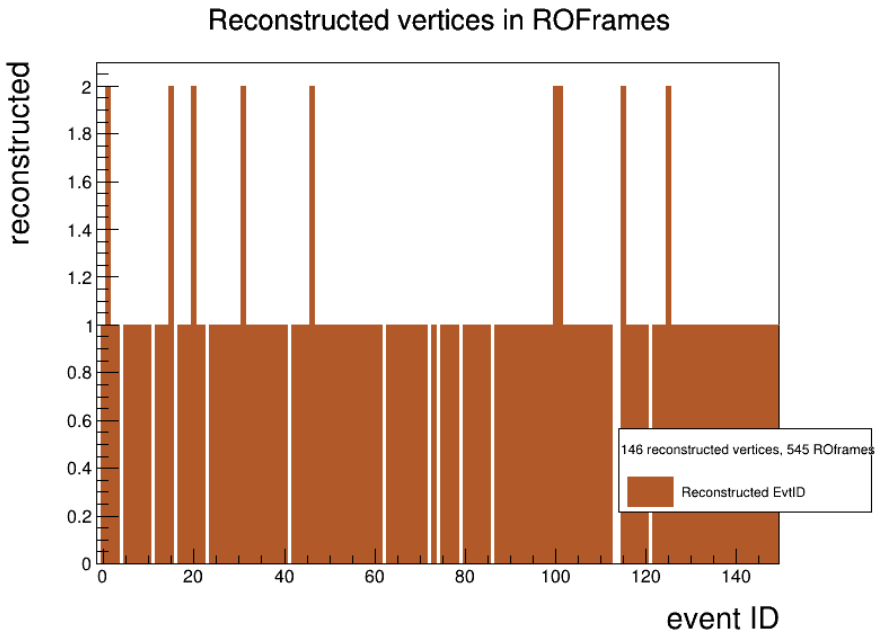


Figure 5.18: IDs of the reconstructed vertices in 545 readout frames. Chosen selections are presented in 5.3 and 5.4.

to the determination of the final vertex. The histogram-based approach only considers data included in a range $\mathcal{R}$ which is large

$$\mathcal{R} = 2 \times Bin_{interval} + 1, \tag{5.3}$$

and it is centered in the origin[v]. The histograms binning is usually set to an odd number.

[v] In production, the origin position is updated with online calibration information, frame position is updated accordingly.

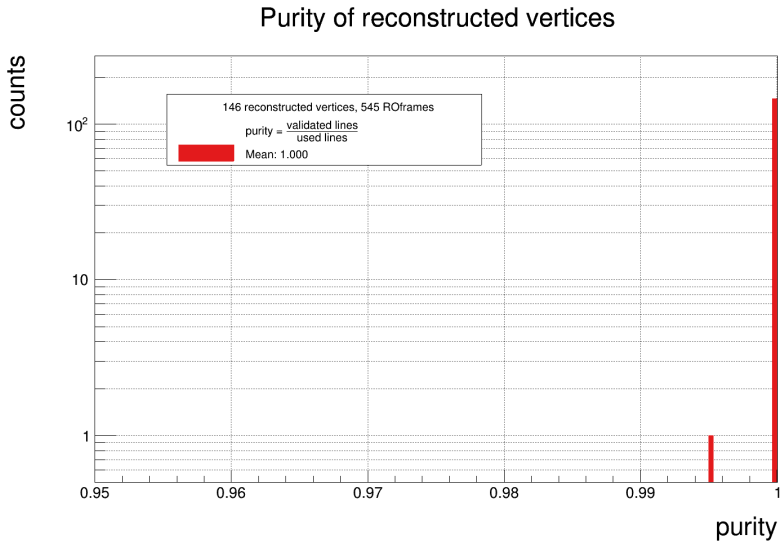


Figure 5.19: Purity of reconstructed vertices. The percentages are remarkably included in the $(96 - 100]\%$ interval, with an average of 99%.

With respect to what happens in the cluster-based vertexer, this reduces a lot the background out of a circular area of radius 250μm. This can also be seen in figure 5.12a where it is visible that the algorithm excludes contribution from a circular region. The plot in figure 5.20 represents the purity of reconstructed vertices versus the number of contributors. In this case as the number of contributors increases the purity does not decrease instead it remains constant.

In figure 5.21 the distribution of the residuals for x, y and z coordinate are reported for the reconstructed vertices with the histogram-based method. The values of the mean and the RMS extracted from the distributions are summarised in table 5.5: it is worthwhile to underline that the results obtained with the two methods are very similar in terms of resolution

	Δx(cm)	Δy(cm)	Δz(cm)
mean	0.0002	0.0003	0.0003
RMS	0.0029	0.0049	0.0043

Table 5.5: Values of the mean and the RMS extracted from the distributions of the residuals for x, y and z coordinate.

The scatter plots in figure 5.22 represent the residuals of the three cartesian vertex coordinates from their nominal Monte Carlo position as a function of the number of contributors to each vertex. In this case it is interesting to notice that the trend in the region of larger number of contributors has a less accentuated reduction of the residuals compared to what observed in previous approach by figure 5.17. In the specific case of the z coordinate residual it stabilises.

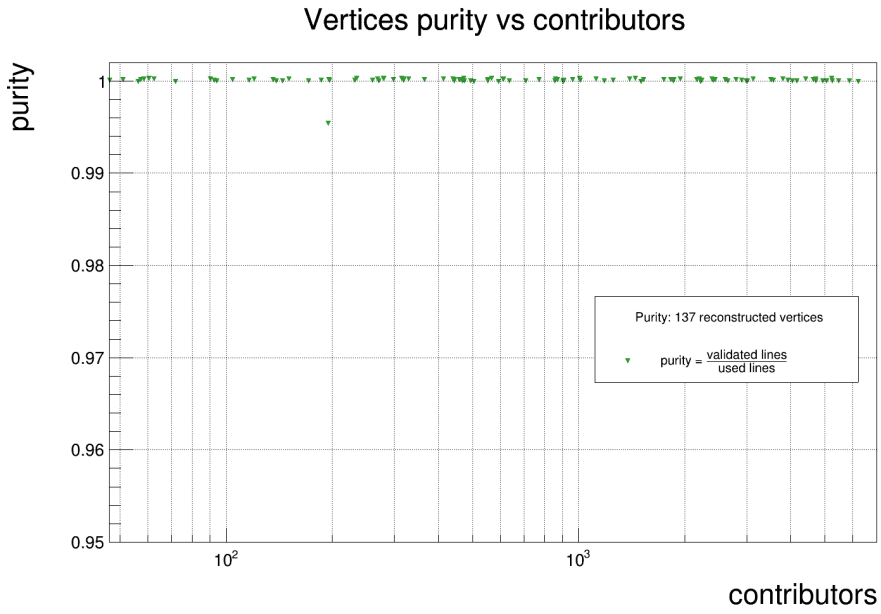
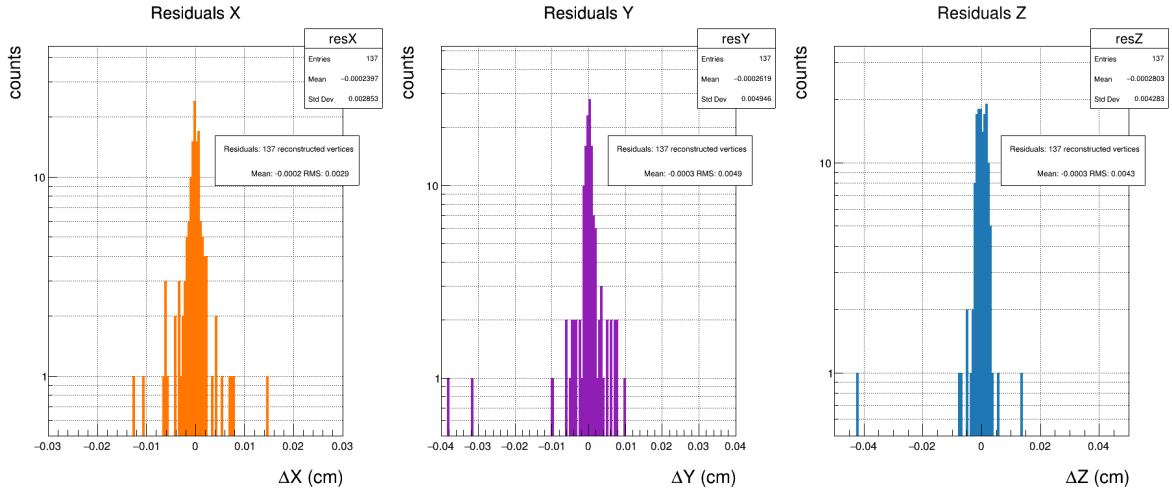


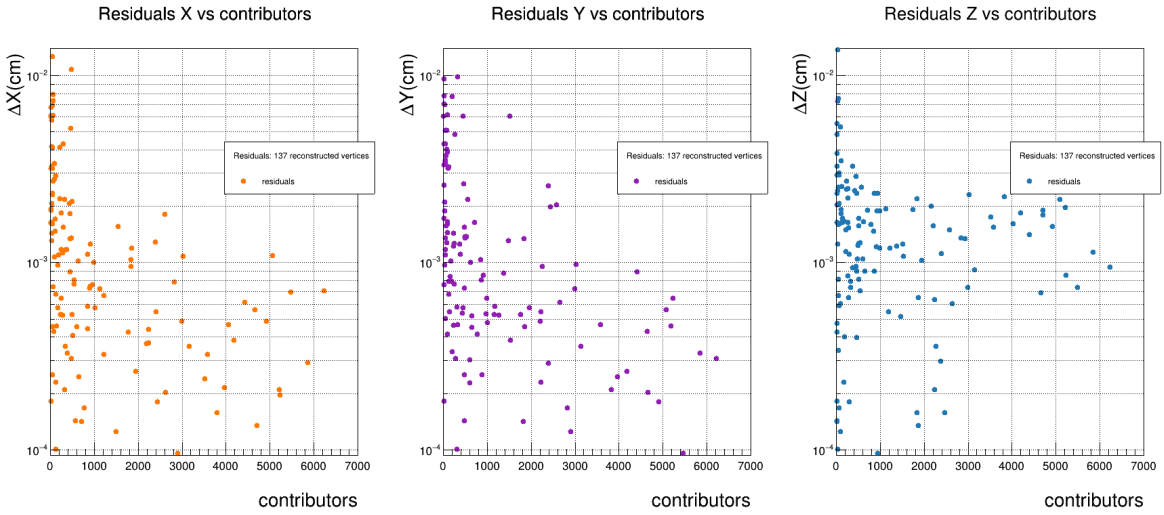
Figure 5.20: Purity versus the number of contributors: by increasing the number of contributors in this case the purity is not affected and remains constant. Selections are presented in table 5.3.



(a) (b) (c)

Figure 5.21: Residuals distributions of reconstructed vertices using the histogram-based approach. Selections are presented in table 5.3.

The difference is probably due to a binning effect which poses a constraint on the obtained resolution, whereas the cluster-based vertexing method does not have any limitation coming from space discretisation.



(a) (b) (c)

Figure 5.22: Residuals as a function of the number of contributors. Selections are presented in table 5.3.

5.5 Comparison of two vertexing algorithms

In the previous sections two approaches for the final step of the primary vertex reconstruction have been described by presenting the achieved performance. The *cluster-based* one, a pure analytical method that can be seen as an unsupervised clusterisation algorithm which applies selections when calculating distances from and between objects (namely clusters or straight lines). The histogram-based method is a "discretisations" of the destination space of the results, which at this stage of the reconstruction can still be a viable option. The usage of three different histograms, and to identify their the maxima to extrapolate coordinates, is an effective method which also offers more agile parallelisation strategies compared with the former implementation. In that scenario the resolution is determined by two factors: the size of the histogram bins and the number of bins considered to perform the final measure. The two approaches have some common features: their dominion is similar, both considers all centroids in the circular region delimited by the beampipe section. Both of them are able to cope with *pileup*, in different ways. The cluster-based method sets a threshold do decide whether two clusters are disjoint by using a selection on their distance projected on the z axis. It is usually set to $8mm$ and clusters are merged if closer than that distance, with the described technique. The histogram-based one reduce the pileup contribution by applying a selection on the number of contributors. In other words, once the first vertex candidates is found, the bin contents of the histograms used for that vertex are set to zero and another iteration is performed on the histogram, to find a local maximum that is greater than the threshold set on the number of contributors. If this request is satisfied the position of the vertex is calculated. This method permits the execution to find vertices which are separated by a distance $\mathcal{D}$:

$$\mathcal{D} = Bin_{size} \times Bin_{interval}, \tag{5.4}$$

If the values reported in table 5.3 are inserted in equation 5.4 , the minimum separation distance to distinguish vertices is found to be $\mathcal{D} \approx 200\mu m$. A point in favour of the reconstruction using clusters of lines is the the minimum number of contributors a vertex needs to be validated. In the cluster-based vertexer case, clusters are sorted by their size, meaning that the cluster counting the largest number of contributors will be the first in the sequence to be processed to extrapolate the vertex. There are, in this case, no cut on the minimum number of contributors on the first vertex: if it exists at least a cluster (therefore two converging straight lines pointing to a common centroid) counting down to two contributors that satisfies all the selection criteria, its centroid will be validated as a legitimate vertex. This sensitivity to events characterised by low multiplicities of tracks that have a p_{T} detectable by the vertex reconstruction allows to access very critical cases as ultra peripheral collisions. More importantly, it constitutes a strong starting point to the vertex reconstruction adaptation for *pp* collision system. The histogram-based approach does not have such a flexibility, although similar performance can be reached by properly setting the binning. It is also important to highlight that none of the two approaches is able to recognise two piled up events very close in the z coordinate, but separated on the transverse plane. In the design of the primary vertex algorithm it has always been assumed to be able to resolve the *pileup* only on the z coordinate. This can be considered a reasonable assumption, given the resolution provided by the linear approximations, the extremely collimated beam and the online alignment and calibration information that can affect the precision. Focusing on the differences between the two algorithms, apart from the performance in matter of efficiency and purity, there is one aspect that is very important The cluster-based vertex finder computes the vertex as the centroid of a cluster of lines. Its position is updated whenever the addition of a line or a merging happens. Within the same readout frame, in case of pileup, it is likely that the two (or more) output vertices will have different coordinates on the transverse plane. The structure of the histogram-based on the contrary, is divided in two main phases: the first one where the position of a *pseudo*-beam is calculated, and the second step when all the lines are matched with the found beam line, looking for possible multiple vertices. By construction this approach will provide aligned vertices with the same coordinate on the transverse plane. With the perspective not to discriminate on the transverse plane for multiple events, this can be considered acceptable. Moreover, for the real data, once the stability of the beam position is verified, one can perform this measurement less frequently instead of repeating it for each readout frame. This will allow for a faster processing of each readout frame and a precise information on the beam position which are used also for calibration and for the online reconstruction.

Concerning the absolute performance of the two methods, the comparison is meaningful if one compares the efficiency and not the resolution. This is proven by the fact that the resolution achieved with both methods are promising since the requirements are order of magnitude more generous (up to $\sim 1mm$) than the provided ($\approx 50\mu m$). As anticipated, this is a very sensitive topic, because of the strict dependency of the tracking reconstruction from the presence of a reconstructed vertex. After all the possible optimisations that still can take place in current implementations of the primary vertex finder, will be useful to evaluate how flexible will be the whole reconstruction chain to cope with the presence of "fake vertices". By using more generous selections in hte primary vertexer, it has been observed that the number of fake reconstructed vertices grow, as well as the efficiency in the detection of low multiplicity events. It can be set a working configuration that allows

for few fake vertices, knowing they will be manged and discarded by later steps, in favour of better efficiency. his tuning is not covered by this work, since it is meaningful to do such a tuning only when the full reconstruction will be run online during the commissioning phase of the O^2. Deeper analyses and investigations need to be performed as soon as a larger than a 150 events dataset will be available. Eventually, vertexer will be tested also on *pp* collision system, usually characterised by lower average multiplicity (reduced of a factor 10) and higher *pilelup*, the routine will extended to maximise the performance also on that scenario. This will allow for an extended and improved Pb–Pb reconstruction by providing mechanisms to cope with ultra peripheral events also in pileup scenarios.

Chapter 6

Implementation and integration of a sequential vertex finder in O^2

As presented in 2.2, almost any minimal piece of a workflow developed for the O^2 is integrated as Data Processing Layer (DPL) *device* running as a separated UNIX process. Devices are distributed either on a single machine or on a cluster of machines and communicate to each other with shared memory or message passing, depending on the deployment strategy. For development and debug purposes, the O^2 libraries are also available in the ROOT version compiled with the framework, allowing for O^2-specific classes to be accessible via ROOT macros. This approach is really effective, as it provides a high level of granularity and a step-by-step testing method of the workflow, keeping executions minimally affected from other pieces of software not really involved in the development. During this work three versions of the vertexing algorithm have been developed and deployed: the first one is a linear and sequential implementation purely running on CPU, the other two are mainly GPU code, steered by CPU callbacks.

6.1 Interface of the vertexer program

One important aspect concerning the usage of *heterogeneous*[i] resources in the O^2 is related to the idea of using a framework capable to automatically perform decisions, according to configurable policies, on what architecture to use when it is available on the target system in use. Considering that the core idea of the "Online-Offline" framework is to have the same identical software stack running both on online clusters like the EPN (see section 2.1.1) cluster and on a laptop, it is necessary to have a common software that can be run on the most common hardware (CPU). At the same time it is a great challenge to have a software capable to exploit all different resources that can be available on the

[i] In this context with "heterogeneous computing" it is referred the execution of the programs on systems that use more than one kind of processor or cores.

target node, for instance the GPUs, ensuring the cross-platform consistency of the results. This logic is rendered using **auto-detection** and **polymorphism** and the primary vertex reconstruction complies with these features as well. The former is *statically* accomplished at compile time, where the build tool detects the architectures and Application Programming Interfaces (APIs) available and produces binary code for all the usable platforms. The latter provides a unified interface to manage the execution, with general methods to call the different steps of the reconstruction, in a transparent way with respect to the underlying technology in use. This is obtained by *dynamically* loading specific libraries and steering classes that inherit from a common class. While there are no evident drawbacks in this kind of approach, there are several advantages. It s very useful very useful from a code readability and maintainability perspective, it also avoids a lot of code duplication for what concerns the usage of the different vertex reconstruction methods. The strategy adopted to encapsulate the different implementations is sketched in figure 6.1. The CPU version is always compiled, while the GPU versions are compiled whenever the corresponding associated platform is found, creating separate libraries that override specific methods used by the single interface. The *VertexerInterface* represents the final API used both by macros and DPL workflow. It stores a pointer to the object instance that matches the selected version. Then, thanks to features of C++ such as polymorphism and the member functions that can be overridden, it exposes a uniform interface that specialises in its internals the correct callbacks to the pointed object, thus the code that steers the architecture. In green and red the boxes are represented the two different GPU code that

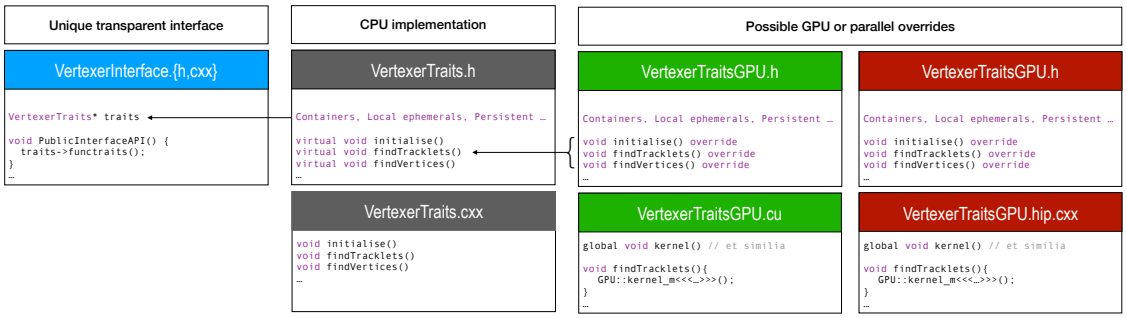


Figure 6.1: Schema of the interface and inheritance chain for primary vertex reconstruction. In light blue the interface finally exposed. The grey class describe virtual vertexer *traits* and also have its CPU implementation compiled. The red and green classes inherit from the grey traits and separately override all the methods that involve different routines such as the GPU executions. Two separate libraries are produced by the compilation on a machine that owns an Nvidia and an AMD GPU. The CPU version is always produced as a default case.

produce separate libraries upon compilation. Depending on the directives coming from the architecture auto-detection, separate libraries are built with their separate headers. Both of them inherit from the CPU library, represented in grey, and override its virtual methods. Thus, the polymorphism will allow to pass a pointer to either of the three classes to the interface and it will act directly on the chosen instance. They will be described in details in chapter 7.

6.1.1 Standalone debugging utility and Monte Carlo truth access

The goal of this section is to highlight the utilities accessible by the workflow. The second part of the chapter will focus on CPU version and its performance estimation, in terms of elapsed time and memory footprint. Any version of the vertexer is able to save on disk at anytime quantities that are useful inspect and evaluate the internal state of the process. A separate component is therefore introduced for debug-purposes only that allows the streaming of parameters to be post-processed into ROOT trees. For compatibility reasons related to the different build systems used by GPU frameworks, such an interface has to be completely separable from the vertexing utilities. This is also in agreement with the idea of running essential code on production setups to ensure, for the sake of performance, that there are not useless operations and checks done at runtime. A toggle has been provided to enable this so-called "debug mode", which activates dedicated regions of code, delimited by precompiler directives, at compile time. The *Vertexer* interface has then a granular system to trigger a snapshot only of the required quantities. The methods constituting this system are always available, also in production code, but they are non operational in that case, to avoid the duplication of debug and production code. The dump is provided in a batch fashion, meaning there cannot be any interaction between the runtime state of execution and the output data. Under the same conditions access to Monte Carlo information is given during the execution of the program. This is indeed something that must be completely excluded from production code, because of the absence of such metadata during the reconstruction operations. In figure 6.2 it is represented a schematic overview of the logical components of the workflow of the vertexer. Monte Carlo truth are provided externally as the input data are. The debugger is a standalone utility completely separated that adds further information. These features have been added as helpful tools during the development, debugging and benchmarking of the program. For instance, to tag the validated tracklets, among those produced by combinatorial matching, allowed for producing the p_{T} distributions of found tracklets, as presented in chapters 4 and 5. To

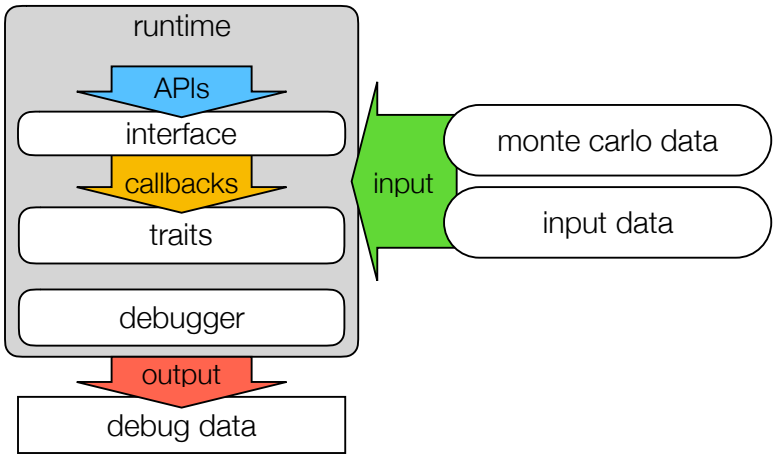


Figure 6.2: Workflow of the vertexer represented as the interaction of its components.

benchmark at its best the vertexer, and not to affect the actual code by including lines for the speed benchmarking, it has been introduced a wrapping template function that takes

as an argument the methods of the *VertexerTraits* and computes the elapsed time by using standard *chrono* library for time measurement.

6.2 Implementation of a host-bound vertex finder

The first version of the primary vertex finder is a serial working implementation, written in C++. The standard language adopted in this work is C++11 with extensive usage of the ANSI C++ Library [54]. Coding guidelines of the O^2 project allow for the adoption of even more modern standards, up to C++17. The vertexer implementation is structured around a single class used as main interface, called *Vertexer*. Such an interface is usable both integrated inside a workflow definition (i.e. *ITS reconstruction workflow*) and from ROOT macros, provided that the necessary headers are included. This possibility allows for strong consistency across the two possible usages[ii] and is effective in providing other developers examples on its usage. The interface has been designed to expose a minimal toolkit to steer the reconstruction. The core idea is that the methods exposed by the toolkit should hardly change in their signature or return values. As a matter of fact, they are all marked as *void* and only change internal status of the class. Results or temporary data can be extracted by specific methods. The rationale behind this choice is to have an entry point as rigid as possible, the Vertexer object, very hard to be changed, and to factorise all the flexibility within the implementation of vertexer internals, called *VertexerTraits*. Traits are a stateful class that stores the data to be processed and the temporary artifacts used to fit the position of the vertices. These traits not only can be switched, thanks to polymorphism, to different architecture implementations, but also are pieces of code that can safely be changed or extended without affecting the main interface. In this way corrections, improvements and refactoring are possibly performed on the code without burdening the developer to also have to update any other piece of software that utilises the primary vertexer interface. In a collaborative development environment this ensures very clean actions and provides an effective separation of competence regions, allowing for a seamless integration of different parts, developed by different people.

Another important aspect was the readability and maintainability of the code. In the best case scenario, the vertexer code base is foreseen to be used for all the duration of the Run 3 data-taking and later. Therefore, it is very likely that other developers are possibly changing, improving and fixing parts of it in later phases. Also, the structure of the code is designed to resemble the version that runs on the GPU, in order to facilitate the comparison between two implementations of the same routine and possibly update both of them in a straightforward fashion, regardless of the fact computing languages differ a bit. To be more specific, not only the GPU *kernels*, introduced in 3.2, find their corresponding counterparts in C++ *pseudo-kernel* to have a bidirectional mapping between the two approaches, but also the logic of the functions is kept as close as possible on different architectures. This helps a lot in testing the code also on nodes where GPUs are not available and allow for a a very fine granularity in comparing lines of code. This will be shown more in details in chapter 7.

[ii] Although in production only the DPL workflow will be used, it is important to have the possibility to isolate the vertexing part only, to possibly run unit and performance tests

6.2.1 Memory footprint and profiling

Specific attention was paid in defining data structures that could accommodate the minimum information needed by different phases of the reconstruction. The core idea was to have, as input of each phase, the thinnest structures storing only data needed in that execution. With many DPL devices running on the same host, it becomes really crucial to have efficient resource management. Cluster data structures passed to the vertexer are either read from file or from a DPL-provided input, are converted into a lighter format that stores only the cartesian coordinates, the cylindrical complementary computed ones, an index to refer the original source and an index to order it into an index table. The total size $\mathcal{S}_{cluster}$ of a cluster is

$$\mathcal{S}_{cluster} = 5 \times sizeof(float) + 2 \times sizeof(int) = 28 \text{ Bytes.} \tag{6.1}$$

For the precision achieved and for the type of mathematical operation done with clusters, there is no need to use double precision types, thus saving resources. The layout of tracklets objects foresees two indices, used to store the position occupied on the vectors by the clusters that constitute them and two floating point values to store their the tangent of the λ angle and the ϕ angle. Notably, those are the two variables on which are applied selections. They are computed for each pair of clusters and, due to the large number of tracklets that are discarded, it has been opted to compute and store full tracklet information only for those which are then selected for successive step. Size $\mathcal{S}_{tracklet}$ is therefore:

$$\mathcal{S}_{tracklet} = 2 \times sizeof(int) + 2 \times sizeof(float) = 16 \text{ Bytes.} \tag{6.2}$$

After the selection process, from the selected tracklets larger objects, the lines, are computed. They store full information related to the tridimensional line and its structure contains more elements:

- a tridimensional point, the origin of the line (3 *floats*)
- three direction cosines (3 *floats*)
- the parameters of a symmetric 3×3 weight matrix (6 *floats*)
- a flag to store an emptiness value (1 unsigned *char*)

for a total of $\mathcal{S}_{line} = 52$ Bytes. The symmetric matrix is at the moment of writing this work not enabled, in the sense that the errors associated to the coordinates of the clusters are not used, hence not propagated, to the workflow, because their contribution is neglected, compared to the accepted resolution on the vertex. This is temporary, in future tunings of the working conditions they might be included and the code will be already in place.

The last data structure is represented by the cluster of lines. By their nature they can grow indefinitely limited by the total number of lines, depending on the number *contributors*, hence their size grows proportionally. All the internal mathematical operations concerning the addition of a line to the cluster are written in form of matrix products and include the error propagation. In this way, in case the error propagation will be included in the vertex evaluation in the future, there will be no deterioration of the time performance. The class that implements a cluster of lines contains a total of 128 Bytes as a base size for

any cluster. In the case of the vertex finder based on clusters of lines the indices of all the lines contributing to each cluster must also be taken into account. The actual process of merging two clusters of lines requires to add one by one all the contributing lines of one to the other. When two clusters are merged, the size of the first one increases proportionally to the number of contributors of the second one that are added, whilst the second is deleted at the end of the process. The final size of the first will be slightly smaller than the sum of the two, because of the possible rejection of some contributors and of the presence of one single overhead. Summarising, the size of a cluster of lines $\mathcal{S}_{cluster\ of\ lines}$:

$$\mathcal{S}_{cluster\ of\ lines} = 128\ + [N_{contributors} \times sizeof(int)]\ \text{Bytes} \tag{6.3}$$

It has been interesting in understanding how effectively the memory usage can grow in solving the combinatorial problem on a serial implementation of the algorithm on CPU. Data processed for single readout frame are not uniform in size: because of the data acquisition frequency and the *triggerless* data taking, the size of the objects stored in the single frame varies a lot. Because of the additional overhead that would be caused by the memory (de-)allocation operations, it is not a viable option, for the sake of performance, to shrink and expand the internal data containers (*std::vectors*) used. C++ *std::vector* objects provide methods to clear their internal buffers, however an actual memory reallocation is actually done only when it is considered necessary by the runtime execution. Overall, also comparing with the complexity estimations done in section 4.3, the majority of the memory is used by the tracklet finding, entirely dominated by combinatorial, then the rest of the algorithm involves larger structures but in lower number. To understand the general memory footprint of a run of the vertex finder algorithm, a plot of the memory used as a function of time during the processing of 200 Pb–Pb events is shown in figure 6.3. When an iteration requires more memory than the previous one, the allocated memory increases, and this may happen from time to time until it reaches a *plateau* corresponding to the maximum value required by the computation. Sometimes also some re-allocation might take place, causing fluctuations on the plateau, but they are not predictable, because are automatically managed by the operating system.

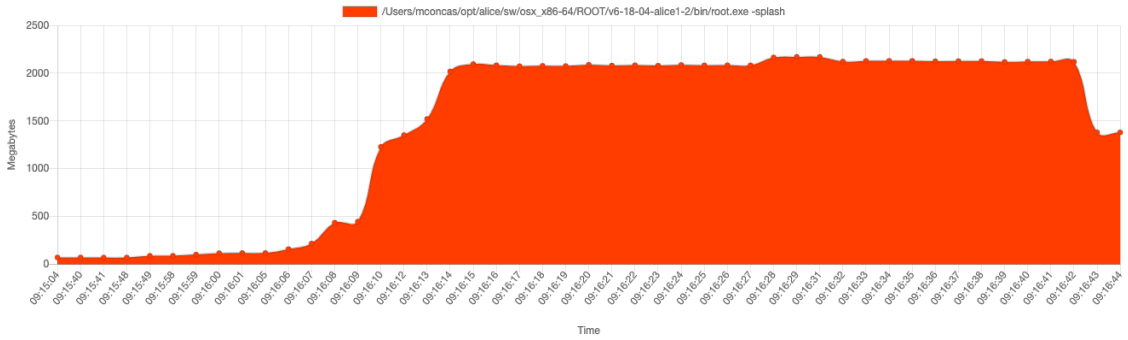


Figure 6.3: Plot of the memory occupancy during the processing of 700 readout frames, for a total of 200 inelastic Pb–Pb simulated events with no restriction on the impact parameter. Vertex reconstruction is invoked and steered using a ROOT macro that reads the data from a file.

It is worth to mention how the current flexibility in the CPU memory management is

a feature provided by the architecture. It will be illustrated more in detail in chapter 7 how the GPU implementation of the corresponding parallel version of the algorithm takes advantage from a static memory setup.

6.2.2 Time profiling

In this section, a benchmarking of the time performance of different implementations is presented. Specifically, the measurement of the elapsed time for every potentially computing-intensive execution is very relevant to estimate the amount of computing resources will be needed for an online reconstruction algorithm. As already presented in section 2.1.1, there will be an ephemeral disk space used as a buffer to retain for relatively short time the incoming data. Their processing will be performed by many instances (DPL devices) of reconstruction routines. There will be more than one execution pipeline, allowing for an horizontal scaling of the computing capability of the O^2 instance. Readout frames, the minimum load of processable memory, have a variable size depending on whether or not they include data from collisions. The elapsed time in reconstruction is not constant, because, depending also on the acquisition frequency and the duration of the frame, a number of them can be almost empty. The RMS of the distributions of the elapsed times for every vertexing phase is therefore included, since frames can be more or less populated.

As a reference for the benchmarks that will presented in the following sections, including the ones performed on GPU version, a summary of the hardware used to develop and test the vertexer software is reported in table 6.1. Benchmarks are performed, for simplicity, on macro-driven vertexer runs. The overhead caused by running in a DPL device is negligible, thus not included.

Development workstation specifications	
CPU	Intel Xeon(R) W-2133 CPU 12× cores @3.60 GHz
Memory	RAM 8 GiB DIMM DDR4 2× @$2.67 GHz
GPU 1	NVIDIA TITAN Xp(R), 12 GB GDDR5X, 3840 × CUDA(R) cores @1.6 GHz
GPU 2	AMD Radeon PRO WX 9100(R), 16GB, 4096 × Stream processors @1.5 GHz

Table 6.1: Summary of the technical specifications of the node used to benchmark the vertex finder. It is equipped with two GPUs from different vendors, very similar in specifications.

Data are read from ROOT files containing trees with the information provided by a *message* of an external source. Time spent in loading data from file has been excluded from the measurement, because data are expected to be provided by DPL itself in form of messages. To count the elapsed time it has been implemented a lightweight wrapper written in C++ and based on the *std::chrono* library, which provides a time resolution down to the nanosecond, perfectly suitable for the required precision. Two different runs have been performed for the two different vertex finders used, the first is based on clusterisation of selected lines, the second on uses the histogram method. The results are reported in table 6.2. The last entry also includes the total elapsed times; it is extracted from the

	clusters of lines		histograms	
Phase	elapsed time (ms)	std dev (ms)	elapsed time (ms)	std dev (ms)
init	0.21	0.41	0.21	0.41
tracklet finder	3.37	17.32	3.37	17.22
tracklet matcher	0.89	6.67	0.89	6.64
vertex finder	0.08	0.59	22.34	109.90
total	4.55	24.68	26.81	133.00

Table 6.2: Summary of the benchmarks on CPU implementations of primary vertex finders. Two approaches have been measured, the one using the cluster finder is faster but not suitable for easy parallelisation, the one using the histograms is slower on CPU but easier to implement in a parallel fashion.

distribution of the totals. The method based on cluster of lines is more than five times faster than the histogram-based one. However, it is not suited to be coded efficiently on a GPU because of the difficulties in translate it into a parallel version but also on the lesser agile management of memory allocations on a GPUs, very frequent operations especially in finding clusters of lines. Although it is perfectly feasible to do a kind of rough adaptation, it either would cost in terms of overhead in (de-)allocations due to the resizing of the vectors containing the information on the contributors or it would require preemptive allocations of large buffers even for those cases in which frames are barely populated. This choice will be better contextualised in chapter 7 and discussed into further details.

Chapter 7

Primary vertexer implementation on GPU architectures in the O^2

The parallelisation of the primary vertex reconstruction consists in dividing the total workload of the algorithm in different pieces not logically inter-dependent and executing them synchronously. In case of an ideal parallel execution of the algorithm, the time spent to compute the whole load is reduced of a factor equal to the number of pieces that the original workflow was divided into. Parallel computing is enabled by hardware architectures such as multi-core CPUs and GPUs, which allow the developers to efficiently run parallel programs thus obtaining faster performance compared to their sequential counterparts. Parallel program executions on CPU and GPU share some common traits but also crucial differences. CPUs perform operations on data elements following an order that depends on which operation has already been completed. The development of an efficient parallel version of an existing "serial[i]" program for CPUs consists in finding the largest subset of operations in the algorithm that can be simultaneously executed, without slowing the processing of each part. In this way the same amount of operations is processed in a fraction of the time. The remaining serial part is made of instructions that are dependent to each others, meaning that each one cannot be run until when the result of the previous operation is computed. Therefore, theses operations have to be run in a sequence with no further optimisation. A GPU-oriented program selects which operation to do on a data element on the basis of its position among other data elements. Parallel programming for GPUs consists in processing the input data by using a parametrisation (a coordinate system) to decide, among data elements, which one has to be processed and what different instructions have to be performed on it. GPU computing is then a viable option when the workflow is linear[ii] and well defined prior the execution of the program: the probability to

[i] A "serial" program is a program that executes all of its instructions in a single sequence.

[ii] The workflows that can be efficiently run on a GPU are characterised by the absence of logical branching (i.e. no "if" statements).

have a deviation (branching) in the logic of the execution is very low.

The multi-process nature of the O^2 framework, where each Data Processing Layer (DPL) device is represented by a process running on a *many-cores*[iii] cluster, extends the concept of *data parallelism* to more than one computing node. In the reconstruction process layout, multiple instances of DPL devices responsible for the reconstruction are distributed across multiple nodes and data taken from a temporary buffer disk are delivered to them. The first trivial parallelisation is performed at the level of the single *frame* to be processed. The processing of any single frame of raw data is not related to the previous one or to the next. Each device is represented by a single-thread process and it is possible to have on the same node more replicas of the same device on the same workflow topology, because of the scaling capabilities. Replicas can be placed also on the same node, depending on how the deployment of the workflow has been optimised by the DPL. Computing resources, on the single node, are efficiently used because the processing of a single frame is done on serial code running in a single-core fashion but in total many frames are processed concurrently by a larger number of processes. This also explains why a parallel version of the primary vertexer based on *multi-thread* would not provide any actual advantage in having threads to compete for resources with other processes. The advantage of offloading parts of a workflow on a GPU grants more availability for resources on the host, literally freeing cores on CPUs and leaving them to other executions. Furthermore, there are some algorithms that scale very well with the number of available computing units, that on a GPU are two orders of magnitude more with respect to CPUs. The primary vertex reconstruction of the ALICE ITS is an algorithm with such a feature.

In this chapter the design and implementation of the vertex finder in its histogram-based version on a GPU architecture will be presented and discussed. The next section is dedicated to the description of those parts of the algorithm which benefit from a parallel execution and the working strategy together with the adopted techniques will be discussed. Later in the chapter, it will be illustrated in details the implementation performed by using different frameworks related to different GPU brands.

7.1 Parallelisation of the primary vertex finder

Rationale in designing a parallel version of an algorithm relies on the identification of the part of the workflow that can be executed at the same time, minimising the inter-execution communication. Tracklet finder (TF) and tracklet selection (TS) phases are included in the category of "embarrassingly parallel" problems. As extensively explained in previous chapters, the tracklet finder performs comparison between clusters on two different layers by using two nested loops; the tracklet selection is computationally similar but it is done with pairs of tracklets sharing one cluster. The complexity of the problem is proportional to the square of the number of clusters for both. For the **tracklet finder** the idea is to use the large number of computing units available in a graphics card to process partitions of the iterations at the same time. Data involved in matches are only read and never overwritten, the routine is not recursive and the data related to the same layer are not

[iii] it defines a computing architecture constituted by multiple machines (a cluster) possibly equipped with *multi-cores* processors

dependent. Outer loop iterations are pinned to a cluster in the middle layer, used as a starting information by the inner loop which iterates on data related to adjacent layer (Layer 0 or Layer 2). In the parallel version, each iteration of the outer loop, that in a sequential one is unrolled one after another, is run by a different thread synchronously, thus allowing for a faster execution. Each thread takes cluster on the middle layer and executes the inner loop which iterates to compare it with clusters of the selected adjacent layers, performing the same selections used for the serial version of the code. Different models of graphics card have different number of computing units and this parameter regulates the number of parallel threads that can be run at the same time. If C is the number of computing units available on a given board, one can assume in the best scenario to have C parallel threads running at the same time[iv]. At present, modern cards are equipped with a number of computing units which is of the order of thousands and the same applies to the corresponding threads(see 6.1). The order of magnitude of the number of threads is comparable to the average number of particles produced in the single collision. In many cases each iteration of the outer loop is executed in parallel on different threads and this allows to achieve the maximum reduction of the duration of the whole tracklet finder to the duration of the longest inner loop.

This improvement can be understood also by estimating the complexity of the parallel approach, which is different from the calculation done for serial version (see section 4.20). Usually, the effect of the parallelisation of a loop of N iterations is translated in reducing the complexity of the loop itself. This is because in the same number of CPU cycles that are needed to accomplish the single iteration on a sequential loop, N iterations are computed instead. From a mathematical perspective, the complexity of the *serial* tracklet finder (T^{TF}_{serial}) is then reduced to its parallel definition ($T^{TF}_{parallel}$):

$$T^{TF}_{serial}(N) \propto N^2 \underset{C \simeq N}{\rightarrow} \mathcal{T}^{TF}_{parallel}(N) \propto N. \tag{7.1}$$

The parallelisation does not change the total number of iterations performed, which is given by the product of the number of iterations of inner loops times the iterations of the outer ones. They are simply computed more efficiently and faster. In the serial execution the tracklet finder is run two times, with different pairs of layers. They can potentially be executed in parallel since there is no interaction between two executions. However, although for collision systems with lower particle multiplicity (such as *pp*) this can really be the case, for Pb–Pb collisions the amount of allocated resources, especially the memory, can be too large with the presented implementation. This possibility will be investigated in future developments, when there will be a definitive decision on the GPU architecture finally adopted. Once the architecture is defined, one will know how many logical resources will be available for the execution of the primary vertexer. In the present work the matching of Layer 0 (L_0) with Layer 1 (L_1) and then L_1 with L_2 are executed in sequence.

Concerning the **tracklet selection**, the best parallelisation pattern is very similar to the previous one: clusters on the middle layer are the data on which threads are "pinned".

[iv] This is a simplification, because the scheduling of the instructions for the single thread also depends on the availability of the memory registers which are finite for a given thread pool (warp/wave). While the threads can be really spawned at the same time, the actual synchronicity of their operations depends on the number of used registers. Otherwise thy are scheduled to maximise the resource usage.

Each thread processes all the tracklets associated to that cluster and such a process is constituted by two other nested loops. Parallel distribution of the workload on threads reduces the complexity of one order of magnitude because all the clusters are processed at the same time. The transformation of complexity for tracklet selection from 4.22 is reported in equation 7.2. Although in this case only the majorant was estimated, remaining on the safe assumption, the complexity reduction is:

$$T^{TS}_{serial}(N) \propto N^6 \underset{C \simeq N}{\rightarrow} \mathcal{T}^{TS}_{parallel}(N) \propto N^5, \tag{7.2}$$

being T^{TS}_{serial} the complexity related to the serial version and $T^{TS}_{parallel}$ the complexity of the parallel one.

Finally, the **vertex finder** in its *histogram-based* approach, has been ported to a parallel version. It was not possible to port the *cluster-based* one because of its intrinsically recursive structure where the internal states, the clusters of lines, are updated at each iteration, with a lot of memory resizing and movements. The creation of the same logic with many threads running, causes the execution to have many synchronisation points and locks[v] to allow for consistent and deterministic execution. A clear example concerns the addition of one line to a cluster. In the logical perspective of the workflow, the distribution of the cluster-based vertexer is not convenient because the outcome of each iteration is not predictable (lines can or cannot be compatible with the cluster) in a general way and therefore not fairly shareable across many instances. The inclusion of the line for the vertex determination cannot be determined a priori in a unique way, because the position of the centroid moves at any update. To maintain a parallel workflow and realise any type of ordering is hard. This is due to the fact that the scheduling of threads accessing a specific resource is not naturally determined, unless a clear order has been instantiated. To impose such an order will automatically imply a "serialisation" of the execution, where each thread has to wait for at least another one to complete its previous step.

Differently with respect to the *cluster-based* vertex finding, the *histogram-based* approach can benefit much more from parallelisation. It has a larger combinatorial phase compared to the other, due to the fact that all the lines are compared one by one. However, each combination of indices of the nested parallel loops can be unrolled and performed at the same time. Complexity T^{LM} of the triangular loop for the line matching (LM) is of the order of the squared number of lines. Compared to the combinatorial in the cluster association, the size here is much smaller and very often the unrolled triangular loop is processable in a single execution, meaning that the number of combinations is less than or equal to the number of available threads. For this specific scenario the magnitude of the complexity is flattened, since it does not depend on the number of lines and it is performed a single comparison and centroid calculation in each thread. Complexity function $\mathcal{T}^{LM}_{parallel}$ is described in 7.3, being L the number of lines and N_t the number of available GPU threads:

$$T^{LM}_{serial}(L) \propto L(L-1) \underset{L \leqslant N_t}{\rightarrow} \mathcal{T}^{LM}_{parallel}(L) \propto O(1) \tag{7.3}$$

[v] A "lock" is an object that abstracts the unique access to a specific resource, ensuring that it can be read or modified by only one entity at a time.

When the number of combinations is larger than N_t there is an automatic repartition of the workload to consecutive iterations (as explained in section 7.2). For each pair of lines, the position of their centroids is computed in parallel and the results are stored into a common memory buffer which will serve as an input for the next phase: the creation of the three histograms used to extract coordinates of the vertices.

Libraries that provide optimised implementation of one-dimensional histogram and their related methods, are included in the two frameworks used for the GPU programming, described in sections 3.2.2 and 3.2.4. The two used libraries, respectively *hipCUB* for ROCm and *CUB* for CUDA share the same rationale and structure in their inner implementations: thus considerations done hereafter are valid for both of them. Data are loaded at the initialisation phase of the GPU histogram. Complexity $T^F_{serial}(C)$, with C being the number of computed centroids, is then reduced of one power since the filling of the histogram is done in parallel. Under the same hypotheses also the histogram filling is flattened, as reported in the following equation:

$$T^F_{serial}(C) \underset{L \leqslant N_t}{\rightarrow} \mathcal{T}^F_{parallel}(C) \propto O(1). \tag{7.4}$$

The same libraries also provide methods to find the maxima in the histograms. On the GPU it is implemented as a *reduction*: an algorithm that uses a binary combination operator to compute a single value out of a sequence of elements in input . The complexity of the parallel reductions is linear with the size of the inputs and this translates in a performance, in terms of throughput, that reaches a plateau in case the sizes of the problem are large enough to saturate the GPU. Compared to the serial version complexity is the same, since the maximum is found with a loop running once over all bins of the histogram. The added value is to have a native implementation on GPU of the method, which is compatible with **cub* histograms, both for CUDA and ROCm. Remaining GPU kernels of the vertex finder, which populate the histogram for the z vertices coordinates, show constant complexity as long as the number of centroids is less than or equal to the number of GPU threads. Each line is assigned to a single thread that computes the coordinates of the medium point with the *pseudo beam* line and stores the z position inside an intermediate buffer. Multiple vertices are separated along their z coordinate and spotted as local maxima of the distribution of z coordinates of the centroids. An iterative procedure will set to zero all the bins used to compute the z coordinate of each vertex and then proceeds to find the next maximum. To avoid corner cases, a maximum number of iterations is configurable. The usage of a selection on the number of contributors is enough in real data-taking conditions, provided that the selection is kept large enough to include the maximum number of piled up events. This is a safe condition for events with low multiplicity. The complexity of this last step is then regulated by this parameter which is kept constant at runtime.

7.2 Matching the parallel design with the GPU architecture

This section will cover the technical implementation details, focusing on the changes made to the algorithm including those driven by the different architectural features and computing capabilities of a graphics card.

The GPU architectures optimise the "single instruction on multiple data" (SIMD) model with deterministic (predictable) workflows. On a graphic card the access to the local memory (cache lines and RAM) is usually faster than a CPU. However, memory allocations and resizing are more expensive in terms of overhead. Any kind of memory operation on a GPU is driven by Application Programming Interface (API) calls that, by using the *driver*[vi], are able trigger actions on the GPU. Very frequent resize and allocations can create latency on the *host-device* communication, hence the rationale was to limit their number as much as possible. To this extent, the approach used was to compute the total amount of memory, required by the average computation, and include all the buffers used by the program into a storing object (*DeviceStore*) responsible of providing access to the memory and capable to perform the higher-level actions on the memory such as general resets, resizes etc. The "store" is instantiated as a data member of the vertexer class, hence at initialisation time, with static configuration which does not change during the processing of the ROframes. The total size of occupied GPU memory is set at the beginning of the run, sizes are statically configured and their maximum size is determined by statically configured parameters. The idea behind that is to allocate a reasonable size, configurable at the beginning of the process, for all the needed buffers and to avoid the reallocations. This allows to use a subset of those buffers for writing data. By storing the sizes of those buffers the *store* is aware of the buffer size which is actually used. The input data size changes not deterministically from frame to frame and the new data containing information on clusters, tracklets and lines are rewritten in the same buffer.

In order to ensure consistent memory management, namely to instruct functions and kernels to read only data of the currently processed frame and not to access "out of bound" stale data, it has been implemented a convenient memory abstraction that resembles the container classes contained in the ANSI C++ standard library which as arrays and vectors. These interfaces allow for an agile and familiar memory management of the GPU, which is beneficial for code maintainability and reuse. It is possible to construct locally (on host) memory buffers that will found a counterpart on the device, by hiding all the actual memory allocations and resizes behind a transparent interface. These *array* and *vector-like* manage all the handling for GPU and CPU calls in their internals, and provide transparent query interface that works both on host and on the device. It is therefore possible to ask for the value of a parameter both if it is on the GPU or on the host.

Concerning the organisation of the GPU memory, there are some techniques adopted to ensure the complete independency of the threads from each others during their executions. The memory location is read by different threads is a safe operation, because it does not change the state of the memory. Despite of this, the output has to be written in a safe way, thus ensuring the absence of conflicts and race conditions[vii] Two different approaches are used in this work, depending on the circumstances, to safely store results of each thread

[vi] A drive in Linux corresponds to a piece of software called *kernel module*, that is responsible to mediate the communication of the hardware plugged to a machine and the system calls of an Operating System.

[vii] A race condition, in this case, is represented by a situation where two or more threads try to write the result in the same memory location. In these cases the developer is not able to deterministically establish the order of the threads, leaving the state of that memory location in an *undefined behaviour*.

computation.

7.2.1 Strided memory access and parallel loop iterations

A commonly used technique in parallel programming is to distribute different iterations of a loop, not dependent each others, on different threads. On a GPU this is implemented by assigning to each thread a global index. The single thread has a local index which is assigned from a pool related to the threads superset that includes it. Threads are logically organised in *blocks*; blocks are then grouped in grids, as described in details in section 3.2. The indicisation of threads inside a block is the same from one block to another, thus making it impossible to distinguish threads by only using their local indices. It is possible to derive a progressive unique number for each thread using the indices that define its coordinates inside blocks and grids and their sizes. The global index `gThreadId` of a thread is a function of the block size along each dimension `blockSize.{x,y,x}` (i.e. the volume of the block). The `gThreadId` index is also function of the coordinates of the block containing the thread inside the grid of size: `gridSize.{x,y,z}`. The formula to calculate the global index of a block `gBlockId` in a tridimensional grid is:

$$\begin{aligned} \texttt{gBlockId} = \texttt{blockIdx.x} + \texttt{blockIdx.y} * \texttt{gridDim.x} \\ + \texttt{gridSize.x} * \texttt{gridSize.y} * \texttt{blockIdx.z}, \end{aligned} \tag{7.5}$$

whereas, the formula to assign a unique consecutive global index to a thread into a tridimensional grid of tridimensional blocks is:

$$\begin{aligned} \texttt{gThreadId} = \texttt{blockId} * \texttt{blockDim.x} * \texttt{blockDim.y} * \texttt{blockDim.z} \\ + \texttt{threadIdx.z} * \texttt{blockDim.x} * \texttt{blockDim.y} \\ + \texttt{threadIdx.y} * \texttt{blockDim.x} + \texttt{threadIdx.x}. \end{aligned} \tag{7.6}$$

Block dimension is limited in such a way that

$$\texttt{blockDim.x} * \texttt{blockDim.y} * \texttt{blockDim.z} \leq \texttt{MAX\_WORK\_ITEMS}.$$

The concept of *work items* is similar across the two different GPUs used in this work and its definition is valid also for the open computing library OpenCL®. The `MAX_WORK_ITEMS` parameter corresponding to the GPU used in this work, summarised in table 6.1, is $1024 \times 1024 \times 64$ for the NVIDIA Titan Xp® and $1024 \times 1024 \times 1024$ for the AMD Radeon PRO WX 9100®. The limitation on the number of threads in a block is proportional to the number of registers that can be used by all the threads and it is limited. The values used to compute the number are limited too, as reported in table 7.1

	`max_work_size[0]`	`max_work_size[1]`	`max_work_size[2]`
Titan Xp®	1024	1024	64
WX 9100®	1024	1024	1024

Table 7.1: Maximum values along the three dimensions of the size of a block for the graphics cards used in this work

Once defined the unique index `gThreadId`, the $i^{th}-$iteration of the loop to be run in parallel will be assigned to the thread with the corresponding global index. This approach is commonly used in parallel programming also on CPU for splitting embarrassingly parallel problems. The difference in this case is that on GPU the indicisation follows a tridimensional indexed grid of tridimensional indexed blocks, whereas on CPU the indicisation of the threads is one-dimensional. As anticipated, the tridimensional sizes of the blocks can be set manually. Since, in the specific case of this work, buffers used as input and output are one-dimensionally indexed, one dimensional blocks (`blockDim.y = blockDim.z = 0`) were used to configure each kernel launch. The declaration of the "dimensionality" is a concept used by the thread scheduler to optimise the assignment of each thread to each computing unit. This step is done at each call of a kernel[viii].

Each thread executes an instance of a kernel and the following parameters are passed to the scope:

- **girdDim**: it is the size of the grid of blocks. It is represented by a `dim3` structure, for the sizes on the three dimensions. Default value is 1.
- **blockDim**: it is the size of a block of threads. It is represented by a `dim3` structure, for the sizes on the three dimensions. Default value is 1.
- **SharedMemorySize**: it is the size of the shared memory allocated for each block of threads. Default depends on the model of graphic card.
- **Stream**: it is an optional parameter. If it is not set, the kernel is enqueued to the main stream. A stream is an endpoint where API calls are enqueued.

These parameters are dynamically determined at launch time by two functions shared with other GPU reconstruction algorithms. They also take into account parameters dictated by the device properties. For instance the number of *CUDA cores* in the Nvidia case or *AMD streaming processors*, can be determined using a general formula that covers also bi-dimensional dominions. In this context, the number of threads to be used is equal up to the number of clusters N_c in the middle layer or to the maximum number of threads. One function returns the size of the block $\mathcal{S}_B$, that is the number of threads to run and the size of the block scales only along the x dimension (y, z dimensions are set to 1). The number of threads along the x dimension X_t is preliminarily compared with block boundaries, to cover corner cases when there are no clusters or their number is greater than the block maximum size. Being $maxThreads.x$ the maximum size of the block for the specific device along x, the value of X_t is computed as:

$$X_t = max(min(N_c, maxThreads.x),1) \tag{7.7}$$

Let $warpSize$ be the size of a thread warp, in case of non-empty input clusters the block size S_B is then evaluated as:

$$\mathcal{S}_B = nearestDivisor(X_t, roundUP(min(X_t, maxBlockThreads), warpSize)), \tag{7.8}$$

[viii] The layout in memory for the data has already been set previously at allocation and transfer time. For specific cases like *textures* or images, GPUs have some builtin structures which optimise the memory organisation that cope very well with multidimensional indicisation.

The function $roundUp(x, y)$ determines the least number, greater than or equal to x which is multiple of y. Function $nearestDivisor(x, y)$ returns y if $x > y$, otherwise the first number greater than x that is a multiple of y is returned. To evaluate the size of the grid, the number of blocks needed to scale the kernels execution, it is necessary to find the minimum number of blocks that accommodates all the needed threads.

Depending on the number of clusters that have to be processed for each readout frame, it may happen that the number of clusters is greater than the number of available threads in the single block. This uncommon scenario theoretically breaks the aforementioned complexity, thus making it proportional to the ratio between the number of clusters and the number of available threads. For modern GPUs this number is rarely greater than 2, meaning that the average complexity is not really affected. The final goal is to have a flexible method able of manage automatically all the scenarios, without adding any further overhead. In other words, it has to be able to do a complete excursion across all the clusters with a fixed-size pool of threads, independently from the number of clusters. To address this case, the used technique is to include the assignment of the $i^{th}-$cluster to the $i^{th}-$thread into a loop, starting from the index which increases its value taking into account the size of the whole pool of threads. At the same time, a check is done to ensure that the value of the index is lower than the number of clusters to be processed. In this way, each thread will start from one cluster and will continue to the next corresponding its current position increased by the volume of the pool. The order of access is thus preserved across all threads. Taking into consideration that `threadId` is an integer number (local to its block) and goes from 0 to $blockDim.x - 1$, the index $\mathcal{I}$ can be derived, for a one dimensional block, as follows:

$$\mathcal{I} = blockIndex.x \times blockDim.x + threadId.x \tag{7.9}$$

and it can increase at each iteration by an offset $\mathcal{O}$:

$$\mathcal{O} = blockDim.x \times gridDim.x. \tag{7.10}$$

In case of only one block, $\mathcal{I}$ corresponds to the regular *threadId* and the increase of the index corresponds to the size of the block, which represents the index right after the last available thread index. Therefore, taking for instance the threadID 0, the thread will run with the first value. In case its value, increased by $\mathcal{O}$, exceeds the number of objects on which the algorithm is running on, it exits. In case, it will jump to the iteration associated right after the last thread in the block. Thread 1 will do the same, going to process the element next to the previous one. In this way, it is ensured that all the elements are processed once by different threads in an automatically scalable way. Threads will run at maximum a number of times equal to the modulo of the ratio between the number of elements (clusters, tracklets, lines) and the number of threads.

Another notable case is the parallelisation of triangular loops. A triangular loop combines two nested loops. The first spans over a sequence of indices, the second has one or both of its boundaries that are linear functions of the index of the outer loop. A triangular loop is used in the last method of the vertex finder to compute all combination of lines, that in the parallel GPU version is done at once. For a single dimensional index space of size n, the number of iterations of a triangular loop is:

$$N_{iterations} = n(n-1)/2 \tag{7.11}$$

The goal of the parallelisation is to define a rule to map a thread index $\mathcal{T}$ (or the one re-assigned by using equations 7.9 and 7.10) to a pair of indices, which correspond to the pairs of indices spanned by a triangular loop. It can be then treated as rectangular loop (ordinary nested loops with fixed sizes). Being $\mathcal{F}_i$ and $\mathcal{S}_i$ the first and the second ordered indices, respectively, they are computed in the triangle as follows:

$$\mathcal{F}_i = \left\lfloor \frac{\mathcal{T}}{n} \right\rfloor \text{ and } \mathcal{S}_i = \mathcal{T}\%n. \tag{7.12}$$

Then, they are compared each other, to maintain the order in the pairing. The condition to be tested is

$$\text{if } \mathcal{S}_i \leqslant \mathcal{F}_i :$$
$$\mathcal{F}_i \to n - \mathcal{F}_i - 2 \text{ and } \mathcal{S}_i \to n - \mathcal{S}_i - 1. \tag{7.13}$$

These formula works for odd and also for even n.

The last critical feature to take into account when designing the kernels, is to efficiently and safely store the results of the computation of each single thread. The buffer reading does not require further safety measures, i.e. threads can safely read data at the same address without interfere each others or implying a complexity increase of the pattern of the algorithm. In this work both tracklet finder and tracklet selection kernels need to store results whose size is not fully predictable. This problem is solved in two different ways. The first adopted approach is to partition the destination buffer in subsets called *strides*. Each thread stores results of combinatorial matches into one single segment. The segments are automatically assigned. Their starting point $\mathcal{S}_p^{\mathcal{I}}$ is computed as function of the index of the thread $\mathcal{I}_{thread}$ and the stride integer size S_{stride}:

$$\mathcal{S}_p^{\mathcal{I}} = \mathcal{I}_{thread} \times S_{stride}. \tag{7.14}$$

This method is very useful if for the results there is not the constraint of being contiguous in memory The size of the combinatorial result[ix] is not predictable, therefore the integer

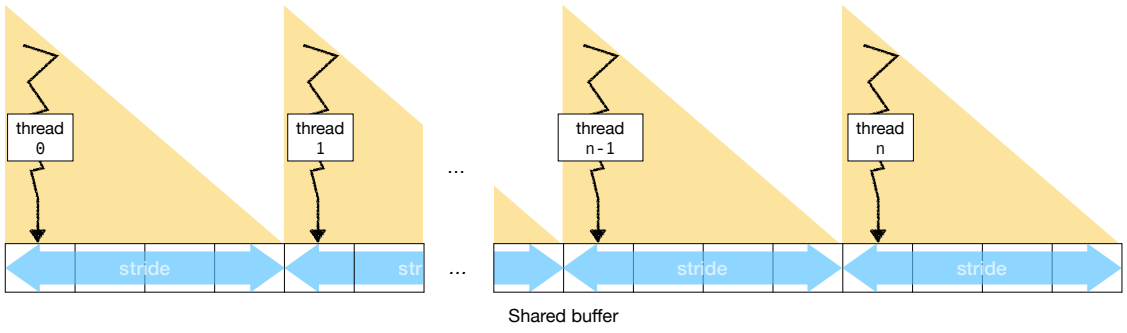


Figure 7.1: Schema of the writing process: depending on the thread index, the starting position for each *stride* is determined by equation 7.14. Their size is equal and statically determined.

[ix] e.g. the number of computed tracklets per thread

size of the stride is set to a value larger enough to store an amount of results larger than the expected average. The *tracklet* object is relatively light to reserve a large number of free slots for a single thread. The numbers are reported in paragraph 7.2.2. An array used as a support, with a length equal to the number of threads, stores the number of found objects and it is compiled to be passed to the next routine. The values, contiguously stored on the array, are used to know size of spans of data to be read. There is no need, for the writing process, to reset the memory until the end of the segment: this is a process that would do some I/O operations.

The kernels used in the vertex finding, which are usually related to the computing libraries that have a static API, require contiguous buffers of memory in input. It is sub-optimal to operate on GPU arrays to spend time compacting the output buffer previously described. Therefore, for the tracklet selection kernel (the process right before the vertex finding), another strategy is implemented, based on running the same kernel two times forwarding different configurations. Each execution runs the same kernel. However, depending on the flag passed in input, the first time it just computes the needed memory layout to save the output and returns its structure. More specifically, it writes the number of found objects into a bookkeeping array, whose size is equal to the number of threads (each thread saves the result in the cell corresponding to its $gThreadId$). This first execution of the kernel is called *dry run*. After the dry run, an *exclusive scan*[38] is performed on the bookkeeping array. This estimates the total amount of elements that are going to be produced in the second run. It also provides the positions on the destination array that are going to store the results. The kernel is run a second time, taking the previous information and using it to know where to store the results, according to the number of object produced. In a general case, whenever it is required to have a contiguous buffer to be filled in parallel on a GPU, it is possible to choose between two approaches: the double kernel run plus the exclusive scan or a manual rearrangement of a *sparse* array to a contiguous one. One has to estimate how much overhead it is caused to run twice the kernel plus a scan than the one caused by memory rearrangements on the GPU. In most of the cases the first option is the best. To move back and forth the memory from device to host and do some rearrangement there[x], is never convenient because of the associated overhead and latency in the data transport.

7.2.2 Parallelisation of reconstruction kernels

Tracklet finder parallelisation. The *tracklet finder* kernel starts from a cluster on the middle layer of the ITS and performs an iteration over a fraction of clusters on the target layer. The same approach is used in the serial code: the clusters are sorted on each layer, meaning that the iteration of the kernel, which aims at finding all possible matches, will span over a set of data that are contiguous in memory. Contiguous threads in a block will operate on consecutive clusters on the corresponding array of clusters on Layer 1. There are possible overlaps in the dominions spanned by threads because clusters on the starting layer can be very close to each other and their span of selected cluster on the target layer often shares some candidates. In figure 7.2 a schema that represents the access pattern of

[x]The CPU and its memory are usually faster than GPU in moving or rearranging the memory.

threads starting from Layer 1 and reading from Layers 0-2, is depicted.

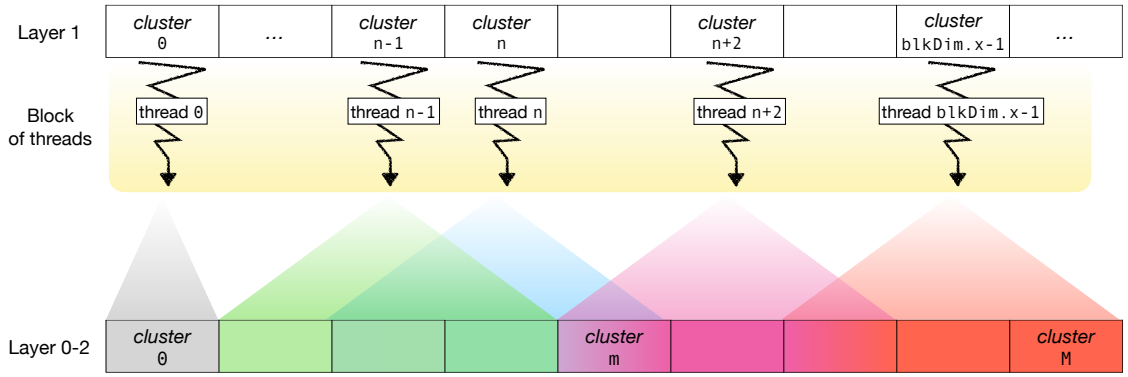


Figure 7.2: The schema shows that a staring cluster is assigned to each thread. After this assignment, the clusters are matched with adjacent ones. Selection windows are centered on the z position around the starting point. Overlaps in the regions of interest are are allowed, because the concurrently reading of data is safe. The idea is to have a shared cache across threads which makes the processing of adjacent threads more efficient.

As long as candidates are in the shared memory of the block, the access is then faster and therefore, once the first thread reads the cluster, it is more quickly accessible by the others. Due to the non-deterministic nature of the associations (clusters may or may not be matched) and to the unpredictable size of spans of target data, it is not possible to benefit from the *local* memory of warps of threads. Future developments could investigate more in this direction. Another important aspect is to safely write the results of the matches on the tracklet destination buffer. This goal is achieved with a *strided* access to the destination buffer. As previously anticipated, with this technique, the output[xi] of the kernel is a pair of arrays. One is used to store the constructed tracklets saved in a sparse mode (interleaved with pieces of memory not relevant to current iteration) and one array that works as bookkeeper by storing the number of tracklets found by each kernel. It is interesting to compute the amount of memory used by applying this approach. A rough estimation of the maximum memory occupancy value is done by making an assumption on the value of some quantities in the most challenging scenarios; to be always under the safest assumption. The length of the segments in which the buffer –accessed in strided mode– has to be partitioned, is determined by the scaling of the parallel algorithm.

If a selection is applied on the azimuthal angle between two clusters (the standard value is $\Delta\phi \leqslant 0.005[rad]$) it has been estimated that in Pb–Pb collisions, for each cluster, a maximum number of tracklets given by

$$T_{max} = 200$$

[xi] For simplicity as "output" of a kernel here it is intended the output data of interest obtained after the kernel run. Kernels are actually `void` functions that do not produce any output in any of the used platforms.

can be tolerated. The bookkeeper array is used to access all the relevant memory position on the sparse one and it is used by the *tracklet selection* kernel instances. At the moment it is not foreseen the possibility to dynamically resize the buffers during the runtime; it is done to optimise its usage in lower multiplicity scenarios, although it is not a hard blocker in the implementation.

Tracklet selection parallelisation. The *tracklet selection* kernel follows a different logic: each kernel runs the first time in a *dry-run* mode filling one slot in a temporary buffer. It corresponds to the index of the processed group of tracklets used as origin point and having the estimated number of tracklets. The temporary buffer is synchronously filled by all the threads, since its size is equal to the number of processed clusters. After the exclusive sum, kernels are rerun and they fill the resulting array to be passed to last kernels for vertex finding. The memory occupancy is therefore dynamically estimated and ti is proportional to the number of found lines.

Vertex finder parallelisation. The kernels passed to the vertex finder merge the information computed in the previous selection phase to estimate the vertex position. The first invoked kernel calculates the centroid position of a pair of validated lines, for each combination of lines. The triangular loop is split and processed by different threads, one per pair, down to a constant complexity ($O(1)$). The results are then stored in a dense[xii] array and used as input seed for histograms GPU-implemented via the **CUB* libraries.

After the vertices have been found, the data are *pulled-out* from the GPU for the first time and stored to output structure to be then passed to next process in line: the track finder. It is worth to mention that as a first approach the GPU-based vertex reconstruction has been developed to create a standalone version. The focus has been put in finding a workflow which starts from the same data that can be provided to the CPU version. This allows the developer to build a GPU pipeline able to process the data in a consistent way which resembles the sequential one based on histograms. In a broader perspective, the integration of this stage with the tracking one, which also has a counterpart on GPU, will involve further discussion on how to be more efficient in terms of resource usage and sharing. Specifically for this case, the two processes share a fraction of the input data, since tracking uses the whole information contained from the whole information related to the seven layers of the ITS. Especially in the GPU scenario, one tries to duplicate memory transfers and duplication as much as possible, to achieve higher efficiency and throughput. In contrast, for the CPU this does not constitute a relevant point since data are already loaded in the scope of the algorithm. Two standalone GPU algorithms now perform slightly different initialisation on data, and for the future developments it will be useful to have one single *entrypoint* to load data at once. In addition, some ordering and index tables are similar and extending the overlap of the two will surely be beneficial to run more efficiently on discrete architectures. The process will be further simplified when the found vertices, needed as seed for the tracker, will not be downloaded but directly read on the board.

[xii] A "dense" array, as the opposite of a "sparse" one, is an array where each position stores a value. A *sparse* array is an array where values are interleaved by empty or uninitialised space, and its requires a rule or a bookkeeper to correctly access the data correctly.

7.2.3 Debug GPU code and intermediate data

An important phase during the development of the GPU algorithm is the cross-check of the results at each step of the algorithm. The access to the information, in intermediate phases, from the GPU requires a memory copy from the device back to the host. More effort is required to perform filtering using the Monte Carlo truth, since the libraries used to implement them, are not compatible with the compiler used to produce GPU binaries. Therefore, the filtering is not performed on GPU but on the host. To compute a filtering based on Monte Carlo labels, it has been necessary to store, inside each tracklet, an index consistent with the ordering of the labels in the Monte Carlo truth containers. Data are then brought back to the host and a check on the matches of labels is performed with the same method used for CPU. This avoids code duplication and ensures consistency.

7.2.4 Porting CUDA primary vertex finder to HIP using ROCm

The first version of a working GPU vertexer has been written by using the CUDA framework from NVIDIA. Motivations were mostly related to the maturity of the libraries, documentation availability and, more importantly, because the TitanXp (see 6.1) graphic card was already available. After the refining of the algorithm and its debug interface, the development moved towards the AMD WX 9100 and its programming platform. The Radeon Open Compute Platform (ROCm), among a plethora of other utilities, offers a script to automatically translate the CUDA code to the Heterogeneous Interface for Portability (HIP) code. The tool is called *hipify* and performs source-to-source conversion of CUDA code to portable C++. It preserves also comments and original code flow. Usually, apart from some non-translatable features, which are strictly bounded to differences between the architectures of the two GPUs, there is a one-to-one correspondence between the API calls and the libraries provided both by CUDA and ROCm. This kind of *mirrored* structure has been intended by design from AMD side, to facilitate the migration of pre-existent code bases to to their framework and thus to their hardware. Concerning the port of the CUDA vertexer using HIP, the tool demonstrated to be very effective, very few language-specific adjustments were needed and they were mostly due to the fact that the CUDA compiler (`nvcc`) and the AMD one (`hcc/hipcc`) shows some different behaviours to manage input data for kernels. For the two frameworks the device properties[xiii] are uniformed to a similar nomenclature, some differences are instead present in the determination of the details. CUDA devices are labeled by a *bitmask* that allow the developer or the program to get information like the number of cores per *streaming multiprocessor*, that provides useful insights on the number of *CUDA cores* as a function of the architecture. For instance, the definition of streaming multiprocessor does not find an equal counterpart in the AMD GPU universe, therefore some further adjustments have been implemented to automatically compute the same related quantities also for the AMD card. Other differences, such as the Dynamic Random Access Memory (DRAM) available, are anyway managed by adaptive code that provides some flexibility as the architecture might change.

Ultimately, to have this *one-to-one* correspondence between the codes developed for different architectures, really unburden the maintainer from doing a double *houskeeping*

[xiii] e.g. quantities strictly related to the architecture, such as: *warpsize, max threads per block... etc.*

of the code, and allows for longer durability of each implementation. This will make it easier to keep both the versions also after ALICE will choose the graphics card for the Event Processing Nodes (EPNs) for the O^2 main cluster. It will be possible then to exploit heterogeneous architectures also on the offline facilities, that in principle are not forced to host the same models of GPUs.

7.3 Performance and profiling

In this section the benchmark of the two GPU implementations of the primary vertex algorithm will be presented and it will be compared with the performance of the serial one. Before estimating the time performance of the new implementation, it is relevant to evaluate the consistency between intermediate and final results, both comparing CPU and GPU and GPU with GPU in case of different vendor models.

7.3.1 Comparison between CPU and GPU results

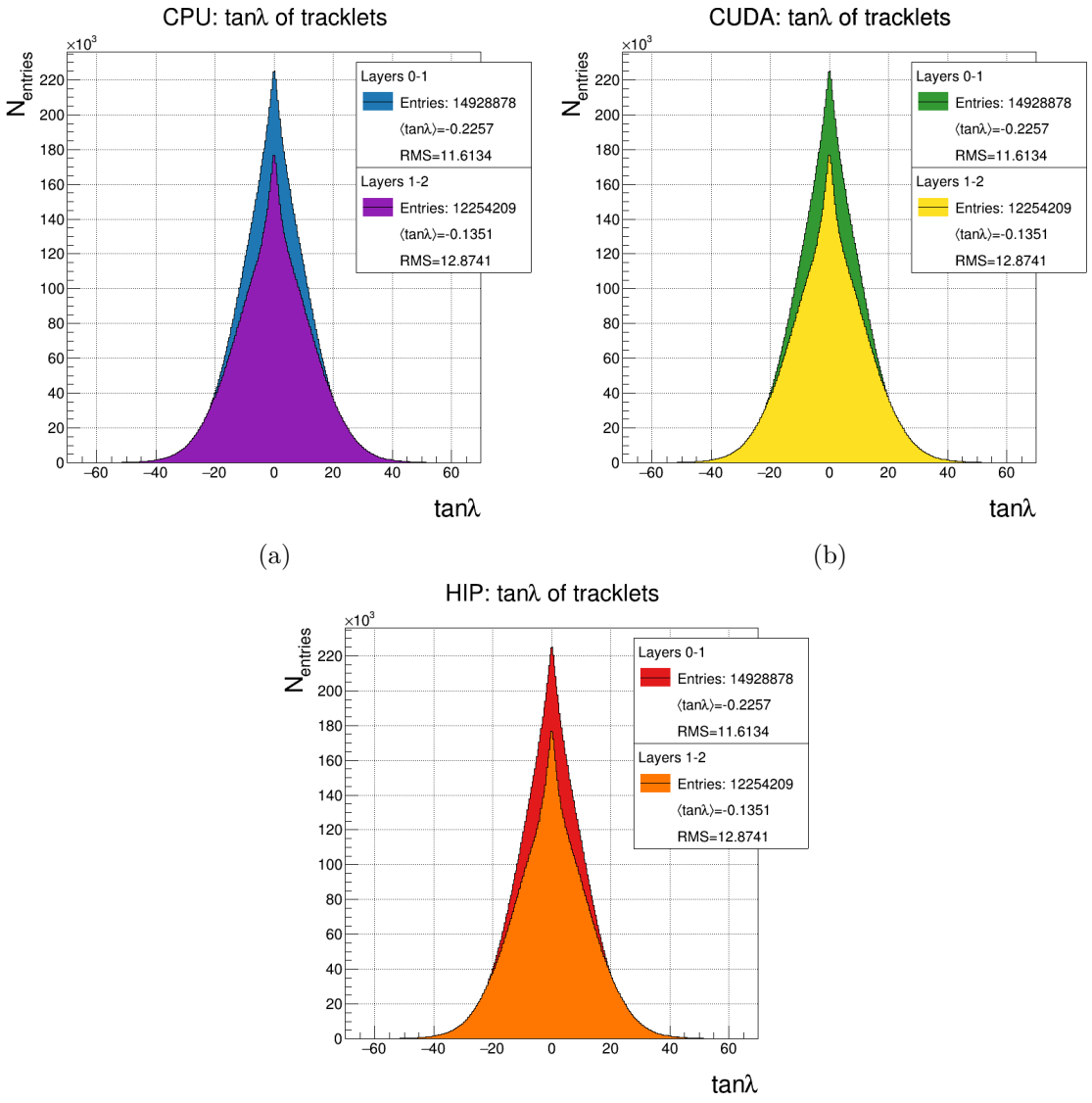
In order to make a meaningful comparison of the results obtained with hybrid architectures (namely the union of CPU and GPU workflows), it is necessary to establish how to consistently compare the data obtained with both technologies. This is particularly important when they share the same interfaces and data organisation. The adopted standard is set to the level of data format representation and rounding and to find its first formulation one has to go back in time to the 1985. The revisions published in 2008[53] and 2019 are those used as a reference in this work. The same standard includes also operations implemented at instruction set level, such as *add, subtract, multiply, divide, square root, fused multiply-add, remainder, conversion operations, scaling, sign operations* and *comparisons*. The rules for rounding and the rounding modes are also specified in IEEE 754. The most frequently used approach is the round-to-nearest-or-even mode (abbreviated as round-to-nearest). The results of these operations are requested to be the same for all the implementations of the standard, or they have to follow a given format and rounding mode. At present, both AMD and Nvidia aim to produce graphics cards for GPGPU computing compliant with the aforementioned standard. Because of the limited precision of the floating-point operations, the rules and properties of mathematical arithmetic do not directly apply. A typical example is the difference in the application of the associativity property of the addition of single precision values (more detailed description in [86]), which leads to results numerically divergent from the theoretical ones even if standard-compliant operations and data representations are used. This is due to the concatenation of rounding functions that happen after mathematical instructions. Depending on the operations sequence, this can lead to relevant differences with respect to the expected value. The order of execution of the mathematical operation is, however, not regulated by any standard. The ordering plays an important role in the optimisation of the executions, and they are strictly related to the target architecture that run on. There are many reasons why the same sequence of operations may not be performed on both the CPU and GPU. The GPU has fused multiply-add while the CPU does not. Parallelising algorithms may rearrange operations, producing different numerical results. A clarifying

example is the *fused multiply-add* case. GPUs, architectures heavily optimised for accelerate matrix algebra operations, implement this as a single instruction, whereas on the CPU these are only achievable with a combination of the *multiplication* and *sum*. The motivation behind this choice is strictly related to the fact that CPU are mainly designed for quick context switching and unpredictable workflows, while GPUs are more suitable for matrix operations and deterministic workflows, where the probability of deviation from a most probable line is negligible.

Keeping into account these differences, the comparison between results of two algorithm implementations on different architectures becomes more sophisticated. A strict check among numerical results is not fully reliable, also in those cases where the output is not a number. The consistency can be estimated looking at the congruence of the products of the computation. Hereafter the discussion on the three main phases of the primary vertex reconstruction are reported together with a direct comparison of the partial results obtained for the two implementations. Each version of the software ran with the same parameters in input and on the same dataset. The serial version used for comparison is the *histogram-based* one, since it has been demonstrated that it has possible parallel implementation on GPU but also for the sake of the consistency, it has a slightly different parametrisation, compared to the *cluster-based* one.

Tracklet finding consistency. In order to evaluate the consistency among different implementations of the tracklet finder a check on the number of found tracklets and on the parameter computed in the executions is performed. The $tan\lambda$ of a tracklet is one of the two parameters calculated in the routines. The plots presented in figure 7.3 report the $tan\lambda$ distribution for the selected tracklets for the three different implementations. For each implementation the distribution of $\tan\lambda$ for two pairs of ITS layers is reported. Two aspects can be noticed in the cross-comparison: the first is that the total number of entries is the same of each triplet of distributions, meaning that both approaches found the same number of tracklets. This is a positive marker of the consistency across the two implementations. In general this would be critical in case of different counts (even if there were just few missing) because, as anticipated, the floating point representation is standard, the selection is performed on the $\Delta\phi$ of pairs of clusters, but such quantity is computed at initialisation time on the CPU. The comparison of clusters pairs is not expected to behave differently on CPU and GPU. The second positive signals are the RMS, averages and integrals of the six reported distributions. To be noticed that both ϕ and $\tan\lambda$ are computed by the GPUs. Since the aforementioned parameters extracted by the distributions are strictly equal down to the fourth decimal number, the results tells that the three implementations give consistent results. For this reason, after this check, the three implementation have been validated.

Trackelet selection consistency. A similar approach has been adopted in evaluating consistency for the tracklet selection phase. The distributions of the angular parameter $\Delta\phi$, obtained for the selected lines with three different implementations of the algorithm, are reported in figure 7.4. Distributions are reported in figure and represent the angular parameter distribution for the selected lines. In this case the number of total entries is almost the same, with negligible fluctuations (a factor smaller than 6×10^{-6}), whereas the mean and the RMS of the distributions correspond to a precision of a micro-radian. The



(a) (b)

(c)

Figure 7.3: Distributions of the *tanλ* parameter for tracklets generated in combinatorial *tracklet finder*. In each plot the superimposition of a larger distribution of tracklets from Layer 0 to Layer 1 and a smaller, related to external tracklets, is presented. Both integrals and parameters of each distribution are equal to the corresponding one related to the implementation on a different architecture. This is a good indicator of consistency across diverse software solutions.

aforementioned difference is due to the fact that `tracklet` objects are constructed on the three different architectures and their data members are there computed, being sensible to the effects described in 7.3.1. Notably, in this case, the effects of rounding and associative arithmetic are very mild, with a fraction of lost lines of the order of 10^{-5}. It is important

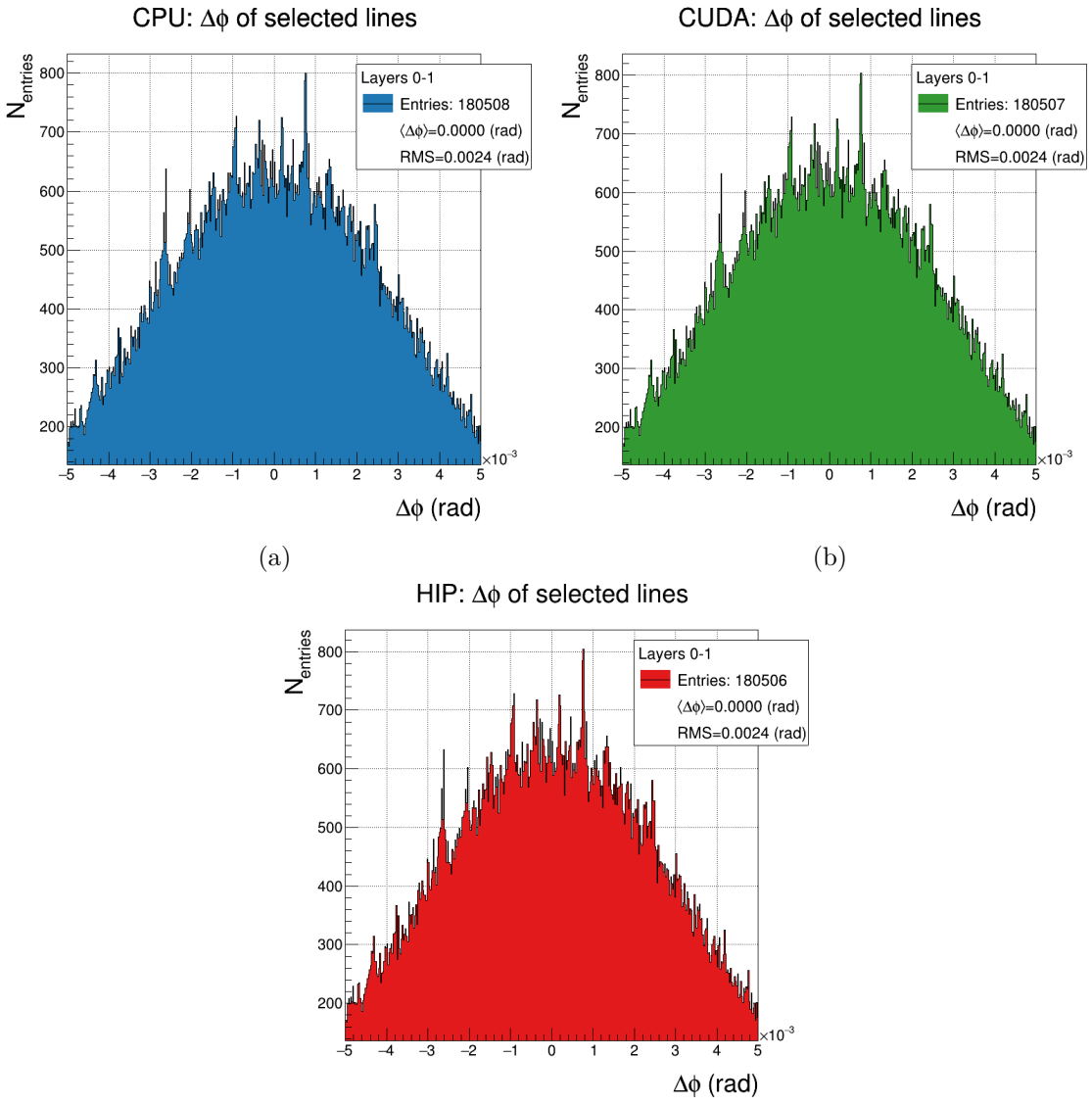


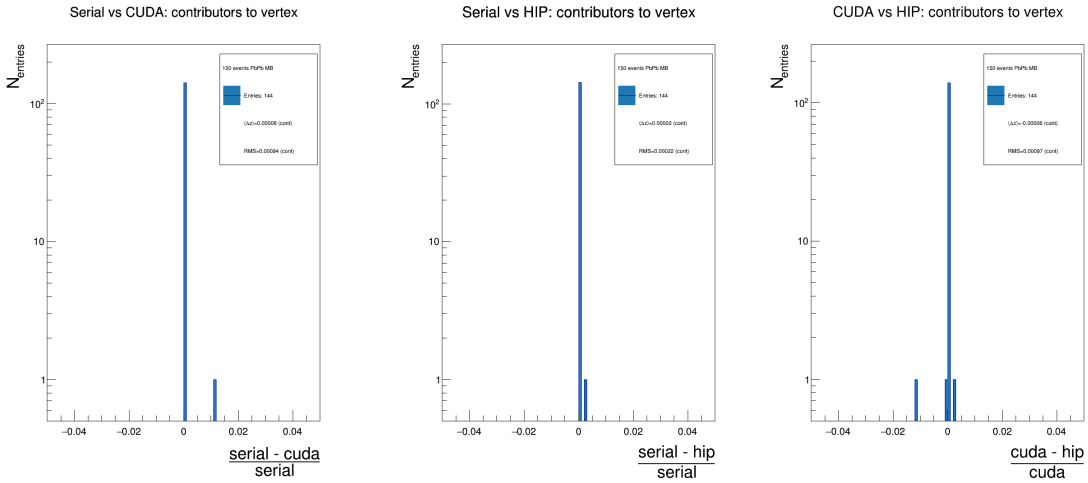
Figure 7.4: Distributions of the $\Delta\phi$ parameter for three different cases of *tracklet selection* algorithm implementation: the blue one (7.4a) reports results of the CPU version, while the green one (7.4b) refers to the results for CUDA and the red one (7.4c) is for HIP. The total amounts of entries differ each other of a negligible factor that in average does not affect the final calculation and the parameters that characterise distributions. This is a good indicator of consistency across diverse software solutions also for what concerns the tracklet selection, despite the different results in arithmetical associations.

to recall that, after the tracklet selection, lines are re-created starting from the clusters, to compute the full information associated to their equations. This phase is again sensible to the different architectural design, therefore the calculated quantities like the direction

cosines of the lines are going to be not perfectly equal compared each other and in some cases the difference might be large enough to be discarded. Trigonometrical functions are usually implemented at the instruction level, as truncated analytical series to some order. Therefore, even assuming that the order of approximation is the same for each GPU and CPU it is not guaranteed the order and the nature of rounding in the calculation. Similarly as it happens in the *tracklet finder*, where the $\tan\lambda$ is approximated to its polynomial expansion, also for the direction cosines there will be some approximation, resulting in not-perfectly equal single precision results. This is unavoidable, due to the fact that `line` objects are larger in memory than the `tracklets` and after the tracklet selection only few of them are kept, improving the efficiency in the usage of memory. The computation of the full information for `lines`, later used to fit the vertex position, is postponed with the intent to save memory. For measuring the consistency of the last *vertex finding* routine it will be performed a direct computation of the difference (residuals) of the computed results to be able to measure the final impact on the estimation on different implementations.

Vertex finder consistency. Last check done for the algorithm is the one related to the vertex finding approaches. The part of the code which start from the triangular loop to calculate the distances of closest approach (DCAs) to then fill the histogram and make the extrapolation, is the most arithmetic-intensive section of the whole algorithm. In this case the consistency check consists in the evaluation of the reciprocal displacement of the final found vertices and as an intermediate result, it is possible to test whether there are critical fluctuations in the number of contributor lines used to find the vertices. Due to the small fluctuations in the number of found lines it was expected to have small fluctuations also in the number of contributors. The number of contributors is not constant for each event, therefore it is meaningful to measure the impact of the fluctuations over the total amount of contributors for each event. In figure 7.5 the cross comparisons of the difference between couples of different implementations are reported. The quantity reported is the signed difference between the number of contributors normalised to one of the two set (specified in each plot). This provides further information on the relative impact of the fluctuation related to the single event. Absolute fluctuations were also investigated, showing at maximum a ± 1 contributor oscillation for each test. In one case the relative fluctuation is close to the 12%, representing the maximum impact registered for a low-multiplicity event (8 contributors).

In figure 7.6 the plots of the residuals on the found vertices coordinates are reported. This estimator is intuitively the most relevant, because the aimed goal is to have three algorithms that can run on three different architectures and provide same results within the accepted resolution for the vertex finding. The minimum requirement to be compliant with the track finder is to have a value which goes from hundreds of microns to one millimeter. The contribution to the final resolution is negligible since the average residual in each case is less than 1μm. The plots show that the results obtained from CPU serial implementation, CUDA parallel implementation and HIP parallel implementation on GPUs are compatible within 1μm which is enough to accept them as interchangeable methods to estimate the vertex position using the histogram-based method. On the portability perspective this represents a strong point since it allows to run on heterogeneous hardware and obtain compatible results, opening the possibility to exploit different architectural configurations also for offline data processing on clusters which own different setups compared to the



(a) (b) (c)

Figure 7.5: Distributions of the number of contributors for three different set of comparison: Serial vs CUDA (7.5a), Serial vs HIP (7.5b), CUDA vs HIP (7.5c). The distributions are normalised to the total number of contributors of the first element in the difference, to estimate the relative impact, in each event, of the fluctuation of the final number of contributors. In the analysed dataset the maximum value is less than 15%, whereas in every case most of the times the number of found contributors coincide.

EPN.

7.3.2 Time performance benchmarks

In this section the time benchmarks for the three methods are discussed. In table 7.2 the elapsed times spent for the each function call that involves GPU operations are reported. Tests have been run on the same dataset used to benchmark the CPU version whose results are summarised in table 6.2. The results from benchmarks are put in a distribution and then the average value and the standard deviation are extracted. Measurements are done wrapping each call into a function that profiles the elapsed time using the `std::chrono` library, with a precision down to the order of μs that for this context is good enough. There exist at the time of this work, both on CUDA and ROCm side, profiling tools to monitor deep into details different aspects of the execution of the only part running on the GPU. For CUDA, benchmarks have been also cross-checked with specific tools such as `nvprof`[68] whereas for ROCMm an issue with code not related to this work forced to *freeze* the library to a specific version (v2.7.0) that did not have an advanced profiling tool yet implemented. However, after the issue will be solved, and the ROCm version upgraded, there will be the possibility to use the `rocm-profiler`[13] tool to cross-check also the AMD code. To measure detailed insights on the memory throughput on the GPU, timing for API calls and elapsed time spent by kernels is really useful at development stage. Ultimately, to decide whether a method is suitable, the whole processing of the readout frame needs to be measured. Therefore the wall-clock time for each vertexer method is

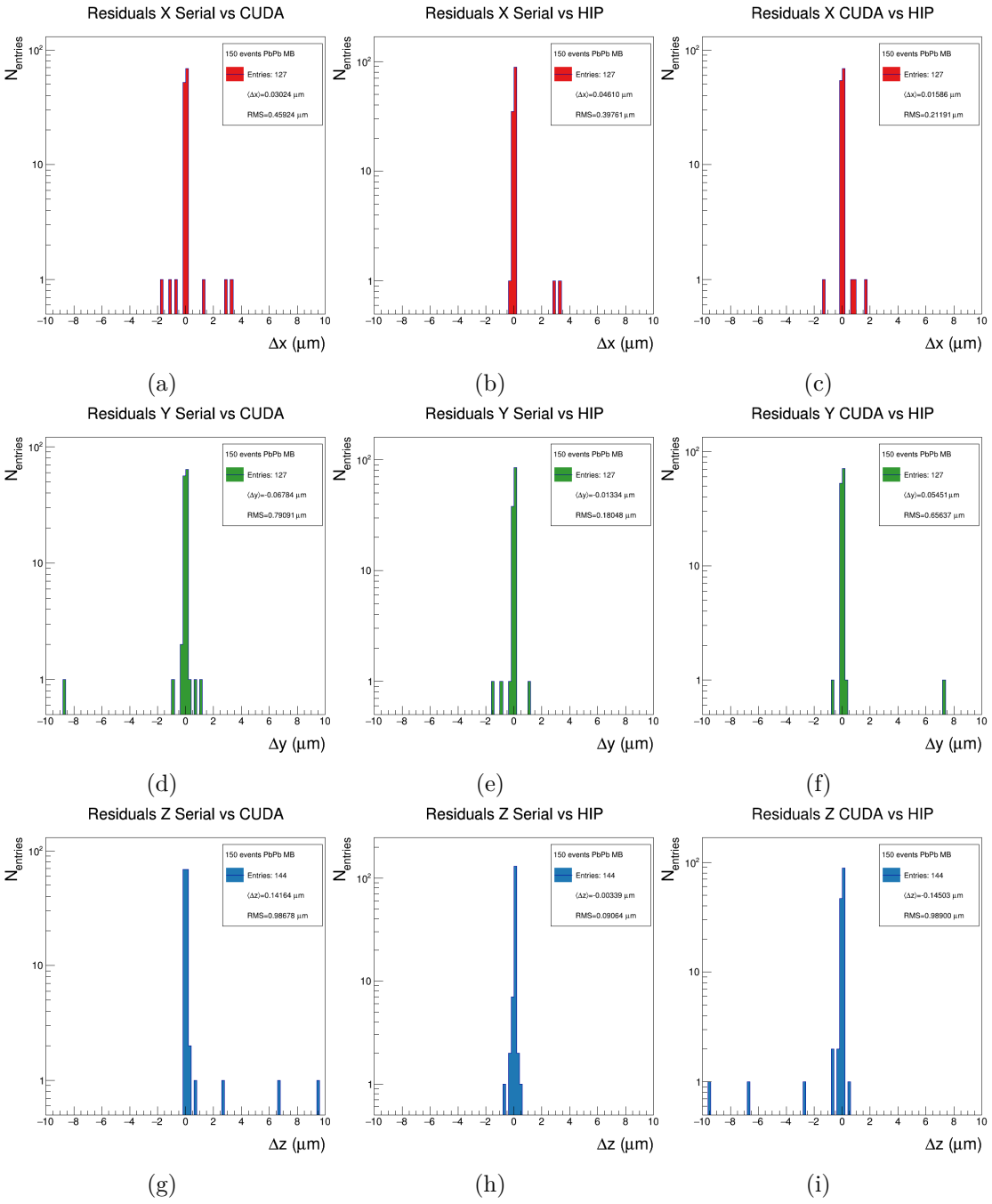


Figure 7.6: Logarithmic-scaled distribution of the residuals for x (upper row), y (middle row) and z (lower row) coordinates. From the left to the right column the comparison of Serial vs CUDA, Serial vs HIP and CUDA vs HIP is reported respectively. The results are presented for tridimensional coordinates of found vertices, aiming to show that obtained results are mutually compatible within a 1μm tolerance window.

	Serial		**CUDA**		**HIP**	
Function	avg	std dev	avg	std dev	avg	std dev
Init (ms)	0.21	0.40	0.35	0.52	2.86	54.05
Tracklet finder (ms)	3.37	17.21	0.01	<0.01	0.14	2.72
Tracklet matcher (ms)	0.89	6.64	0.01	<0.01	0.02	0.02
Vertex finder (ms)	22.38	110.10	1.76	9.17	2.32	6.86
Total kernels (ms)	**26.64**	132.80	**1.78**	9.17	**2.48**	7.37
Total (ms)	**26.82**	133.10	**2.17**	9.72	**5.72**	70.2

Table 7.2: Time profiling: benchmarks reports have been carried out on the whole combination of GPU and CPU code used to steer the result, average and standard deviation are reported. The **Total kernel** entry represents elapsed time spent in the whole computation after the initialisation phase which is sensible to the first run and negligible in subsequent ones (apart from the HIP benchmark). The **Total** row reports distribution parameters of the whole sum, initialisation included.

a crucial indicator. In table 7.2 two different measurements of the total elapsed time are reported. The **Total kernels** benchmark excludes values related to the `init` function, the actual **Total** benchmark includes them. Two results are shown because of the critical initialisation phase in benchmarks. As previously described during the initialisation, at its first call, all the memory allocations are performed once. Especially in the case of the GPU code, this can be very time-consuming. It has been observed that in the case of the AMD GPU reported in table 6.1, the first initialisation can take of the order of seconds. Also for CUDA the first initialisation is slower than the successive ones, but only on the order of milliseconds. While this does not represent a blocker, since it is only the first run, it is also true that especially with low sample size like the one used to run benchmarks (545 readout frames), this affects the RMS and the average, therefore also a total elapsed time only containing kernels was considered as a useful feature to be shown.

To summarize, it is evident that, the GPU-based implementations are faster that the serial one and it is a factor $4.6 - 12.4$ in the worst and best scenario, respectively.

Chapter 8

Conclusions

In 2021, with the start of the so-called Run 3, the LHC will increase the rate of the collisions delivered to the experiments the rate of both pp and Pb-Pb collisions delivered, to provide enhanced luminosity. To benefit from these enhancement, the ALICE experiment is going to design and upgrade its acquisition system in favour of a continuous readout of the data to achieve up to 400 KHz for *pp* and 50 KHz for Pb–Pb. This new data-taking approach will allow the experiment to address its physics program planned for the Run 3, collecting larger data sample that will improve the quality and the precision of future physics measurements. The large data throughput, caused by higher acquisition rate, will be translated in a larger data bandwidth from the readout frontend and an increased size of the collected data eventually recorded on tape. In the most challenging scenario, represented by Pb–Pb collision system, the throughput of raw input data will be reduced and compressed, from an initial bandwidth of 3.5 TB/s down to 100 GB/s written on tape.

Such a reduction will be achieved in a two-phases reconstruction thanks to the Event Processing Node (EPN) farm. A first stage, called *synchronous reconstruction*, will be carried out as a stage of the data taking process, i.e. in a purely *online* fashion. The synchronous reconstruction mainly includes a full reconstruction of tracks in the Time Projection Chamber and the aim is to compress information related to tracks and at the same time to discard all the noise. This first reconstruction will allow to save bandwidth and reduce the data size on the final storage. to save bandwidth and reduce size on final storage. The other detectors will perform a partial reconstruction with partial calibrations and not reconstructed data will be temporarily staged on a 20 PB disk buffer. A second stage, called *asynchronous reconstruction*, will fully reconstruct all the buffered data for each detector, using the full calibration information. Finally, the reconstructed events will be saved on persistent storage to be later analysed.

A new software framework to address both the online and the offline data processing is under intense development in the ALICE experiment for the LHC Run 3. It is called O^2 and it includes all the pieces of software used online and offline, including the one that will run on the EPN farm The framework has a multi-process structure, powered by a versatile data-transport layer (DPL) implementing the ALICE data model and supported by a flexible, horizontally scalable and extendable message passing engine. The architecture of the engine follows a dataflow model, where the data are static and the topologies of the workflows to be deployed are automatically optimised to access them in the most

performing way.

Both synchronous and asynchronous reconstruction will be computed at the LHC Interaction Point 2 where the ALICE experimental apparatus is located. A dedicated cluster, the EPN farm, is being set up, and will count at least 250 nodes with dual-socket CPU slots and 8 GPUs each on board completely dedicated to the reconstruction. Data reduction and compression are mandatory for supporting the continuous readout schema, otherwise it would be unsustainable to record data at the original raw size. To accelerate the online computation, ALICE is going to offload a large share of the high-throughput phases of the reconstruction to GPUs, thanks to the capability of many algorithms to take large advantage from the high density of computing units available on the boards. Moreover, by moving large portions of the computation on discrete devices, CPU resources are freed and able to be used by other parts of the reconstruction that can be processed in parallel.

The work presented in this thesis is about the design and implementation of the primary vertex reconstruction algorithm for the Inner Tracking System (ITS), the innermost silicon detector used in the ALICE experimental apparatus. Online estimation of the primary vertex position in a collision has multiple advantages: it provides important information on the beam position, that can be used as a parameter for the calibration and the alignment of the ITS and other detectors. The primary vertex position is also an essential and critical information for the reconstruction and it is used as starting point for the ITS-standalone algorithm for track reconstruction. The algorithm presented and described in this thesis is already integrated in the ITS reconstruction workflow, which comprises the cluster finding from digitised detector signals, primary vertex reconstruction using tracklets and track finder. It has been designed to work with Pb–Pb collision systems, characterised by a high multiplicity of charged particles produced (an average of 1000 with peaks up to 2500 particles in central collisions). The algorithm includes also an unsupervised clustering routine capable recognise the *pileup* of multiple events. It is also able to detect the presence of multiple collision points in the same *frame* of data processed, by performing spatial clustering of candidate track segments by applying a selection on their distance of closest approach (DCA). Its first sequential version is a pure C++ implementation that runs on ordinary CPUs. The average elapsed time spent to compute the position of the vertices, added to the average time spent by the rest of the ITS reconstruction (less than 100 ms) is one magnitude lower than the maximum time slot allocatable for the ITS reconstruction. The remaining time will be spent to process optional reconstruction steps, which aim at tagging the data related to an event that presents an interesting topology with additional information (e.g. presence of an already reconstructed physical phenomenon), to provide faster data selection for the analyses.

Two different methods are proposed in the final stage of the primary vertex reconstruction: one based on a clustering algorithm and the second based on three unidimensional histograms, one for spatial direction. The two methods are configurable with a few similar parameters related to geometrical quantities such as the DCA between lines and the number of lines that contribute to a cluster. It has been demonstrated that both approaches gives high purity and reasonable efficiency that can be further improved in the future. Optimal working setups for the determination of the selection values will be tuned when the full reconstruction will be available, estimating the effects on the overall performance of the ITS-standalone reconstruction. Finally, there will be a further configuration that will take into consideration the performance of whole process. Tradeoffs between efficiency

and resolution are a second stage in the commissioning of the online reconstruction workflow. At present, the resolution on the spatial position of the vertices is compliant with the minimal requirement set by the reconstruction performance.

Although the algorithm has been developed primarily targeting the Pb–Pb collision system, it has also been tested in *pp* data, as integrated component of the ITS track reconstruction process, by simply applying larger selections in the azimuthal coordinate, to match low momentum particles with excellent efficiency results. Figure 8.1 shows the results of a test performed on a dataset of 10k *pp* collisions simulated at 1 MHz: in blue the efficiency plot for the reconstructed ITS tracks as a function of the p_{T} is presented. In red and in green the efficiency in reconstructing fake tracks[i] and duplicated tracks are reported, respectively.

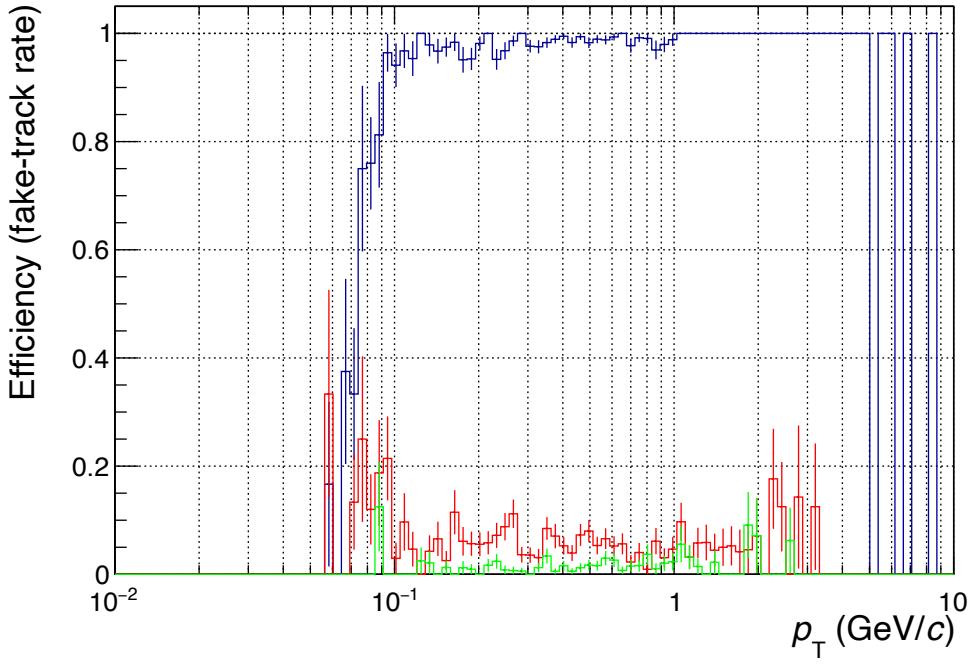


Figure 8.1: Plots of the reconstruction efficiencies for the ITS-standalone vertexer+tracker reconstruction as a function of the p_{T} of the tracks. Reported efficiencies are related to the reconstructed tracks. In blue the efficiency for correctly identified tracks is represented. In red there is the efficiency related to "fake tracks" and in green the efficiency in reconstructing duplicates of tracks.

Considering that the tracker runs only if a reconstructed primary vertexer is present, it is a good showcase that represents how the approach is promising also when dealing different collision systems. For instance, the presented *pp* case is characterised by lower average multiplicity of yielded charged particles and more piled up events per single frame

[i] the attribution of the ID has been carried out with the Moore's voting algorithm[63]

compared to Pb–Pb scenarios. Wider cuts can be chosen to enhance the efficiency at low p_{T}, because the combinatorial background is much lower. The work done in this thesis project will be adapted to the study of the primary vertex estimation in pp and p-Pb collisions systems.

Concerning the GPU implementation of the primary vertex, two versions are presented in this work. They share the same algorithm and logic in the parallel implementation of the histogram-based vertex finder from tracklet. The two versions differ with respect to the targeted supported graphic card: both Nvidia cards and AMD cards are supported through their proprietary drivers, steered by the CUDA and HIP languages, respectively. Their integration with the O^2 framework is enabled by autodetection of the underlying graphical architecture, which allows the installer to automatically produce the required libraries after the detection of the actual hardware. Primary vertex reconstruction on GPU can be selected as a viable option in ITS-standalone reconstruction workflow powered by DPL. The software have been proven to work complying with the requested standard. Some effort in the future will be dedicated in integrating the ITS GPU reconstruction part into a single framework: a dedicated DPL device that is currently responsible for the management of a large part of the reconstruction that is performed on GPU resources. The idea is to optimise the coexistence of multiple concurrent workflows requiring a graphic card for the computation, by providing a single utility to do so.

Consistency between the CPU results with the GPU ones among three implementations has been compared and cross-checked, finding compatibility in the primary vertex determination down to the order of the micron. Vertex finding on both GPUs has better elapsed time performance compared to the CPU version. The tests have been carried out by using GPUs reported in 6.1 which share very similar architectures. Both GPU codes are faster with respect to the CPU implementation of almost a factor ~ 5, in the worst performing scenario, and up to ~ 12 in best cases. While the two GPUs provide almost identical results, the CUDA implementation has the best performance, in terms of speed, if compared to the AMD one running using HIP. The CUDA implementation is also faster compared to the cluster-based CPU version which is the faster among the two purposed solutions. Unfortunately, the sequential cluster-based algorithm is not suitable for being translated in a parallel version, at the moment. The CUDA version is faster of a factor ~ 2, whereas the HIP one is almost as fast as the CPU one.

The comparison is done only in terms of time performance, due to the differences between the two approaches. However, it is worthwhile to mention that as long as the performance of a GPU-accelerated piece of software is as fast as the one running on CPU, it can still be worth to use it to optimise the usage of the CPU, which can then be in that case used to perform some other computations. In the end, offloading is always convenient, since it frees CPU resources. Especially in the asynchronous reconstruction, where TPC reconstruction does not take place, GPUs are most of the time in idle state. Therefore, it is valuable to offload other workloads even in the case they preform as good as the CPU version. In this case the optimisation comes from the better efficiency in the resource usage rather than in the speed of the computation.

Further developments after this work will be oriented in exporting the acquired knowledge related to the O^2 ecosystem and, in particular, to the execution on GPUs of parallel algorithm for the reconstruction, to port many other stages of the reconstruction on graphic accelerators. The goal is to promote the usage of the GPUs by moving some of the current

sequential routines there, and to reduce the pressure and the overhead for all the other pieces of the reconstruction that will still run on the CPUs. For the asynchronous phase of the *pp* reconstruction, having a larger fraction of the reconstruction on GPU will be mission critical to ensure the throughput required to record the volume of data needed for the ALICE *pp* programme for Run 3.

Appendix A

High-energy physics simulations on a High-Performance Computing facility

In the past, the High-Energy Physics (HEP) computing community was among the first pioneers aiming at creating an innovative computing model and strategy to address the execution of large computational problems such as the processing of the large amount of data produced by the LHC experiments. In the early 2000s, the Worldwide LHC Computing Grid (WLCG or just Grid)[25] was invented to pursue the aforementioned purpose by creating a geographically distributed computing network, which gathers many resources around the world and orchestrates them for the execution of computing tasks (simulation, reconstruction and analysis) on HEP data. In the meanwhile, new challenges coming from other fields such as the processing of Big-Data, Machine Learning (ML) and Artificial Intelligence appeared on the scene, with similar requirements in terms of data size but at the same time with differences specifically related to the data processing. The attention of public and private companies started to focus on solving these new computational challenges, modeling the structure of dedicated computing facilities to maximise the efficiency in solving specific problems. As matter of fact, in recent years, large computing centres have invested many resources on the deployment of new High-Performance Computing (HPC) clusters, which aim to satisfy the request of a market focused on solving computing-intensive problems rather than high data throughput ones. Nowadays, the largest share of computational demand is held by HPC-like workflow and it is common to observe how large computing centres, often providing pledges both on the HEP research and to other scientific or third-party companies, try to move to more uniform and power efficient resources trying to accommodate every use case at their best. In the future, thanks to this growing interest towards a HPC model, it will be more frequent to have computing centres providing providing HPC resources to the high-energy physics (HEP) computing in the form of HPC resources.

Until the end of LHC Run 2, the ALICE computing model could not make use of any high-performance computing resource, because of many factors, the most limiting one was related to the typical HPC architecture not being suitable to process the average ALICE payload. Large HPC data centres are in production since a while, and in the past, with the former computing frameworks, tentative studies have been performed to evaluate the suitability of this kind of resources and their possible integration with the ALICE middleware. Unfortunately the results of those tests were discouraging.

With the beginning of the LHC Run 3 things are going towards a different direction. As anticipated in section 2.3, one of the features of the new physics simulations is a more granular structure of the workflow based on an efficient resources sharing among devices. In addition those devices share the workload and perform operations in parallel. This computing model was designed to match the computing schema used for simulations to the HEP architectures. This approach is not very common in the HEP computing panorama because of the usual low fraction $Memory/N_{cores}$ of the computing nodes. A typical simulation workflow is a process that starts from a very small data pool in input, runs very CPU-intensive routines to simulate and transport particles, and then merges partial outputs. Inputs are mostly configuration data that specify the running condition of a general simulation. They are taken from some calibration objects that can be mocked or recorded during the data taking, to emulate the operational condition of the apparatus during the simulation. To this purpose, Monte Carlo simulations are *anchored* to a specific period of data taking and produced data are consistently comparable with actual recorded data. From relatively small inputs these kind of workflows will produce large outputs limiting the input-output (IO) part to very short time. This phase happens at the end of the simulation "job", in the so-called *merging* process, where data are put together associating partial results and writing down on files. In contrast, the data reconstruction in HEP, which is a *data-driven* process, foresees large data inputs and large data outputs since it transforms acquired data in physical objects. The derivate quantities computed do not really reduce the size of the output, unless they are used to discriminate and select a subsample of them[i]. Last part of the computational effort, is represented by data analyses. This is a computing process that starting from a large data input produces a relatively small output at a cost of a heavy IO-intensive workflow. The data analyses present a less standard workflow because it depends on the specific aim of the analysis.

A.1 High-throughput computing at LHC

Historically, HEP computing model was designed to accommodate its most critical use cases: data reconstruction and data analyses. The features of this two tasks are such to embody the high-throughput computing (HTC) paradigm, that generally describes the use of many computing resources over long periods of time to accomplish a computational task. Joint efforts like the WLCG were born to profit of geographically sparse resources, to face the challenge of asynchronously processing of huge amount of data if compared with standards of the end of last century. The WLCG is a global computing infrastructure

[i]This is what actually happens in the EPN cluster.

whose mission is to provide computing resources to store, distribute and analyse the data collected by the LHC experiments. This schema allows to make equally available the data to all partners, regardless of their physical location. The WLCG has a hierarchy of computing centers organised in *tiers*. There is only one Tier 0, corresponding to the CERN computing centre: it owns replica of the data collected by the experiments. They are stored on tape archives, the most convenient storage solution available, thanks to the low ratio between the average written data lifetime and the cost of the storage. Then the tiers 1 are distributed around around the world and there a replica of a subsample of the Tier 0 data on tape is available. Lastly, the Tier 2 does not have a data archive on tape but it can temporarily stage a large chunk of data to more flexibly accommodate the offloaded executions of analyses, reconstructions and simulations. In figure A.1 a schematic representation of the hierarchy is reported. The distinction across Tier 1 and Tier 2 is also

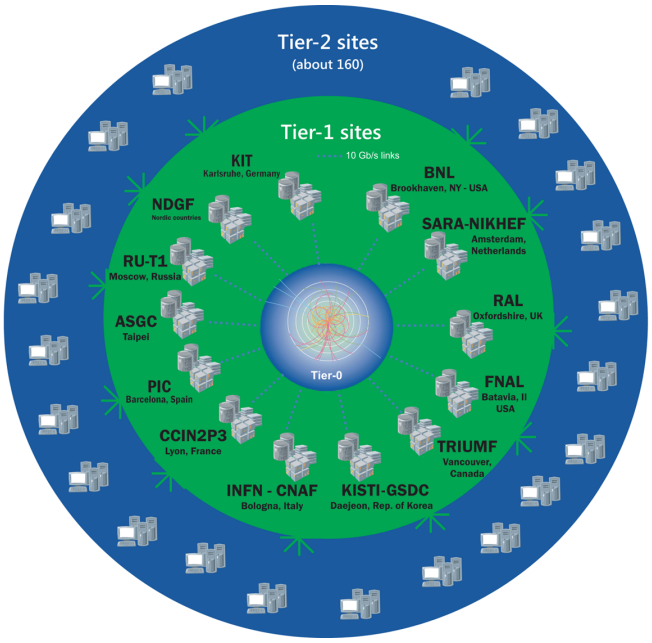


Figure A.1: Tier-ed hierarchical structure of the WLCG, in centre there is the Tier 0, the CERN computing facility.

dependent on the specific experiment computing model. In the case of ALICE, in recent years the distinction has been *blurred* also because of the usage of additional resources as the so-called *Analysis Facilities*. The Grid is realised distributing and geo-replicating data and services around the world. This is to avoid as much as possible to move data around, because of the huge overhead it causes, especially in the past when bandwidth was very limited compared to the data size to be managed.

Generally speaking, HTC workloads are those kind of computational efforts that can be solved by adopting tools like the Grid. The paradigm is to *move* the process execution close to data by managing centrally the job the job submission coming from all the experiment.

A.1.1 The ALICE middleware for Run 3

Fast data access is a crucial point in every LHC experiment. The ALICE experiment during Run 2 had a central file catalog and granted access to files from everywhere around the world, adopting unique URLs. Such a catalog is the so-called AliEn File Catalog (ALice ENvironment)[31]. ALICE has also been among the first experiments using XRootD[7] as storage transfer protocol, providing agile and resilient mechanisms to manage distributed data. From the *tiered* perspective, the ALICE computing model comprises only Tier-1 and Tier-2 centres, local batch systems setup as Tier-3 are not officially part of it.

ALICE production Grid environment is called AliEn, which can be run both on laptops and on dedicated Grid sites as well. The unique trait of AliEn consists of porting the ultimate user interface to user's personal computers. It means that the user can virtually access and browse their files on the Grid space in a seamless way from the operating system shell. The files will appear as they are locally present, then when actually needed locally they are downloaded. This approach really simplifies the interactions of the physicist with the Grid dominion. This is in contrast with other frameworks used by other experiments usually requiring login nodes called *user interfaces* to access the full framework and the Grid middleware. For the Run 3 the perspective is to keep to use the same middleware.

A.2 Simulation run on a HPC cluster

As anticipated, a physics simulation represents probably the closest computing case in the HEP computing field, that can fit on HPC architectures and the possibility to run on a real cluster has been explored. This opportunity arose within the context of a Partnership for Advanced Computing in Europe (PRACE) hosted by the CINECA, specifically on their MARCONI[71] cluster. The aforementioned partnership foresees a sharing of the computing time quotas among the Italian community of the LHC experiments. After obtaining the access to a partition of the cluster, the possibility to run the Run 3 simulations on the corresponding nodes has been investigated.

A.2.1 Setup of the MARCONI-A2 cluster at CINECA

The CINECA centre is geographically very close and logically linked to the main data processing and computing technology research centre of the Istituto Nazionale di Fisica Nucleare (INFN) in Bologna, which is a reference site for the italian computing community and also hosts a Tier-1 for the WLCG: the Centro Nazionale Analisi Fotogrammi (CNAF). The MARCONI cluster is, at the time this thesis is written, a Tier-0[ii] system, co-designed by CINECA and based on the Lenovo® NeXtScale platform, that replaced the former IBM® BG/Q system (FERMI). The partition of the cluster used by the presented tests is named A2 and is based on the Intel® Xeon Phi product family and offers to the scientific community a technologically advanced and energy-efficient high performance computing system.

[ii] in the CINECA classification of its clusters

A preliminary phase of "negotiation" with the administration of the cluster has been done, to point out critical requirements, generally shared across LHC experiments, in terms of software requirements, connectivity and extra-node communication. The centre proximity to CNAF played an important role in separating the hosting of the middleware services from the actual resources where execution of payloads was intended to be run. This approach has simplified a lot the deployment of custom services not optimised for those kind of cluster configurations[iii] and compact HPC farms. Pivotal technologies for HEP computing that has been provided by system administration are:

- a Portable Operating System Interface POSIX-mounted installation of the CERN Virtual FileSystem (**CVMFS**)[84]. It is a HTTP-based filesystem for software provisioning on-demand, capable to rapidly provide software upon POSIX system calls in a transparent and automatic way, downloading it *on-the-fly* from direct source (CERN) or cached systems (Squid[33]);
- an up-to-date installation of **Singularity**[57]: a virtualisation technology very common on HPC farms. It enables the usage of linux containers in "userland" (no need for administration privileges). Containers are nowadays used in many computing fields to isolate job payloads, thus providing a consistent software environment for their execution. During Run 2, ALICE used containers in production environment also for opportunistic computing[36] on the High Level Trigger computing farm;
- an HTCondor[78] computing element (CE) and a SLURM[88] endpoint on the cluster edge nodes, to allow the integration with the middleware of each experiment;
- the connectivity towards CERN and CNAF to access the calibration database and the final storage. The links have been realized by partially routing the network traffic on General-Purpose Network (GPN) and dark fiber[iv] at 40 Gbit/s.

It has been adopted a bottom-up approach, starting from the minimal installation of the software on a single node and manual submission of the job, up to the launch of the job managed by the dedicated CE. node and A computer of the A2 partition of the MARCONI cluster is very far from the typical configuration of a WLCG one. The main features of a HPC node and a node in the WLCG sites are reported in Table in table A.1. The critical difference is in the RAM/core ratio that is very low on the A2 nodes. Moreover, the Knight-Landing architecture (KNL) from Intel® has a peak of frequency per core which is lower if compared with other modern processors ($\sim$ 1.8 GHz for the KNL on A2). In the HEP computing environment, to estimate the computing capabilities of a generic computing centre, the HEPSpec (currently at its HEPSpec06 standard specification) is used: a score which is computed by running specific benchmarks that takes into account many factors as

[iii] computing centres like CINECA provide also computing services to third-party and enterprise customers which are usually very restricting in terms of isolation and security of their computational payloads. On WLCG sites it provided isolation and security for executions as well, but they are built differently often providing outbound connectivity to their jobs.

[iv] optical fiber present in the links but not officially employed, usually used as backup lines in case of malfunctioning of the main connections

	MARCONI A2 node	Typical WLCG node
CPU	Knight's Landing: 68 or 272(HT) cores, x86_64, 0.25 HS06 of typical Xeon	1/2 Xeon-level CPUs: 32-64 cores, x86_64, 10 HS06/thread, Hyperthreading enabled
RAM	1,3-0,3 GB/thread: 96 GB RAM	2-4 GB/thread
Net	no outbound connectivity (internet)	outgoing external connectivity, CVMFS software provisioning
Scratch	no local disk large scratch areas via GPFS/Omnipath	local scratch space 20 GB/thread
Interface	batch nodes via SLURM: Only entire nodes can be provisioned	via Computing Element: Single thread slots
Lease	24h	48h+
Access	Access granted to individuals	Access via pilots, late binding VOMS AAI end-user access

Table A.1: Comparison between typical nodes on HPC and WLCG sites.

computing speed, IO performance and memory availability. This standard metric has been developed by the HEPiX Benchmarking Working Group in order to replace the outdated "kSI2k" metric. Its adoption provides a consistent and reproducible CPU benchmark to describe experiment requirements, laboratory commitments, existing compute resources, as well as procurements of new hardware. In table A.1 are reported also the scores of the two architectures in the first row.

A.2.2 Results for *pp* simulation

The access to the cluster is also provided by a frontend login node, where the users can directly access their account via a secure shell connection (SSH). There, the only allowed method to run on a production worker node is to launch a SLURM batch job[v]. With SLURM it is also possible to launch an interactive shell session which gives the opportunity to the user to interact directly with the node. This is useful for quick debugs and tests on the environment but then the work has been carried out to be launched as a *fire-and-forget* script. On the local read-only filesystem is hosted a Singularity image containing all the requirements for the ALICE O^2 software. This image envelop the batch execution that has been remotely deployed in a consistent environment. After performing some preliminary tests to evaluate the general IO from different partitions, the construction of a quick automation has been performed, namely by writing utility scripts to launch the simulations and to benchmark their execution. Tests have been launched via the SLURM submission interface, as batch payloads. Because of the shorter average duration of a *pp*

[v]In a *batch* execution the user cannot interact directly with the runtime of the job. Most of HEP computing jobs are batch.

simulation compared with the Pb–Pb one, tests have been done on a time window of two hours per job.

In a first attempt, the goal was not to benchmark the elapsed time in executions because per-core (and hence per-process) the time performance are slower by construction. The goal was instead to find the best configuration in terms of machine occupancy to maximise the computing efficiency: the time spent by *each* computing unit (272 cores on KNL, HT enabled) performing operations, divided the Wall Time as the sum of the wall times spent by every CPU. Interesting results are also obtained and presented for the automatic adaptive simulation able to switch from shared-memory approach to the ZeroMQ-empowered[89] one.

Due to the repetitive nature of the simulation it was chosen an arbitrary number of events, large enough to see if some interesting scaling effects can be found. The approach used was to launch multiple instances of the simulations. Each instance has been configured to produce a fraction of 1200 events of *pp* collisions and has been distributed on 20 processes. The goal of this test was to see in first place how the machine reacts with multiple jobs running in parallel and also to see which is the saturation point after which by increasing the number of instances, the elapsed time do not reduce anymore.

The plot in figure A.2 shows the measured CPU efficiencies for different number of concurring instances run at the same time. On the ordinate axis the scale goes from 0 to 272, which corresponds to an efficiency of 27200%. The efficiency is normalised to a maximum value of 272 to better visualise the fraction of resources used by threads compared to the number of (hyper-)threads available on the host. Figure A.3 illustrates the elapsed

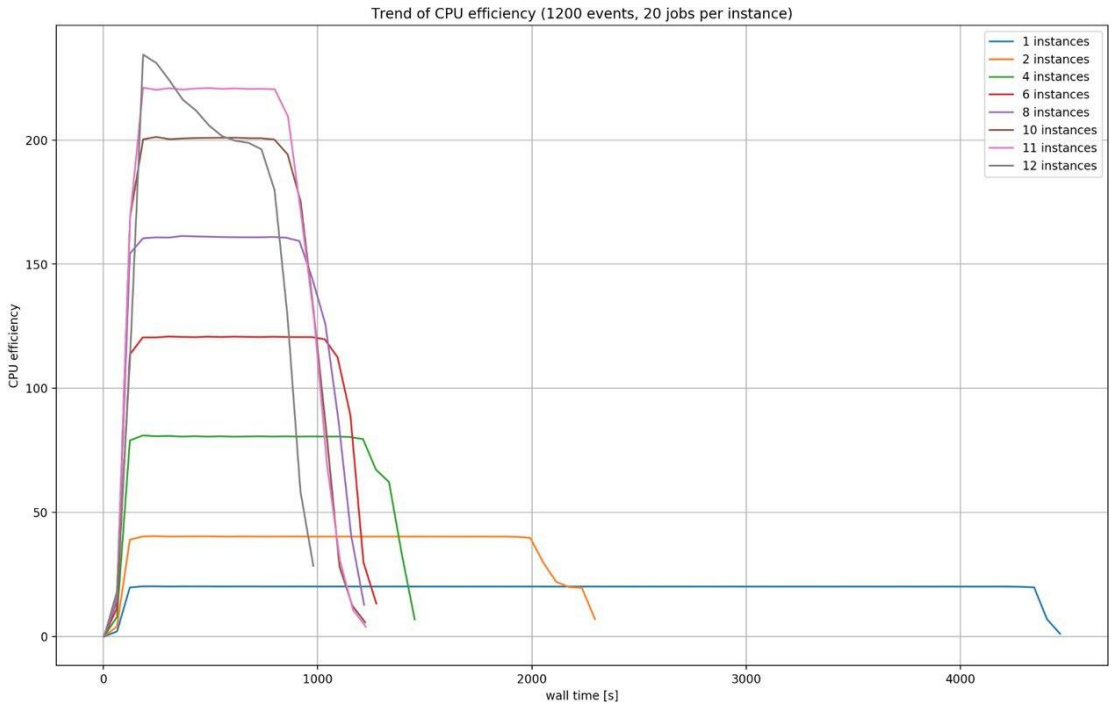


Figure A.2: Total efficiency in the usage of the machine for 8 different numbers of instances. Each instance runs with 20 processes in parallel.

time spent on simulations with different number of processes running in parallel. Multiple tests have been carried out with different number of events to be generated. Cross-like markers are used to indicate those simulations processes automatically switched to the message passing approach thanks to the FairMQ flexibility[80]. It is interesting to see that

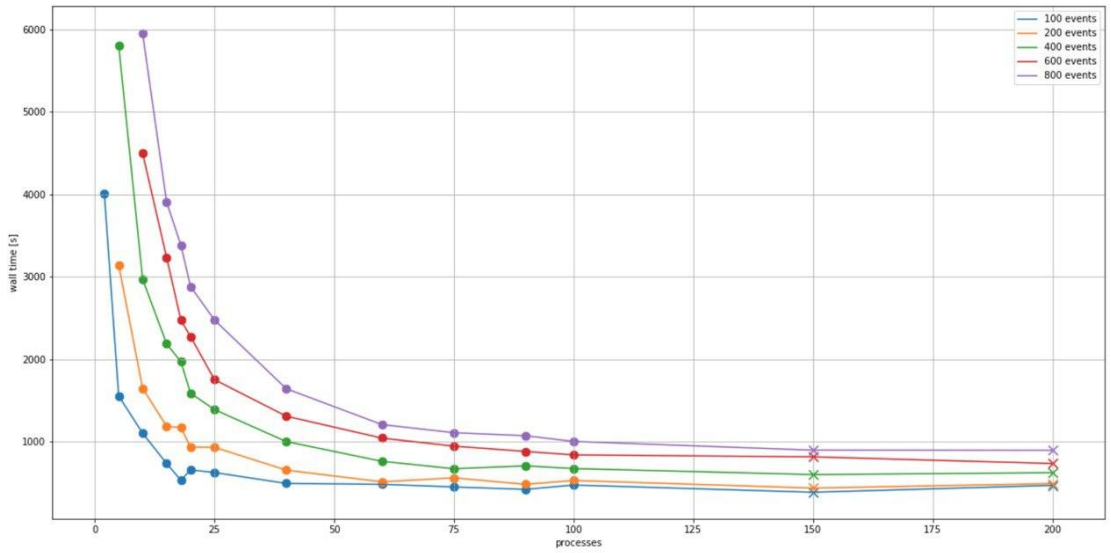


Figure A.3: Elapsed time per simulation with different number of events as a function of the number of processes involved in the simulation. Cross-like markers are used to highlight simulations that communicate with message-passing, whereas the others use a shared memory by default.

for *pp* events the simulation scales well with the number of processes when it is scaled up to 25-30 processes, then the trend does not sensibly improve a lot and the performance saturates. In figure A.3 the cross markers refer to a number of processes ($N_{processes}$) higher than 150. They are differently marked because with such a high number of processes the device that steers the simulation automatically switch the communication channel from shared memory to message-passing for in-node communication. The switch is automatically enabled because of a calculation done by simulation that at the startup checks if it can allocate $N_{processes}$ buffers of 500 MB to communicate with others. If the total amount of required shared memory is greater than the available memory, it automatically commutates to the message passing that has a more flexible memory management and can accommodate all the inter-process communications. The KNL node has 96 GB of memory, but 10 GB are reserved to the operating system, for a total of 86 GB available. Simulation has also to load in memory all the detector geometry data and other services, which at least takes additional 10 GB, thus reducing the available memory for the simulation down to < 75 GB. This triggers the automatic swap. It is interesting to notice that the performance follows the trend with no overhead caused by the switching.

A.2.3 Results for Pb-Pb simulation

Tests on Pb–Pb simulation were done with a more organised approach much closer to the usual job submission method used in the production phase. Largest part of the machinery used for launching the executions is very similar to what already developed for the tests with *pp* simulations. At the CNAF an ALICE VOBox[vi], a piece of middleware used by ALICE to manage the job submission on a given Gride site, has been set up. Jobs have been submitted manually from the VOBox. In real Grid-like scenarios, jobs are submitted from a central "dispatcher" to local VOBoxes, but this it is not going to affect tests of performance because from an integration perspective this corresponds to a configuration that can be done later. The ALICE jobs are submitted from the Grid via an HTCondor interface: a Computing Element has been attached to the VOBox to transform the HTCondor submissions into a SLURM compatible format.

The Pb–Pb simulation is generally more expensive in terms of computing overhead, because of the larger complexity of the execution. Its merging phases, the stage where all partial results are merged to a single output, also called *reshuffling*, have a duration that scales with the number of simulated events. Simulations have been launched with different $N_{processes}$ configurations and the Wall Time has been measured until the CPU usage reaches the maximum in the process. The number of simulated events has been set proportionally to the number of processes assigned. This choice is useful to measure possible saturation of the efficiency by scaling the size of the problem with the number of dedicated processes. It has to be considered that simulations of Pb–Pb events can also use the event splitting across multiple processes, by proportionally sharing the load across the processes. It is easier to see where the performance related to the scaling saturates. The goal of these tests was to investigate the compatibility of the simulation with the KNL architecture by stressing its capabilities to find performance boundaries. Concerning the absolute performance, i.e. the elapsed time for a simulation of the rate of simulated events produced per time unit, they are low by definition, due to the lower maximum frequency of the KNL architecture.

By inspecting the resource usage for one job, it has been observed that at the end of the simulation and transport phase, the global memory and the CPU time vs Wall Time is at its absolute maximum. In figure A.4 the effect for multiple simulation configurations is shown by two plots. The one on the left shows the elapsed CPU time as a function of the Wall Time, while the plot on the right reports the memory usage benchmark as function of the Wall Time. The first plot is useful to have insights for the efficiency of the jobs and its variations. From the slope it is possible to compute the value of the efficiencies and more importantly it shows how the CPU efficiency remains constant up to the merging phase, notoriously characterised by long periods of low CPU activity because of the IO operations. Concerning the usage of the memory, it is relevant to observe how it is reduced during the merging phase, where all the partial results are combined and the simulation objects that have been duplicated by each process during the execution are deleted.

To measure the scaling capabilities for these simulations it has been chosen to benchmark the elapsed time spent by the process up to the end of the CPU intensive part, before

[vi] "Virtual Organization (VO)" is a dynamic set of entities (users or institutions) defined around an ensemble of rules and conditions.

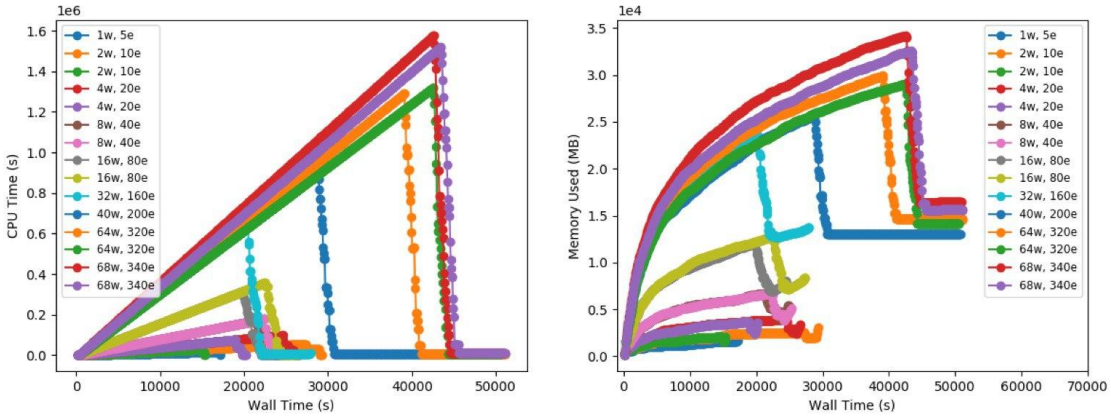
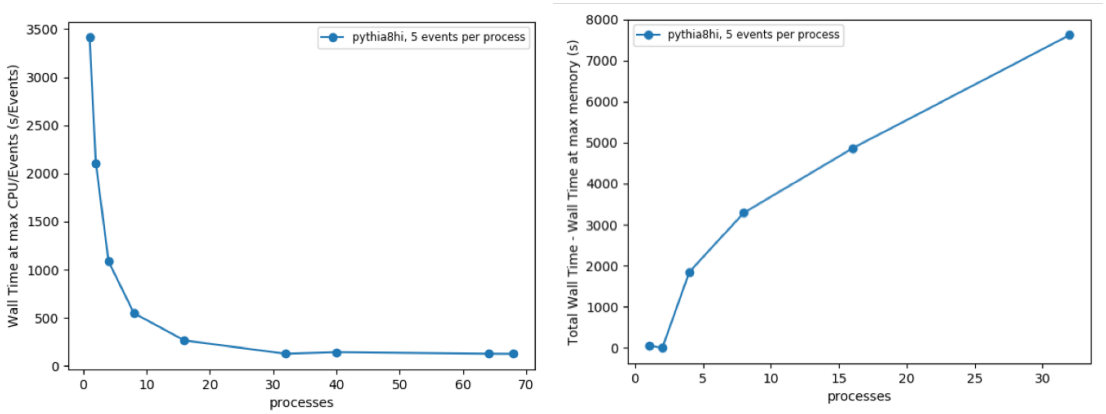


Figure A.4: Results for Pb–Pb events simulation. Plot on the left represents the CPU time as a function of the Wall Time. The plot on the right represents the memory usage as a function of the wall time. The computing intensive part of the simulation ends when maximum CPU Efficiency and memory is reached. During the merging phase a lot o temporary buffers are cleaned and the CPU is substantially waiting idle for the IO processes to finish.

the reshuffling phase. The results are reported in the plots of figure A.5: the elapsed CPU time spent per event as a function of the number of processes is reported on the left, while the plot on the right represents the duration of the reshuffling phase as a function of the number of processes.

There are two main reasons to motivate the separate benchmarks for CPU intensive and reshuffling phase. The first is that with some specific configuration, especially with large number of processes and events, it has been noticed that jobs took too much time to complete. This is caused by the duration of the reshuffling phase, hence they took longer than the allowed Time To Live (TTL). Since no specific optimisation was done on the reshuffling phase at that time, it was decided to anyway benchmark those cases up to the end of CPU-intensive part, to have data also in case in future the overall execution would have been taken less time and stayed in the TTL. The second motivation was to factorise the IO-intensive part (merging) from the CPU intensive one (simulation and transport) and to consistently compare the results only on the part concerning simulation. When the number of processes is around ~ 25, the elapsed time the elapsed time spent to compute a single event stabilises around $200 - 250$ s because the scaling of the speedup given by the event-splitting saturates.

At the time tests were done the merging phase was done by a single process for each simulation, regardless the number of processes used in the simulation. It means that the scaling, multiplied by five, can be considered as a function of the number of events simulated. These results have been very useful also for simulation developers to test on this setup, to spot bugs and improve some aspects previously put in low priority since there were not blockers for other architectures.



(a) (b)

Figure A.5: Elapsed time for the simulation A.5a and the reshuffling A.5b phases. The number of processes is equal to the number of workers.

A.3 Conclusions

Results of testing Run 3 simulation on a HPC cluster proved that the O^2 simulations are able to run on HPC architectures, not optimised for HEP and HTC workloads. The performance obtained with the KNL architecture were not expected, because of the ratio between the number of available cores and the total memory for every node. The peak frequency for single core is also less than a half of a modern CPU running on most of HTC data centres used by the Grid. Both for *pp* and Pb–Pb simulations the scaling factors are limited to a number of processors which is around 32, therefore, albeit simulations are able to scale up to more than 200 processes, the gain stops much early. Running multiple instances (jobs) is not a suitable strategy, since available memory is also a limiting factor and it leads rapidly to saturation with similar results in exchanging the number of instances and the number of processes per instance. The premise was clearly not optimal and the results confirmed it, however it demonstrates that on *pro bono* HPC resources ALICE can simulate data with Run 3 software while this was not possible at all with former framework during Run 1-2.

On the infrastructural aspect, it has to be pointed out that the system administration at CINECA have been much welcoming in accommodating the specific requirements of HEP use cases. This has not to be taken as guaranteed on other HPC sites, where outbound connectivity, CVMFS and plug to Grid ecosystem very often is the real blocker of opportunistic projects aiming at exploiting resources not primarily dedicated to HEP jobs executions. Further work on this front is expected to be necessary in trying to implement similar projects on different sites, depending on the level of flexibility the target HPC centre offers.

Appendix B

Alidock: isolated and consistent environment for ALICE software

The Alidock[23] project has been carried out during the development of the main project presented in this thesis. It is distributed as a Python package and install a command line tool that serves as an entry point into a Linux shell. Such a shell is executed inside a custom environment, completely isolated from the host Operating System (OS). The environment is always the same, created to be homogeneous across different hosting OSs.

The development of such a tool had as a scope to simplify the daily life of the ALICE software users (developers and physicists) in two different scenarios:

- to have a configurable, clean and disposable working environment that could be provided *on-demand* (libraries, software, OS) for working with ALICE software both from Run 2 and Run 3. The goal was to deliver the possibility to quickly have a working environment to perform some tests and do some development;
- to be a simple tool to just quickly give to users a supported and tested working environment where to run ALICE software in events like the analysis tutorials[24]. These events are more focused on the ALICE newcomers, usually students, with their own computers, that gather at CERN to attend short intensive periods of initialisation with the goal to acquire some knowledge about the tools, techniques and technologies used by the ALICE experiment in different fields (data analysis, software development, etc.).

B.1 Virtualisation of Linux environments

The enabling technology adopted as a core of Alidock is the *virtualisation* by using Linux containers. Virtualisation in computing science consists in a set of techniques aiming at abstracting both the software and hardware resources (depending on the depth of the virtualisation) used by a computer. With the virtualisation is possible to emulate up

to a whole *virtual* machine that runs into a physical one. The concept of virtualisation is included in a broad semantic area, from the OS only, to whole virtually simulated computing hardware (disks, network interfaces, virtual CPUs and memory, etc.). The advantages in using virtualisation are essentially two:

- *Resource emulation*: a virtual machine is a set of virtualised hardware resources used to run an operating system as it were hosted on a physical machine. To virtualise the hardware means to emulate, at a software level, the behaviour of real pieces of hardware. Through the hardware virtualisation it is possible to mimic a CPU architecture that is substantially different from the one actually running in the underlying physical hardware.
- *Resource partition*: a single piece of hardware resource can be logically limited or divided into smaller pieces of virtual resources of the same kind. A virtual machine or a Linux container can be configured to show a subset of the capabilities available on the physical host, like the number of cores for a CPU or a subset of the total memory or persistent disk.

B.1.1 Linux containers to deliver consistent environments

A Linux container is a lightweight virtualisation solution consisting of an OS-level method for running single or multiple isolated Linux systems on a single control host. It is a technology based on two powerful features of the Linux kernel: *kernel namespaces* and *cgroups*. The former allows the underlying operating system to abstract a computing resource and to expose the availability of the whole resource or fraction of it to processes. The processes in the same namespace share the same resources, while processes in different namespaces are considered as two separate instances that may not have access to the same targets (filesystems, devices, network interfaces). The latter is responsible for setting configurable limits to the resources given to a process, for instance the maximum CPU time can be spent on that task. By combining the two, the kernel is able to launch lightweight virtualised instances with custom configurations, OS flavour, libraries, etc. Each Linux container shares the underlying kernel with all the other system processes and other containers on the host and but it is not aware of the surrounding environment if not involved in its execution scope. The underlying physical hardware is not emulated but it can be arbitrarily exposed to the container through the kernel drivers. In figure B.1 a schema summarising the logical hierarchy of components in a system that runs multiple Linux containers is reported.

Containers are very useful, because they enable the possibility to deploy workloads, on same machine, that run on different operating systems with custom libraries and software dependencies. In principle they are completely separated from the one hosted on the system and to other running containers. This feature is instrumental for the Alidock tool, since its main purpose is to provide a working environment consistent with software requirements and dependencies needed by the installed ALICE software.

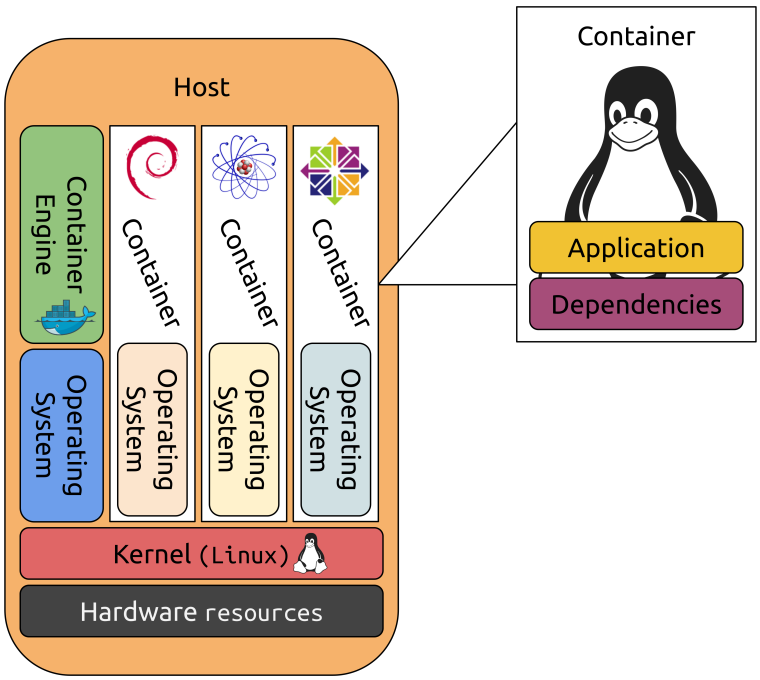


Figure B.1: Scheme of a container-based virtualisation architecture on the Linux operating system. The hardware and the kernel are not emulated in the environment.

B.1.2 The Docker container engine

Nowadays, several tools are available to streamline the deployment of Linux containers on local machines and on clusters. In the case of Alidock, the container engine adopted is Docker[61]: this is a service that can be installed both on local machines and servers into a cluster for large scale deployments. Docker is equipped with a complete Application Programming Interface, with implementations available in different programming languages. It has versions for Linux, macOS®, and Windows® and allows for deploying Linux containers on any of these platforms. It requires to run with administration rights to directly interact with the kernel[i] and it is able to effectively run any type of custom container. Docker has also the possibility to expose host resources such as network interfaces, graphic and audio cards everything in Linux is seen as a kernel device, with a high level of granularity. The architecture of Docker is based on a daemon that runs in background and through its API communicates with kernel to deploy containers.

[i]At present, every tool available to use Linux containers that exploits namespaces and cgroups requires super-permissions, Singularity[57], mentioned in Appendix A, included.

B.2 Alidock software structure

The Alidock executable is contained into a Python 3 package, available on an online repository and published on the official PyPi[70] repository. It provides a user-friendly tool that wraps some of the more useful Docker APIs, to launch containers and possibly to expose host directories and other resources present on the hosting machine. The rationale behind the tool is to provide an extremely simplified and handy command line interface (CLI) for users who are therefore quickly prompted to a custom Linux shell, ready to be employed for the ALICE software usage and development.

The command line wrapper instructs the Docker daemon directly for downloading, updating and accessing the container which is optionally kept running upon any session detachment. Alidock has a certain level of configuration that can be managed through a configuration file to override reasonable defaults values for most common use cases. The running container is completely decoupled from the Python script and entirely managed via the daemon. Its corresponding image name, that is used to specify the "flavour" of container to be run, can be overridden. This specific feature makes the tool more flexible, to accommodate custom and not common use cases like the development of the vertexer finder on GPU, that requires some additional dependencies in terms of libraries and kernel devices to be accessible.

Special attention has been dedicated to the installation, update and usage details, keeping those steps as easy as possible. Installation on **NIX** systems –i.e. all non-Windows supported OSs, where it is installed directly via the `pip` package installer– is done via a command that automatically downloads the installation script. The script checks for the correct and working installation of Docker daemon and reports possible issues providi. To each call from the CLI, with a certain grace-time, the software checks for updates of its internal scripts, while Docker provides automatic updates for the running container. Older installations are then able to auto-update themselves in a transparent and non invasive way. Alidock does not require root permissions and it is installed in user domain. It labels running containers with a unique identification number and this ensures the absence of conflicts even in case of usage from multiple users on the same machine.

B.2.1 Alidock container

The default container that used with any Alidock installation is based on an official version supported by ALICE. The image, used as a base for the Alidock environment construction, is the same used by the ALICE continuous integration of software and testing (better described in section 3.3.1). At the beginning of each test or build, a new container is launched in a disposable way, aiming at reproducing a clean installation of the tested software on any supported architecture. Then, if tests are passed, the software is validated, guaranteeing that on that specific system it can be installed. This ensure there will be no issues in the installation if Alidock as long as it uses that container image. The ALICE base container is created starting from a plain Cern CentOS 7 (CC7)[1] Docker image where there have been installed all the required software dependencies to successfully build the ALICE software. On top of that base, a small set of utilities needed for the correct functioning of the Alidock session is installed.

The ALICE software is the installed using the official installer called *aliBuild*[10]. Such

a tool is designed to manage the compilation and installation for each piece of software, usually used by the ALICE collaboration in different contexts. It also supports multiple operating systems. It has an extremely valuable feature concerning the possibility to install pre-compiled latest versions of software updated on daily basis. The software release cycle in ALICE foresees the release of pre-compiled packages for CC7. AliBuild allows the user to install pre-compiled packages for all the software that with respect the specific installation is noted as a prerequisite, by just checking that the underlying operating system is CC7. This possibility accelerates a lot the installation time because relatively small fraction of the code needs to be compiled that usually takes more time than downloading the pre-compiled package. Such a feature is very efficient to promptly obtain a working installation of the software.

B.2.2 Alidock: usage and rationale

In virtualisation, containers are most of the time used as ephemeral boxes to run specific services: application inside containers are very likely to be *stateless*, meaning that their inner state needs to be stored in a persisting place upon restarts and unexpected halts of the container system. This feature is instrumental for any high-availability service, which is able to recover from a failure in a negligible amount of time by re-deploying the container thus ensuring the service is always available. In Alidock as well, the container serves as a wrapper around a secure shell (SSH) server, a program that enables remote connection to a computer, in this case virtualised by the container. Persistent directories from the host are exposed to the container. The one included by default is used as "home" directory. By installing and using software there, it will survive any container restart, deletion etc.

At its first launch Alidock tells Docker to launch a micro-service represented by the SSH server, and to silently wait for connections. Then, thanks to virtual the networks provided by Docker, the Alidock CLI is able to access the container instance via SSH. The idea is to login into the container same as it was a remote session into a remote machine. In this way a dedicated session, based on CC7 and where the ALICE software dependencies are available, is ready to be used like it was on a different computer. The detachment from the SSH session leaves the container keeping running in background. The login on the container is done automatically by using key pairs that are generated at the first launch[ii] and their existence is especially needed to support the multi-user case, where the access to other users' containers should not be allowed. The usage of a remote session as method to access the container benefits from all the technologies available such as terminal multiplexing and remote X11 forward. The users can therefore access graphical softwares requiring video output a Graphical User Interface (GUI). There is a set of parameters that can be passed to the CLI, even though its default configuration has been set to suffice most of the use cases. Command line parameters allows the user to login as super user inside the container, providing administration permissions *inside* the container. This permissions are not effective in the outside environment, not accessible bu the container. This feature, as anticipated, is granted by the isolation that comes with kernel namespaces. It is possible

[ii] private key is kept on the "local" filesystem whereas the public is shared by putting it with the directory used as home.

to attach different piece of filesystem, if necessary, not included in the "home" directory. Such operation needs to be performed at container startup, and it is not possible to do that at runtime. That operation is easy, since ephemeral containers are pretty good in being relaunched very quickly with a different configuration.

B.2.3 Alidock as a development environment for GPU software

Concerning the design of the Alidock tool, some features have been added specifically to support some requirements in the development of the ITS primary vertex algorithm on GPU. By itself, Alidock is suitable to deploy at the same time different configurations and the resources to be exposed to each instance can be set by such a configuration. Graphic cards are interesting resources to be added to the container runtime environment and it has been useful to provide the possibility to logically separate two different GPUs installed on the same development workstation 6.1.

Docker supports both Nvidia and AMD platforms, the GPUs to be accessed from the container, with high level of integration[iii]. Implementation details are different for different kind of GPUs, since they interact with their runtime libraries in different ways. Two different strategies have been developed to interface Nvidia and AMD GPUs to the Alidock runtime, in a streamlined mode automatically usable from the CLI. It is therefore possible to expose arbitrarily either one or both GPUs inside the Alidock session by simply passing one or two parameters to the command.

Containers used for this specific case are different from the official supported ones, since they have been built with the specific runtime libraries to use GPUs. Thanks to the possibility to use more containers at the same time with different "flavours", it is possible to run tests on different driver setups and library versions. This can be done without applying any change on the environment outside, which saves a lot of time in case of some possibly *disruptive* test that breaks the working environment on the host.

To give an example, upon updates from the side of the ROCm[14] and CUDA[67] framework, it has been possible to test it on safe disposable containers with few commands. In the worst cases Alidock has been a safe tool to quickly fallback to a working setup after some major change in the mutable structure of the O^2 at its early stages. Other use cases are more related to have a quick deployment of the ALICE software on a new machine without having the time to compile all the stack. The possibility to use the pre-cached packages has been largely used in many cases and by many users outside the scope of this work.

B.2.4 A tool for quick provisioning of software playgrounds

The main field of usage of the tool has been at the "ALICE Analysis Tutorial" events organised on yearly basis at CERN. There, one class of 20-30 newcomers in the ALICE experiment has an intense week of lectures on many topics strictly related to the techniques and the software tools used in the experiment to perform physics analyses. Typically, one

[iii] containers must have hardware libraries installed and some other adjustments, other than being able to access a "device" that is used as an abstract interface by Linux for GPGPU runtime.

entire lecture is dedicated to the software installation using the tools officially supported by the experiment such as the aforementioned aliBuild installer. The users bring their own laptop and the goal is to install the software that later on will be used to practice on some tutorials about the actual analysis. Although the aliBuild tool is very effective in supporting many platforms, it has been noticed that a lot of time during the lesson was spent to fix issues due to the variegated plethora of operating systems brought in and problems related to mismatching versions of OSs and dependencies. On top of that the different capabilities of laptops caused different times spent in compiling the code and in rare cases it required extra time.

To overcome all these issues it has been chosen to suggest, in general, to use Alidock as tool to bring an homogeneous and consistent environment to the installation. By using Alidock each computer was running a CC7 container, also taking advantage from the cached builds and they could relatively quickly install the required software.

B.3 Conclusion

During the development of the main work presented in this thesis, this tool has been invented and implemented with the goal to give to the collaboration an easy tool to solve a fraction the issues possibly encountered during the software installation. At the same time the rationale has been to keep its default behaviour as simple as possible, to reduce the burden of using it as an interface to the official installation procedure. The Alidock tool is able to provide an interactive session into a "containerised" environment, granting the access through a secure shell connection. It is compatible with most of the current operating systems thanks to the easy virtualisation brought by Docker. It has been useful to test different setups and configuration both of logical hardware and libraries during the development of the primary vertex finder on GPU. It has been successfully used up to the last edition of the ALICE Analysis Tutorial as a tool to quickly provide a working setup on ALICE newcomers' laptops and to build the software stack used in analysis-oriented lectures. Its adoption is not limited to the usage during the offline tutorials because of its simple usage and also thanks to the possibility to rapidly deliver working setups usable as "testbeds" for various applications. Some people use Alidock as a regular tool to wrap ALICE software in their every-day workflow. The `alidock` command has been typed more than 25K times as it can be seen on the main page of its source code [23]. Every time an official Alidock container is pulled, the Docker repository anonymously increases a counter, therefore the metric is accessible from the official repository [52].

Acknowledgements

My professional thanks go to my supervisors: Prof. Danilo Demarchi and Prof.ssa Stefania Bufalino for the trust and autonomy granted me, for their availability and support.

I would also like to thank the whole group of ALICE Torino, whose stimulating, inclusive and collaborative environment is the main reason why I decided to continue with my studies. In particular I would like to thank Prof. Massimo Masera for his extraordinary support and wise advice.

Special thanks also go to Maximiliano Puccio and Dario Berzano, for the continuous and consistent contribution of notions, food for thought and brilliant ideas, but also of points of reference and support even outside the workplace. Then the whole group of (post-) doctoral students, in particular: Luca Barioglio, Fabrizio Grosa and Ivan Ravasenga, not only competent, brilliant and capable colleagues, but also friends.

My personal thanks go to all members of my family, in all its "variations on the theme". Finally to Valentina, pillar of my few certainties: thanks.

Bibliography

[1] CERN Centos 7. *Linux at CERN*. https://linux.web.cern.ch/linux/centos7. [Online; accessed 5-March-2020]. 2020.

[2] Georges Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Phys. Lett.* B716 (2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[3] Roel Aaij et al. "Observation of $J/\psi p$ Resonances Consistent with Pentaquark States in $\Lambda_b^0 \to J/\psi K^- p$ Decays". In: *Phys. Rev. Lett.* 115 (2015), p. 072001. DOI: 10.1103/PhysRevLett.115.072001. arXiv: 1507.03414 [hep-ex].

[4] U. Abeysekara et al. "ALICE EMCal Physics Performance Report". In: (2010). arXiv: 1008.0413 [physics.ins-det].

[5] Shreyasi Acharya et al. "Production of charged pions, kaons and (anti-)protons in Pb-Pb and inelastic pp collisions at $\sqrt{s_{\rm NN}} = 5.02$ TeV". In: (2019). arXiv: 1910.07678 [nucl-ex].

[6] Jaroslav Adam et al. "Centrality dependence of the charged-particle multiplicity density at midrapidity in Pb-Pb collisions at $\sqrt{s_{\rm NN}} = 5.02$ TeV". In: *Phys. Rev. Lett.* 116.22 (2016), p. 222302. DOI: 10.1103/PhysRevLett.116.222302. arXiv: 1512.06104 [nucl-ex].

[7] D. Adamova and J. Horky. "An optimization of the ALICE XRootD storage cluster at the Tier-2 site in Czech Republic". In: *J. Phys. Conf. Ser.* 396 (2012), p. 042001. DOI: 10.1088/1742-6596/396/4/042001.

[8] B. Adeva et al. "The Construction of the L3 Experiment". In: *Nucl. Instrum. Meth.* A289 (1990), pp. 35–102. DOI: 10.1016/0168-9002(90)90250-A.

[9] Gianluca Aglieri Rinella. "The ALPIDE pixel sensor chip for the upgrade of the ALICE Inner Tracking System". In: *Nucl. Instrum. Meth.* A845 (2017), pp. 583–587. DOI: 10.1016/j.nima.2016.05.016.

[10] AliSW. *ALIBUILD*. https://alisw.github.io/alibuild/. [Online; accessed 5-March-2020]. 2020.

[11] J. Allen et al. "ALICE DCal: An Addendum to the EMCal Technical Design Report Di-Jet and Hadron-Jet correlation measurements in ALICE". In: (2010).

[12] J. Allison et al. "Recent developments in Geant4". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 835 (2016), pp. 186–225. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2016.06.125. URL: http://www.sciencedirect.com/science/article/pii/S0168900216306957.

[13] AMD. *ROCm profiler*. https://rocm-documentation.readthedocs.io/en/latest/Programming_Guides/hip_profiling.html. [Online; accessed 26-March-2020]. 2020.

[14] AMD. *ROCm, a New Era in Open GPU Computing*. https://rocm.github.io/. [Online; accessed 27-February-2020]. 2020.

[15] Gene M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1967, pp. 483–485. ISBN: 9781450378956. DOI: 10.1145/1465482.1465560. URL: https://doi.org/10.1145/1465482.1465560.

[16] B Abelev et al and. "Technical Design Report for the Upgrade of the ALICE Inner Tracking System". In: *Journal of Physics G: Nuclear and Particle Physics* 41.8 (July 2014), p. 087002. DOI: 10.1088/0954-3899/41/8/087002. URL: https://doi.org/10.1088%2F0954-3899%2F41%2F8%2F087002.

[17] B Abelev et al and. "Upgrade of the ALICE Experiment: Letter Of Intent". In: *Journal of Physics G: Nuclear and Particle Physics* 41.8 (July 2014), p. 087001. DOI: 10.1088/0954-3899/41/8/087001. URL: https://doi.org/10.1088%2F0954-3899%2F41%2F8%2F087001.

[18] I. Antcheva et al. "ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization". In: *Computer Physics Communications* 180.12 (2009). 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures, pp. 2499–2512. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2009.08.005. URL: http://www.sciencedirect.com/science/article/pii/S0010465509002550.

[19] P. Antonioli, A. Kluge, and W. Riegler. "Upgrade of the ALICE Readout & Trigger System". In: ().

[20] P Antonioli, A Kluge, and W Riegler. *Upgrade of the ALICE Readout and Trigger System*. Tech. rep. CERN-LHCC-2013-019. ALICE-TDR-015. Sept. 2013. URL: https://cds.cern.ch/record/1603472.

[21] Giuseppe Battistoni et al. "Overview of the FLUKA code". In: *Annals of Nuclear Energy* 82 (2015). Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms, pp. 10–18. ISSN: 0306-4549. DOI: https://doi.org/10.1016/j.anucene.2014.11.007. URL: http://www.sciencedirect.com/science/article/pii/S0306454914005878.

[22] Laurent Baulieu et al., eds. *Lecture Notes, Les Houches Summer School, 97th Session: Theoretical Physics to Face the Challenge of LHC*. Vol. 97. Oxford Univ. Press. Oxford: Oxford Univ. Press, 2015. ISBN: 9780198727965. DOI: 10.1093/acprof:oso/9780198727965.001.0001. URL: http://dx.doi.org/10.1093/acprof:oso/9780198727965.001.0001.

[23] D. Berzano and M. Concas. *alidock*. https://github.com/alidock/alidock. [Online; accessed 4-March-2020]. 2020.

[24] Dario Berzano et al. "Software training for the next generation of physicists: joint experience of LHCb and ALICE". In: *EPJ Web Conf.* 214 (2019), p. 05044. DOI: 10.1051/epjconf/201921405044.

[25] I. Bird et al. "Update of the Computing Models of the WLCG and the LHC Experiments". In: (2014).

[26] M. Bondila et al. "ALICE T0 detector". In: *IEEE Trans. Nucl. Sci.* 52 (2005), pp. 1705–1711. DOI: 10.1109/TNS.2005.856900.

[27] boost. *boost*. https://boost.org/. [Online; accessed 26-February-2020]. 2020.

[28] Robert S. Boyer and J. Strother Moore. "MJRTY—A Fast Majority Vote Algorithm". In: *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Ed. by Robert S. Boyer. Dordrecht: Springer Netherlands, 1991, pp. 105–117. ISBN: 978-94-011-3488-0. DOI: 10.1007/978-94-011-3488-0_5. URL: https://doi.org/10.1007/978-94-011-3488-0_5.

[29] Peter Braun-Munzinger et al. "Properties of hot and dense matter from relativistic heavy ion collisions". In: *Phys. Rept.* 621 (2016), pp. 76–126. DOI: 10.1016/j.physrep.2015.12.003. arXiv: 1510.00442 [nucl-th].

[30] R Brun et al. *GEANT 3: user's guide Geant 3.10, Geant 3.11; rev. version*. Geneva: CERN, 1987. URL: https://cds.cern.ch/record/1119728.

[31] P. Buncic, A. J. Peters, and P. Saiz. "The AliEn system, status and perspectives". In: *eConf* C0303241 (2003), MOAT004. arXiv: cs/0306067 [cs-dc].

[32] P Buncic, M Krzewicki, and P Vande Vyvre. *Technical Design Report for the Upgrade of the Online-Offline Computing System*. Tech. rep. CERN-LHCC-2015-006. ALICE-TDR-019. Apr. 2015. URL: https://cds.cern.ch/record/2011297.

[33] Squid Cache. *Squid*. https://squid-cache.org/. [Online; accessed 3-March-2020]. 2020.

[34] Serguei Chatrchyan et al. "Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC". In: *Phys. Lett.* B716 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021. arXiv: 1207.7235 [hep-ex].

[35] ALICE Collaboration. *O2 software project for the ALICE experiment at CERN*. https://github.com/AliceO2Group/AliceO2. [Online; accessed 26-March-2020]. 2020.

[36] Matteo Concas et al. "Plancton: an opportunistic distributed computing project based on Docker containers". In: *J. Phys. Conf. Ser.* 898.9 (2017), p. 092049. DOI: 10.1088/1742-6596/898/9/092049.

[37] P Cortese et al. “ALICE technical design report on forward detectors: FMD, T0 and V0”. In: (2004).

[38] Nvidia CUB. *Device scan reference*. https://nvlabs.github.io/cub/structcub_1_1_device_scan.html. [Online; accessed 26-March-2020]. 2020.

[39] G. Dellacasa et al. “ALICE technical design report of the zero degree calorimeter (ZDC)”. In: (1999).

[40] Advanced Micro Devices. *AMD*. https://amd.com. [Online; accessed 26-March-2020]. 2020.

[41] Advanced Micro Devices. *RDNA Architecture*. https://www.amd.com/system/files/documents/rdna-whitepaper.pdf. [Online; accessed 26-March-2020]. 2020.

[42] Eulisse, Giulio et al. “Evolution of the ALICE Software Framework for Run 3”. In: *EPJ Web Conf.* 214 (2019), p. 05010. DOI: 10.1051/epjconf/201921405010. URL: https://doi.org/10.1051/epjconf/201921405010.

[43] Lyndon Evans and Philip Bryant. “LHC Machine”. In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08001–S08001. DOI: 10.1088/1748-0221/3/08/s08001. URL: https://doi.org/10.1088%2F1748-0221%2F3%2F08%2Fs08001.

[44] Davide Falchieri. “DRM2: the readout board for the ALICE TOF upgrade”. In: *PoS* TWEPP-17 (2018), p. 081. DOI: 10.22323/1.313.0081.

[45] M. J. Flynn. “Some Computer Organizations and Their Effectiveness”. In: *IEEE Transactions on Computers* C-21.9 (1972), pp. 948–960.

[46] *git*. https://git-scm.com/docs. [Online; accessed 26-March-2020]. 2020.

[47] H. H. Gutbrod et al. “An international accelerator facility for beams of ions and anti-protons. Conceptual design report”. In: (2001).

[48] Matthias Hanauske et al. “Neutron Star Mergers: Probing the EoS of Hot, Dense Matter by Gravitational Waves”. In: *Proceedings, The Modern Physics of Compact Stars and Relativistic Gravity 2017 (MPCS2017): Yerevan, Armenia, September 18-22, 2017*. Vol. 2. 1. 2019, pp. 44–56. DOI: 10.3390/particles2010004.

[49] HashiCorp. *Packer*. https://packer.io/. [Online; accessed 24-February-2020]. 2020.

[50] Byungsik Hong. “Nuclear Matter Under Extreme Conditions: from Quark-Gluon Plasma to Neutron Stars”. In: *J. Korean Phys. Soc.* 72.12 (2018), pp. 1515–1522. DOI: 10.3938/jkps.72.1515.

[51] I. Hrivnacova et al. “The Virtual Monte Carlo”. In: *CoRR* cs.SE/0306005 (2003). URL: http://arxiv.org/abs/cs/0306005.

[52] Docker Hub. *Alidock official docker container*. https://hub.docker.com/r/alisw/alidock. [Online; accessed 7-April-2020]. 2020.

[53] “IEEE Standard for Floating-Point Arithmetic”. In: *IEEE Std. 754-2008* (2008). DOI: 10.1109/IEEESTD.2008.4610935.

[54] ISO. *ISO/IEC 14882:2011: Programming languages — C++*. Sept. 2011, p. 732. URL: https://isocpp.org/std/the-standard.

[55] Kitware. *CMake*. https://https://cmake.org/. [Online; accessed 24-February-2020]. 2020.

[56] Mikolaj Krzewicki and Volker Lindenstruth. "ALICE HLT Run 2 performance overview". In: *J. Phys. Conf. Ser.* 898.3 (2017), p. 032056. DOI: 10.1088/1742-6596/898/3/032056.

[57] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. "Singularity: Scientific containers for mobility of compute". In: *PLoS ONE* 12(5) (2017). DOI: 10.1371/journal.pone.0177459.

[58] Chris Lattner and Vikram Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation". In: *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*. CGO '04. Palo Alto, California: IEEE Computer Society, 2004, p. 75. ISBN: 0769521029. DOI: 10.5555/977395.977673.

[59] Andreas Mathis. "From gated to continuous readout - the GEM upgrade of the ALICE TPC". In: *PoS* MPGD2017 (2019), p. 055. DOI: 10.22323/1.322.0055. arXiv: 1803.04140 [physics.ins-det].

[60] Michael D. McCool, Arch D. Robison, and James Reinders. *Structured parallel programming patterns for efficient computation*. 2012.

[61] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment". In: *Linux J.* 2014.239 (Mar. 2014). ISSN: 1075-3583.

[62] Levente Molnar. "The ALICE HMPID detector ready for collisions at the LHC". In: *Nucl. Instrum. Meth.* A595 (2008), pp. 27–30. DOI: 10.1016/j.nima.2008.07.088. arXiv: 0805.0737 [nucl-ex].

[63] Gordon Moore. "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff". In: *Solid-State Circuits Newsletter, IEEE* 11 (Oct. 2006), pp. 33–35. DOI: 10.1109/N-SSC.2006.4785860.

[64] Berndt Muller and James L. Nagle. "Results from the relativistic heavy ion collider". In: *Ann. Rev. Nucl. Part. Sci.* 56 (2006), pp. 93–135. DOI: 10.1146/annurev.nucl.56.080805.140556. arXiv: nucl-th/0602029 [nucl-th].

[65] nanomsg. *nanomsg*. https://nanomsg.org/. [Online; accessed 26-February-2020]. 2020.

[66] Nvidia. *Nvidia*. https://nvidia.com. [Online; accessed 26-March-2020]. 2020.

[67] Nvidia. "Nvidia Cuda C Programming Guide". In: *Compare A Journal Of Comparative Education* (Jan. 2010).

[68] Nvidia. *Nvprof*. https://docs.nvidia.com/cuda/profiler-users-guide/index.html. [Online; accessed 26-March-2020]. 2020.

[69] A. A. Penzias and R. W. Wilson. "A Measurement of Excess Antenna Temperature at 4080 Mc/s." In: *Astrophysical Journal* 142 (July 1965), pp. 419–421. DOI: 10.1086/148307.

[70] PyPi. *Find, install and publish Python packages with the Python Package Index*. https://pypi.org. [Online; accessed 5-March-2020]. 2020.

[71] SCAI. *MARCONI, the new Tier-0 system*. http://www.hpc.cineca.it/hardware/marconi. [Online; accessed 27-February-2020]. 2020.

[72] Jurgen Schukraft and Reinhard Stock. "Toward the Limits of Matter: Ultra-relativistic nuclear collisions at CERN". In: *Adv. Ser. Direct. High Energy Phys.* 23 (2015), pp. 61–87. DOI: 10.1142/9789814644150_0003. arXiv: 1505.06853 [nucl-ex].

[73] Clive Seguna et al. "Proposal for a new ALICE CPV-HMPID front-end electronics topology". In: *Proceedings, 13th Conference on PhD Research in Microelectronics and Electronics (PRIME 2017): Taormina, Italy, June 12-15, 2017*. 2017, pp. 173–176. DOI: 10.1109/PRIME.2017.7974135.

[74] Sabyasachi Siddhanta. "Muon physics at forward rapidity with the ALICE detector upgrade". In: *Nucl. Phys.* A982 (2019), pp. 947–950. DOI: 10.1016/j.nuclphysa.2018.10.034.

[75] Torbjörn Sjöstrand et al. "An Introduction to PYTHIA 8.2". In: *Comput. Phys. Commun.* 191 (2015), pp. 159–177. DOI: 10.1016/j.cpc.2015.01.024. arXiv: 1410.3012 [hep-ph].

[76] J. E. Stone, D. Gohara, and G. Shi. "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems". In: *Computing in Science Engineering* 12.3 (May 2010), pp. 66–73. ISSN: 1558-366X. DOI: 10.1109/MCSE.2010.69.

[77] Reisaburo Tanaka. "LEP accelerator and detectors". In: *Proceedings, LEP Symposium 2001 : Beyond the Electroweak Scale: Tokyo, Japan, November 5-6, 2001*. 2001. URL: http://tkyice.icepp.s.u-tokyo.ac.jp/info/lepsymp/proceedings/reisaburo.pdf.

[78] Todd Tannenbaum et al. "Condor – A Distributed Job Scheduler". In: *Beowulf Cluster Computing with Linux*. Ed. by Thomas Sterling. MIT Press, Oct. 2001.

[79] M. Al-Turany et al. "ALFA: The new ALICE-FAIR software framework". In: *Journal of Physics Conference Series*. Vol. 664. Journal of Physics Conference Series. Dec. 2015, p. 072001. DOI: 10.1088/1742-6596/664/7/072001.

[80] M Al-Turany et al. "Extending the FairRoot framework to allow for simulation and reconstruction of free streaming data". In: *Journal of Physics: Conference Series* 513 (June 2014), p. 022001. DOI: 10.1088/1742-6596/513/2/022001.

[81] M Al-Turany et al. "The FairRoot framework". In: *Journal of Physics: Conference Series* 396.2 (Dec. 2012), p. 022001. DOI: 10.1088/1742-6596/396/2/022001. URL: https://doi.org/10.1088%2F1742-6596%2F396%2F2%2F022001.

[82] Antonio Uras and for the ALICE MFT Working Group. "Muon Physics in ALICE: The MFT Upgrade Project". In: *J. Phys. Conf. Ser.* 446 (2013), p. 012054. DOI: 10.1088/1742-6596/446/1/012054. arXiv: 1212.6517 [hep-ex].

[83] Abraham Villatoro Tello. "AD, the ALICE diffractive detector". In: *AIP Conf. Proc.* 1819.1 (2017), p. 040020. DOI: 10.1063/1.4977150.

[84] Derek Weitzel et al. "Accessing data federations with CVMFS". In: *J. Phys. Conf. Ser.* 898.6 (2017), p. 062044. DOI: 10.1088/1742-6596/898/6/062044.

[85] Wenzel, Sandro. "A scalable and asynchronous detector simulation system based on ALFA". In: *EPJ Web Conf.* 214 (2019), p. 02029. DOI: 10.1051/epjconf/201921402029. URL: https://doi.org/10.1051/epjconf/201921402029.

[86] Nathan Whitehead and Alex Fit-Florea. "Precision and performance: floating point and IEEE 754 compliance for NVIDIA GPUs". In: *rn (A+ B)* 21 (Jan. 2011).

[87] Sandra Wienke et al. "OpenACC: First Experiences with Real-World Applications". In: *Proceedings of the 18th International Conference on Parallel Processing*. Euro-Par'12. Rhodes Island, Greece: Springer-Verlag, 2012, pp. 859–870. ISBN: 9783642328190. DOI: 10.1007/978-3-642-32820-6_85. URL: https://doi.org/10.1007/978-3-642-32820-6_85.

[88] Andy B. Yoo, Morris A. Jette, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. ISBN: 978-3-540-39727-4.

[89] ZeroMQ. *ZeroMQ*. https://zeromq.org/. [Online; accessed 26-February-2020]. 2020.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.