Dissertation zur Erlangung des akademischen Grades
Dr. rer. nat.

A Selection Framework for LHCb's Upgrade Trigger

Niklas Nolte
geboren in Hildesheim, 1994

2020

Lehrstuhl für Experimentelle Physik V
Fakultät Physik
Technische Universität Dortmund

Erstgutachter:	Prof. Dr. Johannes Albrecht
Zweitgutachter:	Dr. Johannes Erdmann
Abgabedatum:	17. Dezember 2020

Kurzfassung

Das LHCb Experiment am Large Hadron Collier am CERN wird momentan für die nächste Datennahme verändert und modernisiert. Die instantane Luminosität wird um einen Faktor fünf erhöht, damit mehr Daten in kürzerer Zeit aufgenommen werden können. Die erste Stufe der Datennahme, der Hardwaretrigger, wird entfernt. LHCb muss nun eine Kollisionsrate von 30 MHz in Echtzeit verarbeiten. In dieser Arbeit werden drei Projekte vorgestellt, die signifikant zu der Entwicklung eines schnellen und effizienten Triggersystems beitragen.

Der erste Beitrag ist ein Scheduling Algorithmus mit vernachlässigbarem Overhead in der neuen Trigger-Applikation. Der Algorithmus steuert das Multi-Threading des Systems und ist der erste Algorithmus in LHCb, der den technischen Spezifikationen des Systems genügt. Durch die Restriktion auf Inter-Event Parallelismus können die meisten teuren Entscheidungen schon vor der Laufzeit der Applikation getroffen werden.

Der zweite Beitrag besteht aus mehreren Algorithmen zur Filterung und Kombination von Teilchen in der Kollision. Diese Algorithmen sind bis zu mehreren Größenordnungen schneller als die aktuellen, etablierten Algorithmen. Der Einsatz der neuen Algorithmen in der zweiten Trigger-Phase (HLT2) ist ein wichtiger Schritt zur Vervollständigung eines Trigger-Systems, dass den erhöhten Anforderungen entspricht.

Das letzte Projekt beschäftigt sich mit der Bandbreite, mit der der Trigger Kollisionen abspeichert. Dazu wird die wichtigste Selektion im HLT2 betrachtet, der topologische Trigger. Dieser Trigger versucht, Zerfälle von beauty Hadronen inklusiv zu selektieren. Zuerst wird der Selektionsalgorithmus selber optimiert. In einem zweiten Schritt werden die Kollisionen, die der Selektion entsprechen, getrimmt. Irrelevante Information für die Analyse von beauty Hadronen in diesen Kollisionen werden entfernt. Damit kann die Bandbreite pro gespeicherter Kollision verringert werden.

Abstract

The LHCb experiment at the Large Hadron Collider at CERN is planning a major detector upgrade for the next data taking period. The instantaneous luminosity is increased by a factor of five to generate more data per time. The hardware trigger stage preceding detector readout is removed and LHCb now has to process 30 MHz of incoming events in real time. The work presented in this thesis shows three contributions towards a fast and efficient trigger system.

The first contribution is a multi-threaded scheduling algorithm to work with negligible overhead in the Upgrade regime. It is the first implementation for LHCb capable of realizing arbitrary control and data flow at the required speed. It restricts itself to inter-event concurrency to avoid costly runtime scheduling decisions.

The second contribution comprises several algorithms to perform particle combination, vertexing and selection steps in the second HLT stage. These algorithms show competitive performances which can speed up the selection stage of HLT2 by orders of magnitude. Employing these algorithms marks a significant step towards an HLT2 application fast enough for the increased event rate.

The third contribution concerns the most important and most costly HLT2 selection at LHCb, the Topological Trigger. It aims to inclusively capture many beauty-flavored decay chains. First, the algorithm for event selection is optimized. In a second step, output bandwidth is reduced significantly by employing a classification algorithm for selecting additional relevant information present in the event.

Contents

1 Introduction

In Geneva, Switzerland, the European Organization for Nuclear Research (CERN) operates the largest ever built machine: The Large Hadron Collider (LHC) [1]. This circular particle collider measures 26.7 km in circumference and collides protons and heavy ions at the TeV energy scale. At four distinct interaction points around the perimeter, these collisions of unprecedented energy are captured by four large detectors: ATLAS, CMS, ALICE and LHCb.

Until this day, these experiments have made countless measurements in fundamental particle physics, including but not limited to the discovery of new particles and precision measurements of particle properties and interactions. The most important discovery was achieved in 2012 by the ATLAS and CMS collaborations: The Higgs-Boson [2, 3]. It is the last found elemental particle predicted by the best tested fundamental physics theory, the Standard Model (SM) of particle physics [4, 5, 6, 7].

Although the SM yields incredibly precise predictions all over the high energy physics landscape, it fails to cover several observed phenomena of our universe, like the existence of dark matter [8] or the asymmetry of matter and antimatter [9, 10]. There is also no description of gravity included in the Standard Model. We must therefore conclude that the SM is, at best, an incomplete model of the interactions in fundamental physics.

With steadily increasing energies and collision rates, experiments at the LHC try to find hints for physics Beyond the Standard Model (BSM) in the vast amount of data they generate. Every measurement gives rise to new theories and discourages others. The LHCb detector, which is in the focus of this thesis, specializes in indirect searches for BSM physics in beauty and charm decay processes. Around 1000 engineers and physicists collaborate to operate this detector, to process the data and to analyze it.

As the LHC is currently in the second long shutdown LS 2, LHCb is undergoing a major detector upgrade to prepare for a fivefold increase in luminosity in the third Run of the LHC. Major changes include a new tracking system and a full

replacement of all readout electronics. Most importantly for this thesis, the LHCb trigger system is being reworked completely [11].
A trigger system acts as filter for the huge amount of incoming data, as we cannot afford to persist multiple terabytes per second of collision information. It reconstructs particles trajectories in different detector components and ultimately combines them to form decay and collision vertices. To find the interesting physics needle in the haystack of interactions that a high energy hadronic collision creates, one can impose requirements on particle trajectories and vertices.

Until the end of Run 2 in 2018, the first selection step was performed by a hardware trigger, L0, that filtered events with a rate of 1 MHz. The two subsequent High Level Trigger (HLT) software stages thus had to operate at a much lower rate than the nominal LHC collision rate.
From 2021 onwards, there will not be a hardware trigger stage in LHCb's data processing flow to achieve a higher amount of flexibility and efficiency. This poses significant challenges to the HLT processing farm, as it has to operate in real time with a collision rate of 40 million per second. LHCb is the first of the four big experiments at the LHC to take the leap towards a software-only trigger system. Together with the aforementioned fivefold increase of luminosity, the computational requirements on the HLT computing farm increase by about two orders of magnitude with respect to previous data taking periods. In contrast to these requirements, the allocated computing budget does not allow for a farm that is orders of magnitude more powerful than the one that operated in 2018.

The second big challenge, next to computational load, is the limited output bandwidth. The trigger needs to efficiently select physics signatures for all analyses that researchers are interested in. Inclusive selections over such a broad range of physics signatures results in large bandwidth outputs. Therefore, the solution is made up by about $O(10^3)$ exclusive selections built into the second HLT stage. The fivefold increase of luminosity implies a fivefold increase of interesting data to be collected per unit of time, but also a similarly increased level of noise. Selections need to be very pure to fit into the output bandwidth while not discarding too much interesting data. Significant efforts are being put into the optimization of both HLT reconstruction and selection stages to meet both timing and bandwidth requirements.

Some of these efforts are presented in this thesis. Chapter 2 introduces the SM and problems that LHCb tries to gain further insight into. Chapters 3 and 4 discuss the upgrade LHCb detector and the upgrade trigger with its challenges in detail.

Chapter 5 takes a step back from the context of high energy physics to introduce principles for high performance computing on CPU architectures. Starting from Chapter 6, specific optimizations proposed for operation in the upgrade trigger are discussed. Chapter 6 describes the first feature complete implementation of a new scheduling algorithm to efficiently execute the HLT control flow with thousands of algorithms in online production. Chapter 7 is concerned with the implementation of faster algorithms for performing physics selections and decay chain reconstruction. This work includes the introduction of a new data model for particles. Finally, Chapter 8 tackles the output bandwidth problem by scrutinizing the biggest bandwidth contributor, the Topological Beauty Trigger. It aims to inclusively select beauty decays based on kinematic and topological signatures only. With a subsequent classification algorithm, only essential data from the event is filtered to save bandwidth without sacrificing efficiency.

2 The Standard Model of Particle Physics

The Standard Model of particle physics (SM) describes interaction dynamics of elementary particles as a quantized field theory and is the best tested and verified particle physics theory to this day. It is capable of modelling three out of the four fundamental types of interactions, the electromagnetic, the strong and the weak force. Gravity remains unaccounted for.

The collection of elementary particles within the SM comprise 12 fermions of half integer spin and 5 bosons of integer spin. We will henceforth imply the existence of an antiparticle for each mentioned particle, exhibiting the same mass but inverted quantum numbers.

Six of the fermions are called quarks, which have third integer electric charge. There are up-type quarks (up, charm, top) which have a charge of $+2/3$ and down-type (down, strange, bottom) quarks of charge $-1/3$. Quarks also carry another quantum number called color.
The other six fermions are called leptons, which are all colorless. Three of them (electron, muon, tau) have an electric charge of -1. The other three are called neutrinos and they are electrically neutral. All fermionic elementary particles have a common spin of $1/2$.

The bosons act as mediator for the fundamental interactions. Photons are exchanged during electromagnetic interaction, which couples to the electric charge. Quarks and charged leptons can interact electromagnetically.
The $W^{\pm}$ and Z^0 bosons propagate the weak interaction, which couples to a quantum number called weak isospin. Only left-chiral particles and right-chiral antiparticles exhibit weak isospin. Chirality is an abstract concept of transformation behavior that has a visual analogy for massless particles: Left-chiral means that spin and momentum point in opposite directions, while for right-chiral particles they point to the same direction. This analogy cannot be made for massive particles, as their momentum depends on the observer frame. All particles can interact under the weak force, given "correct" chirality. Neutrinos, in fact, only interact weakly.

The strong force couples to color and is mediated by eight different gluon states, which also carry color themselves. There are three colors and three anti-colors. All colors together make, in analogy to the additive color system, a colorless (or white) state, and so does a color with its anti-color. Because only colorless systems are stable in nature, we do not find quarks by themselves, but in bound states called hadrons. The most common hadrons are $q\bar{q}$ states called Mesons, and qqq states called baryons, but in principle all integer linear combinations of $q\bar{q}$ and qqq are possible. $qq\bar{q}\bar{q}$ states called tetraquarks and $q\bar{q}qqq$ called pentaquarks have already been found in recent years [12, 13].
The Higgs boson, found in 2012 [2, 3], is the only spin 0 elementary particle. It originates the Higgs field, giving mass to all particles. A visual summary of elemental particles and the interactions they take part in can be seen in Figure 2.1.

While the SM gives very accurate predictions and has been tested successfully over decades, there are several phenomena besides gravity and general relativity that it fails to explain. The SM predict massless neutrinos, but neutrino oscillation implying a finite mass has been found around 2000 [14, 15, 16]. The Nobel Prize was awarded for these findings in 2015. The observable matter-antimatter asymmetry [9, 10] or the existence of dark matter [8] in our universe are only a few examples of other unexplainable phenomena in the SM framework. Therefore, we must conclude that the SM is an incomplete model of the universe and must be overhauled or at least extended.

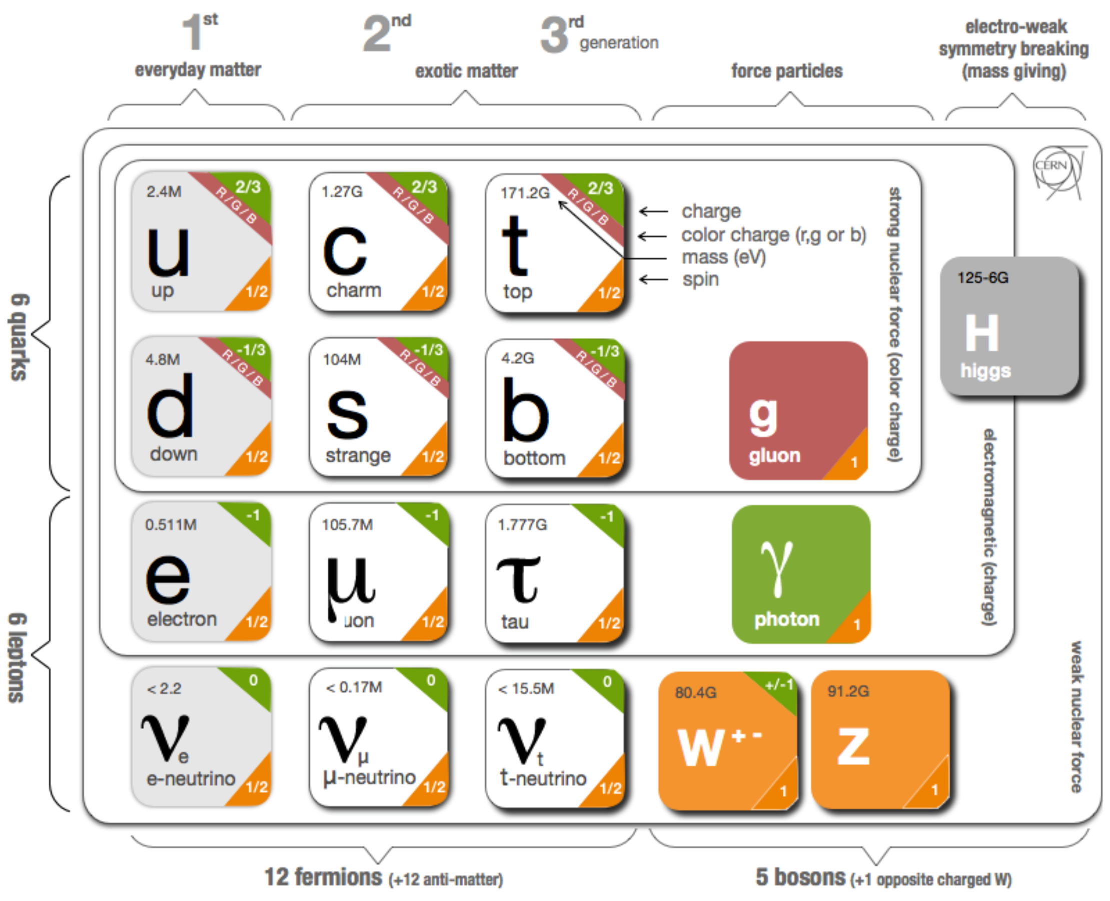

1st everyday matter
2nd exotic matter
3rd generation
force particles
electro-weak symmetry breaking (mass giving)
6 quarks
6 leptons
2.4M 2/3 R/G/B u up 1/2
1.27G 2/3 R/G/B c charm 1/2
171.2G 2/3 R/G/B t top 1/2
charge
color charge (r,g or b)
mass (eV)
spin
strong nuclear force (color charge)
125-6G H higgs
4.8M -1/3 R/G/B d down 1/2
104M -1/3 R/G/B s strange 1/2
4.2G -1/3 R/G/B b bottom 1/2
g gluon 1
0.511M -1 e electron 1/2
105.7M -1 μ μon 1/2
1.777G -1 τ tau 1/2
γ photon 1
electromagnetic (charge)
< 2.2 0 νe e-neutrino 1/2
< 0.17M 0 νμ μ-neutrino 1/2
< 15.5M 0 νt t-neutrino 1/2
80.4G +/-1 W+- 1
91.2G Z 1
weak nuclear force
12 fermions (+12 anti-matter)
5 bosons (+1 opposite charged W)

Figure 2.1: Elementary particles described within the Standard Model. Charge is shown in green in the upper right corner of each box, color in red and spin in yellow at the lower right. A particles mass in eV is given at the left upper part of the respective block. Modified from [17].

2.1 Obtaining a solution to unsolved problems with LHCb

With the help of collision experiments, physicists hope to find hints for physics beyond the Standard Model (BSM physics) that help to develop a model fitting these still unexplainable phenomena. The focus of this thesis, the LHCb experiment, is one of these collision experiments. By colliding protons with high energy and luminosity from the LHC, an unprecedented amount of beauty and charm decays are recorded with the LHCb experiment. The current LHCb dataset comprises about $O(10^{11})$ beauty and $O(10^{13})$ charm decays [18, 19, 20]. With data sets of such statistical power particle and decay properties can be measured very precisely. Measuring SM parameters helps in two ways. Deviations from predicted values of these properties aid the invention of extensions to the SM and new models. A measurement matching the prediction helps with the constraint and exclusion of other models.

Research within the LHCb collaboration focuses on different observable parameters. A prominent class of observables are those describing asymmetries under Charge- and Parity (CP) transformation. This is referred to as CP violation (CPV). CPV is predicted within the SM to a small degree, but additional sources of CPV are required to explain the matter-antimatter asymmetry [21]. Precise measurements of CP observables are thus very valuable in the pursuit of explanations to the problem. $B_s^0 \to \phi\phi$ and $B^0 \to D^+D^-$ are two examples of decays which are sensitive to parameters describing CP violation. CP measurements on these decays are currently limited by statistical power of the analyzed dataset [22, 23].

A second interesting field of research in LHCb are searches for violation of charged lepton flavor universality. The SM predicts a certain symmetry between lepton flavors. This implies that, up to differences induced by the lepton mass differences, branching fractions of decays involving charged leptons should not depend on the lepton flavor. However, LHCb published measurements that deviate from the SM with 2-3 standard deviations significance [24, 25, 26, 27]. The uncertainties on measurements of lepton flavor universality depend on the statistical powers of datasets recording decays like $B^0 \to K^{(*)0}e^+e^-$, $B^0 \to K^{(*)0}\mu^+\mu^-$, $B^0 \to D^{*-}\tau^+\nu$ and $B^0 \to D^{*-}\mu^+\nu$.

To decrease statistical uncertainties of all of these measurements, more data is needed. During the current shutdown of the LHC, the experiment is upgraded to be able to record new data at higher rates. The current LHCb dataset corresponds to an integrated luminosity of 9/fb. With the upgraded detector 50/fb shall be recorded within the next decade. The new data will be used to perform even more precise measurements and thus strengthen or negate recently found deviations and constrain theoretical models even further.

3 The LHCb Detector at the LHC

Hints for physics beyond the Standard Model might be found in various niches of phase space and it might be very hard to detect or very rare. The physicists' best guesses have traditionally been to inspect particle decays at higher and higher energies to uncover new states, new particles and potentially new physics. Over the course of many decades, they have therefore built more and more powerful particle colliders that operate at the highest possible energy. This chapter introduces the Large Hadron Collider and the LHCb experiment, located at the CERN research facility. Note that the LHCb detector will be described in its envisioned state after the current upgrade.

3.1 The Large Hadron Collider

The result of striving towards higher energies ultimately led to the creation of the Large Hadron Collider (LHC) [1], the largest ever built machine. It is a circular hadron collider operated by the European Organization for Nuclear Research (CERN) in Geneva, Switzerland. It measures 26.7 km in circumference, resides in a tunnel that is 50-175 m underground and accelerates protons and heavy ions to 13 TeV and 5 TeV per nucleon, respectively. With such large energies, the LHC is able to create a unique environment for particle decays. While these and even higher energies can also be observed from cosmic particles, a particle collider has the advantage of a controlled environment. 10^{11} protons per bunch and 40 million bunch crossing per second generated by two oppositely accelerated beams yield an instantaneous luminosity of $1 \times 10^{34}/(\mathrm{cm}^2\,\mathrm{s})$. This results in an unprecedented amount of statistics to be gathered by detectors around the LHC in a very stable environment.

In order to bring the particle beams to such high energies, smaller particle accelerators are employed as "pre-accelerators" that feed into the LHC. From Run 3 onwards, protons extracted from the ionization of hydrogen are first fed into the Linear Accelerator 4 (LINAC4) and then into the first circular collider, the BOOSTER with a circumference of 157 m, where they reach a maximum energy of 2 GeV. Before the second long shutdown, the first acceleration process was taken care of by LINAC2, which was retired after over 40 years of operation. Larger circular colliders follow in the acceleration chain, starting with the Proton Synchrotron (PS) with a circumference of 628 m and a maximum operating energy of 25 GeV. The last pre-accelerator that protons go through before eventually merging into the LHC is the Super Proton Synchrotron (SPS) with a 7 km circumference, where they are accelerated to about 450 GeV. The full chain of acceleration and eventual collision is displayed in Figure 3.1.

There are four major experiments at the four distinct interaction points around the LHC perimeter, ATLAS [29], CMS [30], ALICE [30] and LHCb [31]. ATLAS and CMS are torroidal general purpose detectors that cover spacial angles of almost 4π. They were the first experiments to observe the Higgs boson and currently focus on precision measurements of Higgs decays and top quark physics.
Researchers in the ALICE collaboration specialize on studying extremely hot quark-gluon plasma created by heavy ion collisions to better understand conditions during the big bang and in the very early universe.
The work presented in this thesis is about the fourth detector at the LHC: The Large Hadron Collider beauty (LHCb) experiment.

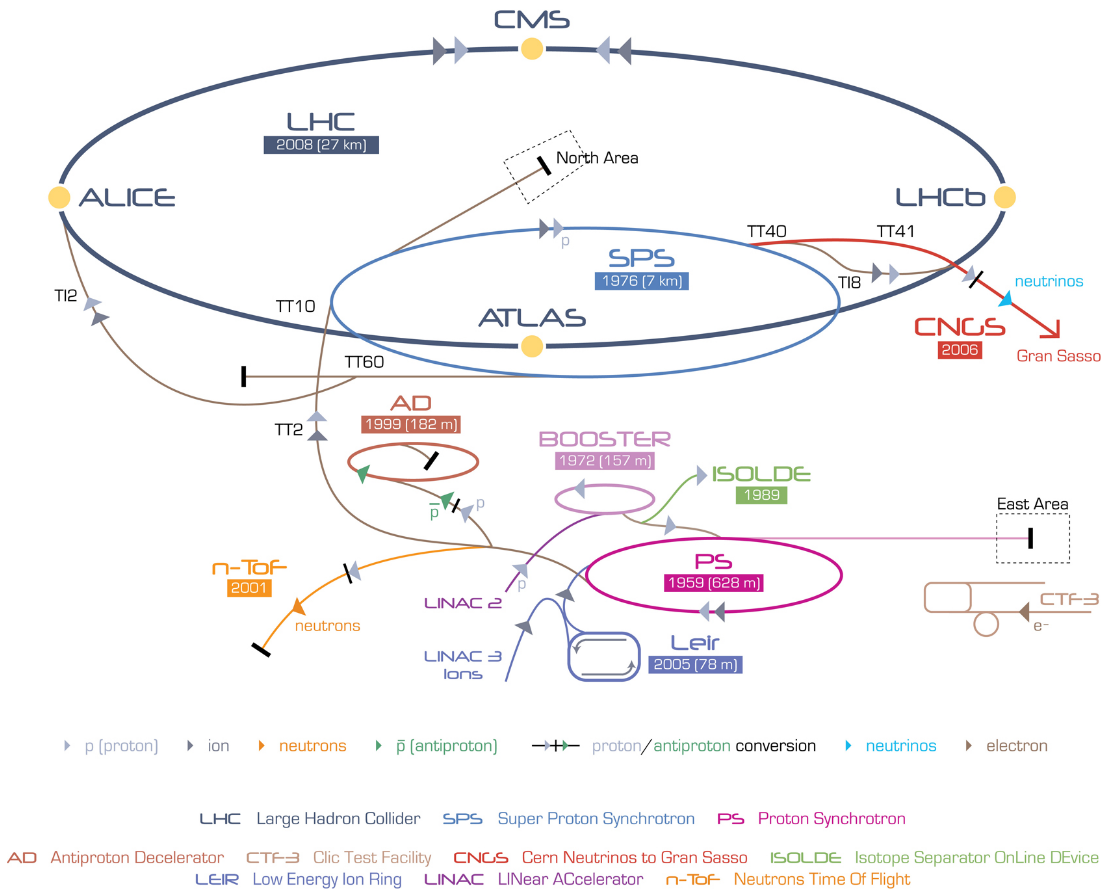


Figure 3.1: The collider complex and the experiments at CERN [28] before LS 2. Protons are extracted from a hydrogen bottle and are accelerated by the LINear ACcelerator 2 (LINAC2). They are fed into the BOOSTER, where they reach an energy of 1.4 GeV. The next stop is the Proton Synchrotron (PS), followed by the Super Proton Synchrotron (SPS), from where they are injected into the LHC with an energy of 450 GeV. The major changes for Run 3 include the replacement of LINAC2 by LINAC4 and the increase in the BOOSTER energy from 1.4 GeV to 2 GeV .

3.2 The LHCb detector

The LHCb detector weights about 5600 t and is over 20 m, long. It is a one arm forward spectrometer. Unlike the other experiments it does not cover the full spacial angle, but only an interval from 10 mrad to 250 mrad vertically and from 10 mrad to 300 mrad horizontally. In high energy physics, spacial angles are more often displayed in units of pseudorapidity η. The LHCb angular coverage corresponds to a range of $2 < \eta < 5$, with $\eta = -\log(\tan\theta/2)$. One usually finds asymmetric detectors where collisions happen asymmetrically as in fixed target experiments. However, LHCb specializes in precision measurements on beauty and charm decays to study CP violation and rare processes. The motivation for a forward spectrometer can be seen in the angular distribution of $b\bar{b}$ production in a LHC collision in Figure 3.2. Although LHCb covers only part of the 4π spacial angle, it is able to capture about 24 % of all $b\bar{b}$ pairs. The reason for this distribution is a high Lorentz boost in beam axis direction for light particles. Although both hadron bunches participating in the crossing have a symmetric momentum on average, the actual collision happens on parton level. Partons within the hadron can have a quite broad relative momentum distribution which results in a local momentum asymmetry and therefore a boost of all particles emerging from the collision. As the LHC deals with energies in the TeV scale and b quarks have a mass in the GeV region, relatively small parton momentum asymmetries cause a huge boost for the $b\bar{b}$ pair of interest.

The detector started to take data in 2010. It remained mostly unchanged until the finalization of Run 2 of the LHC in 2018. During that time, LHCb gathered data corresponding to a total of 9/fb integrated luminosity, based on which more than 500 papers have already been published. LHCb operated at an instantaneous luminosity of $2 - 4 \cdot 10^{32}/(\text{cm}^2\,\text{s})$, beneath the nominal LHC instantaneous luminosity, to lower the detector occupancy and minimize radiation damage on the detector components.

To be able to gather data at a higher rate, the LHCb detector is currently being upgraded to be able to take data at five times increased instantaneous luminosity. The hardware trigger preceding readout is removed to achieve greater flexibility and efficiency. Chapter 4 describes the reasons, the implications and the challenges associated with the upgrade in greater detail, specifically focussing on the trigger. To cope with the increased detector occupancy, the upgrade comprises multiple changes in the detector hardware that are outlined throughout this chapter. The

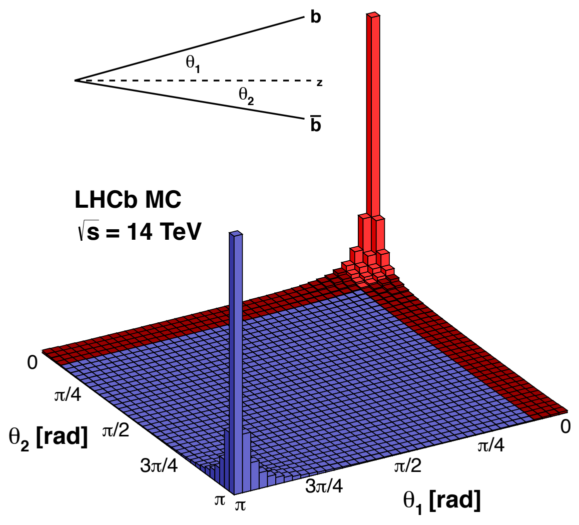


Figure 3.2: Angular distribution of $b\bar{b}$ pair production in proton collisions at a center of mass energy of 14 TeV [32]. The region highlighted in red is the angular area covered by LHCb.

detector is described as is envisioned and implemented for the beginning of Run 3 of the LHC.

A visual illustration of the detector layout is displayed in Figure 3.3. Note that, although it will not be explicitly mentioned for every subdetector, the entire readout electronics structure is being replaced to meet the challenges of triggerless readout. Now follows a description of the subdetectors that LHCb comprises as of Run 3. Detectors are classified as tracking and particle identification systems. The conventional LHCb cartesian coordinate system is used: The z-axis extends along the beam, the x-axis horizontally and the y-axis vertically with respect to ground level.

3.2.1 The tracking system

The purpose of the tracking system is spacial particle trajectory reconstruction and momentum estimation. Particles traverse tracking stations and leave evidence of their traversal through material interaction. On a high level it could be interpreted as a high frequency camera system that captures the interactions and resulting

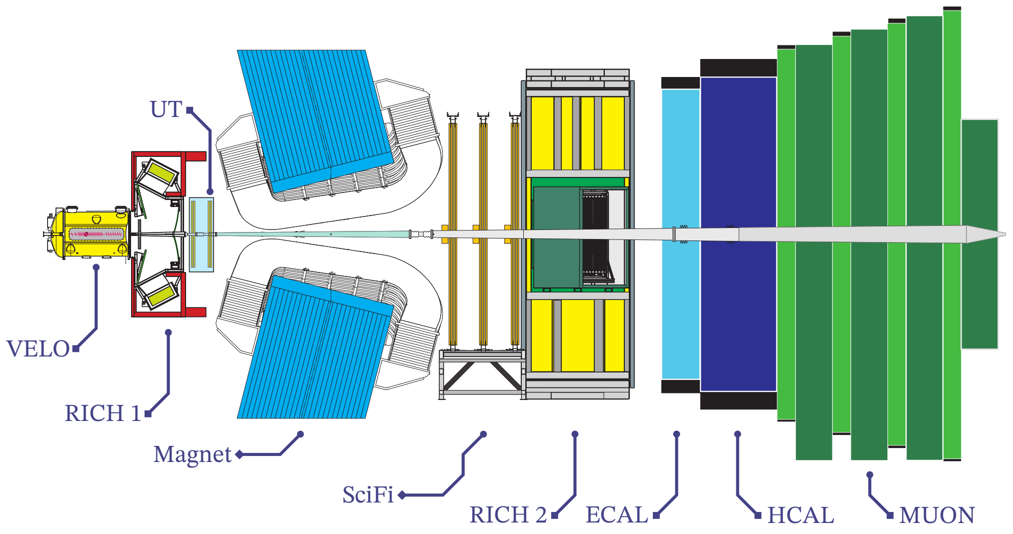


Figure 3.3: Schematic representation of the LHCb detector from the side. From left to right: Vertex Locator (VELO) where the *pp* interaction happens, Ring Imaging Cherenkov detector RICH1, Upstream Tracker (UT), Magnet, SciFi Tracking stations, RICH2, electromagnetic and hadronic calorimeter (ECAL/HCAL), and muon stations [33].

final states as closely as possible.
The LHCb tracking system consists of the Vertex Locator (VELO), the Upstream Tracker (UT), a big magnet and a completely new detector, the Scintillating Fiber Tracker (SciFi).

The Vertex Locator

The silicon VErtex LOcator (VELO) [34] is LHCb's highest accuracy tracking device. It measures a bit over 1 m in length, surrounds the interaction point and comes as close as 5.1 mm to the beam line. The beam pipe is removed to reduce material interaction before the first measurement. Only a very thin aluminum foil surrounding the beam protects the 26 VELO stations from electromagnetic induction. Each station consists of two L shaped modules which can be retracted during unstable beam conditions to avoid serious damage. The state in which the modules are retracted is called ”open”, as opposed to the default ”closed VELO” state. The VELO hosts roughly 40 million pixels that yield excellent impact parameter resolutions in the 10 µm scale. A scheme of the VELO detector modules together with the LHCb angular acceptance can be seen in Figure 3.4. Notably, the limited number of VELO modules have been placed strategically to further optimize impact parameter resolution and to assure that particle trajectories within the LHCb acceptance traverse at least 4 modules.

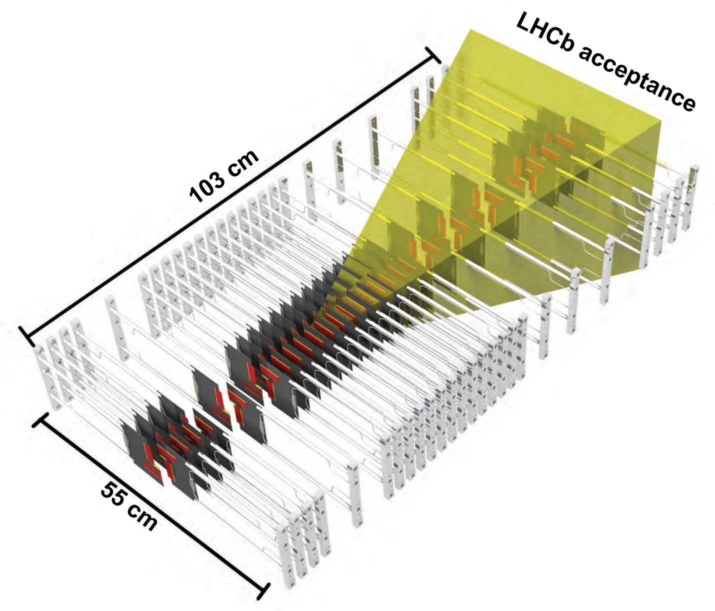


Figure 3.4: Alignment of the VELO stations with respect to the LHCb acceptance (yellow area) [34]

The Upstream Tracker

The Upstream Tracker (UT) [35] gets its name from the fact that particles traverse this tracking station just before they enter the magnet. Silicon microstrips of varying granularity based on average detector occupancy make up the four planes of this detector. The region close to the beam pipe has significantly higher occupancy, as most of the particles have a very high boost in z direction and thus a small pseudorapidity. A outline of the UT planes can be seen in Figure 3.5. The finest microstrips are placed just around the beam pipe and the coarsest on the outside. A particle-microstrip interaction yields one dimensional information, in the UT case an X position as the microstrips extend in Y direction. The two inner detector planes are therefore tilted with respect to the outer ones by an angle of $\pm\ 5^\circ$ to allow for Y position information by combining information from two interactions. This plane configuration is optimized for momentum estimation of particle trajectories.

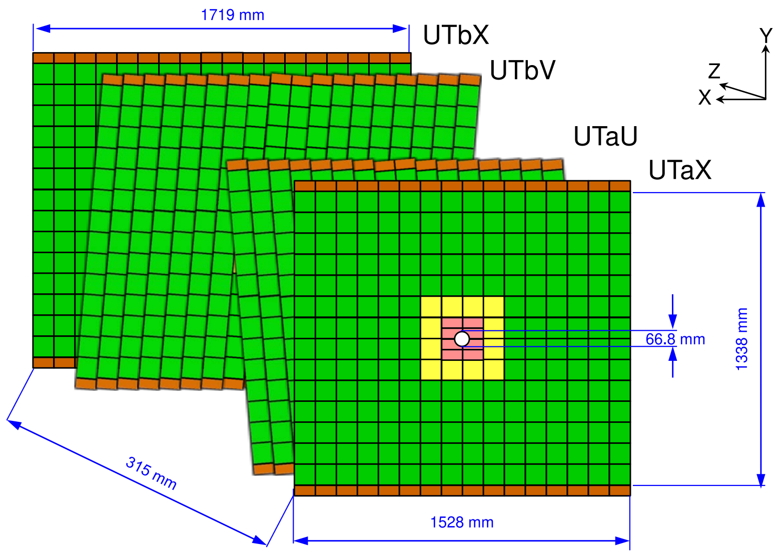


Figure 3.5: The Upstream Tracker planes and their geometrical properties [35].

The magnet

A big dipole magnet is placed directly after the UT. It is capable of producing an integrated magnetic field of 4 T m. Charged particles traversing the field are bent according to the Lorentz force. A calculation of curvature radius enables a momentum estimation as precise as about 0.5 % relative uncertainty when taking into account all trajectory position information available within the LHCb detector. The main magnetic field component of the magnet is in y direction, which results in particles being bent mostly along the x direction. As oppositely charged particles mostly end up in different regions of the detector, the magnetic field is invertible to be able to account for detection asymmetries.

The SciFi tracker

The Scintillating Fiber Tracker (SciFi) is the last tracker in the particle stream. It is placed behind the magnet to enable said momentum estimation. The placement of the SciFi detector relative to the magnet can be seen in Figure 3.6. As its name suggests, the active detection units are scintillating fibers which extend mostly along the y axis. Charged particles excite molecules in the fibers, causing a photon emission that can be detected by SIPMs at the upper and lower end of the detector modules. In total, over 10 000 km of fiber are being used to build the SciFi detector. When looking closely at Figure 3.6, one can see that there are three SciFi stations with four layers each. The middle layers of each station are tilted by 5° for the same reasons as in the UT. This configuration achieves a spacial resolution of under 100 µm in x, which is crucial for momentum resolution as the magnet bends trajectories mostly in this direction.

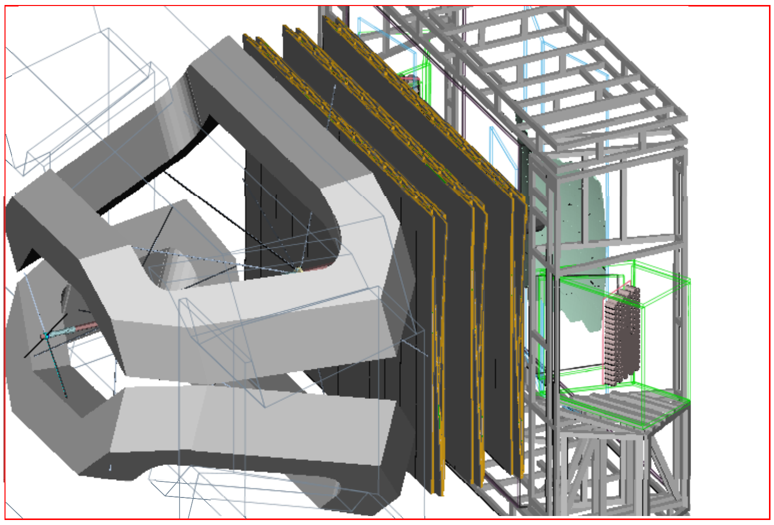


Figure 3.6: The LHCb SciFi tracker between the magnet and the RICH2 detector [35].

Track types in LHCb reconstruction

Tracks reconstructed in the LHCb detector are categorized by where they left hits. Long tracks are the most precisely reconstructed ones as they leave hits in all tracking detectors. Upstream tracks go out of LHCb acceptance before reaching the SciFi detector. Downstream tracks which originate from longer lived particles like K^0_S have no hits in the VELO. Unlike the first three categories, T tracks and VELO tracks are rarely used in later analysis as these are only visible in one of the detectors and thus miss crucial information.

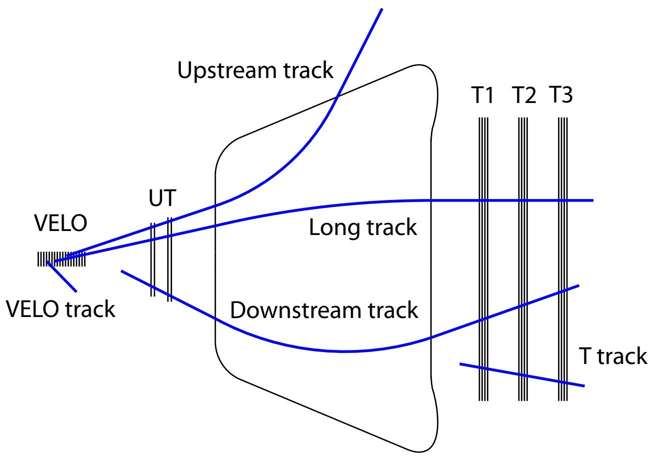


Figure 3.7: Track types in LHCb reconstruction.

3.2.2 The particle identification system

The particle identification (PID) system at LHCb consists of two Ring Imaging CHerenkov (RICH) detectors, two calorimeters and four muon stations at the very end. The combined information in these detectors yield particle hypotheses that are crucial for decay reconstruction. Within the scope of LHCb, the considered final states for the PID system to distinguish are protons, kaons, pions, electrons and muons.

The Ring Imaging Cherenkov Detectors

The two RICH detectors at LHCb are placed up- and downstream of the magnet, respectively. Their positions can be seen in Figure 3.3. If particles traverse a medium (non-vacuum) faster than light, they emit Cherenkov radiation at an opening angle

$$\cos\theta = \frac{c}{nv}, \tag{3.1}$$

with n being the material's refractive index and v being the particle's velocity. With help of a highly precise mirror system the emitted Cerenkov light cones are captured and projected onto a photo detector screen to form distinct rings. The radius of these rings gives direct information about the particles velocity.

Combined with momentum information from the tracking stations, a mass estimation can be calculated to form a particle hypothesis. The RICH detectors are especially effective in classifying kaons versus pions.
Practically, LHCb produces particles over a momentum range of about two orders of magnitude. After the upgrade, RICH1 will cover particles with momenta up to 40 GeV [36] at the full LHCb acceptance angle. Higher momentum trajectories can be effectively classified with the second RICH detector placed further downstream. As high momentum tracks mostly exhibit a small pseudorapidity, RICH2 only covers an angle of about 120 mrad.
The RICH detectors employ two different media to work on different momentum ranges: C_4F_10 and CF_4 gas. In Figure 3.8 the opening angle dependent on the particle momentum is displayed for Pions, Protons and Kaons passing through.

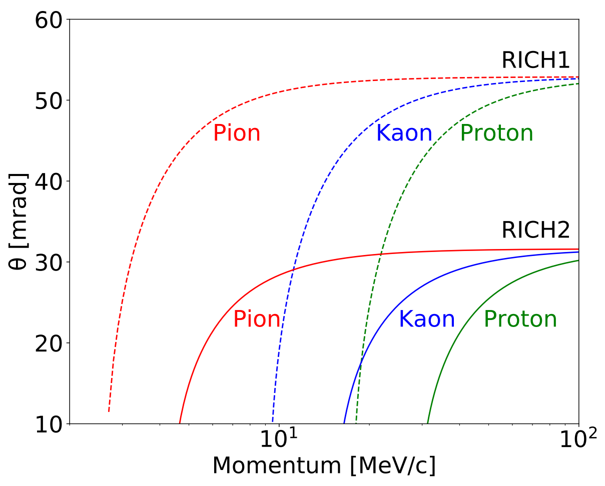


Figure 3.8: Opening angle of Cherenkov radiation emittance for π, K and p in LHCb.

Electromagnetic and hadronic calorimeters

Calorimeter systems gather PID information by observing energy depositions of incoming particles. LHCb's calorimeter system consists of an electromagnetic (ECAL) and a hadronic calorimeter (HCAL) [36]. Both are made up of alternating layers of scintillators and absorber material. The particle showers resulting from interaction with the absorbers cause photon emittance in the scintillating material.

The intensity of the emitted radiation give information about the amount of energy deposited. Photons and electrons shower when reaching the lead absorber present in the ECAL, while hadrons mostly pass through to be stopped at the iron absorber plates in the HCAL.
LHCb's calorimeter system does not change much during for Run 3. Only the readout electronics are replaced to cope with the increased readout frequency, as is done in all the other subdetectors.

The muon stations

The rear end of LHCb consists of four muon stations that serve the purpose of distinguishing muon tracks from others, and they do so extremely efficiently. Previous LHCb results have shown that muon separation is so efficient that beauty decays as rare as one in 10^{11} can be extracted from the huge amount of produced data [37].
Muons tracks are the ones capable of traversing the full detector, any other particle showers in or before the HCAL. The stations are separated by 80 cm thick iron plates to make sure that no hadron, even if it managed to pass the HCAL, traverses more than one station. Notably, a fifth station located before the ECAL was removed for the start of Run 3.

To summarize LHCb's PID system, Figure 3.9 shows how different final state particles interact with the detector and how one can unambiguously classify particle trajectories based on their traversal.

3.2.3 An overview over the LHCb software

The LHCb software framework is a versatile amalgamation of multiple packages that work together, but serve different purposes and different user groups. The main programming languages are C++ [39] and Python [40].
A high level overview is given here:

1. **GAUDI** [41] is the base for almost all other packages in LHCb. It implements an event loop and configurable algorithm and service classes that may be tweaked after compilation via python. Notably, GAUDI is not only used within LHCb, but also by ATLAS and other experiments.

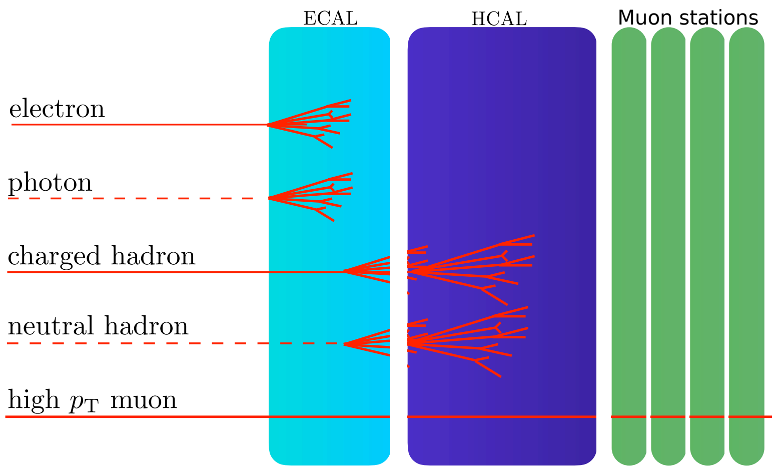


Figure 3.9: Shower pattern of different final states propagating through the LHCb PID system. Modified from [38].

2. **LHCb** [42] is the first LHCb specific software package. It implements specific reconstruction classes, the trajectory and the particle itself and many lower level helpers.

3. **Online** [43] is responsible for acquisition of data, managing the event building, data transfer to the HLT farm and monitoring of the experiment.

4. **Rec** [44] is the reconstruction package, implementing most of the trigger and offline reconstruction and particle identification algorithms.

5. **Brunel** [45] is a pure python package, implementing tests and runnable reconstruction sequences on top of Rec, LHCb and GAUDI, acting on the trajectory implementation.

6. **Phys** [46] implements selection and combination algorithms acting on particles.

7. **Moore** is a pure python package that configures the entire trigger application. This package is run in the HLT farm.

8. **DaVinci** [47] is used to configure offline reconstruction and selection jobs. It is mostly used as first input to further analysis steps outside the core framework.

9. **Gauss** [48], based on GAUDI, is used to generate hadron interaction simulations via PYTHIA [49] and EVTGEN [50] to mimick actual LHC interactions at LHCb as precisely as possible.

10. **Boole** [51] digitizes the output of Gauss. Particles resulting from the simulated interactions are propagated through a detailed description of the detector, generating simulated detector response to be able to compare real data with. Notably, several other experiments choose to unfold the data rather than implementing a detector response.

11. **Allen** [52] is a new software package to execute HLT1 on GPU architectures rather than CPUs. It will be used as baseline HLT1 implementation from Run 3 onwards. However, it will not be discussed further here.

3.3 LHCb beauty and charm decay topology

Now that the detector has been outlined, this section discusses what the kind of physical signatures we are interested in. LHCb specializes on beauty and charm decays. Due to high lifetimes of most B and C hadrons, they fly mm-cm distances within the detector before decaying. As an example, Figure 3.10 shows the topology of the decay $B_s^0 \to D_s^+(\to K^+K^-\pi^-)h^+$ within the LHCb detector. Final state particles are reconstructed from hits in the tracking stations and the VELO. Trajectories from most reconstructed tracks in a typical LHCb event are prompt, meaning that they extrapolate back to a primary vertex very precisely. However, final states from beauty and charm decay chains do not. The displacement from primary vertices thus provides a very clean and fast selection for these decays. To go further, a selection algorithm can base its decision on whether two or more tracks extrapolate back to a common point, the decay vertex of the beauty and charm hadrons. Most often, the reconstructed vertex is also displaced from the beam pipe. This represents another very distinctive topological feature of these kinds of decays. Consider for example the selection for the displayed decay. A trigger selection designed to select this decay starts by selecting kaons and pions from all final state particles with a PID requirement and displacement cut. The first task is the reconstruction of the D_s^+ vertex by combining two kaons and one pion. A D_s^+ candidate is created with a momentum corresponding to the three body combination it was reconstructed from. This D_s^+ candidate is then combined with hadron final state particles to form B_s^0 candidate decay vertices.

If any of these vertices pass a set of requirements like vertex displacement, we assume to have successfully reconstructed a $B_s^0 \to D_s^+(\to K^+K^-\pi^-)h^+$ decay and the event can be persisted.

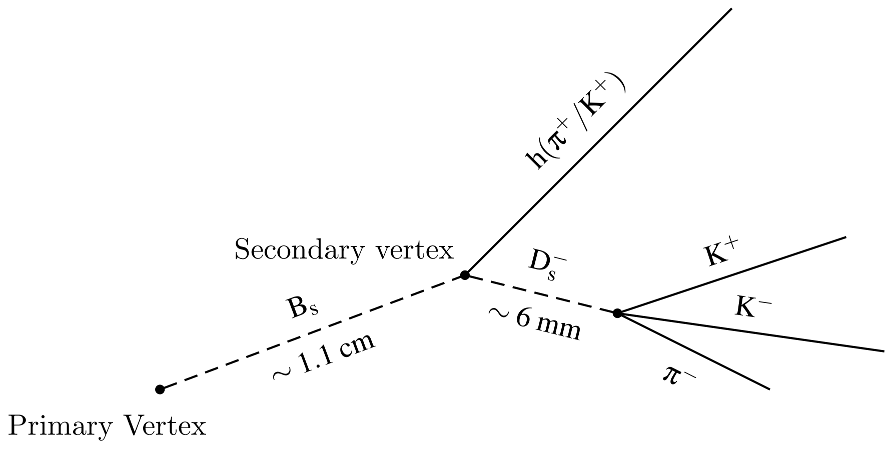


Figure 3.10: Topology of the exemplary decay $B_s^0 \to D_s^+(\to K^+K^-\pi^-)h^+$ within the LHCb detector [53]

Displaced signatures are so distinctive, that several beauty decays can be selected solely based on kinematic and topological features, making the selection inclusive over many decays. The trigger selection that attempts inclusive selections is presented in detail in Chapter 8. As charm hadrons exhibit a 20 times higher production rate [20, 18], a similar selection for inclusive charm is impossible.

The next chapter will cover the LHCb trigger system to select these signatures in detail, considering changes and challenges that the detector upgrade poses for the real time data processing workflow on the software side.

4 The LHCb Upgrade Trigger

The purpose of the LHCb detector is to capture decays as result of hadron-hadron collisions generated by the LHC and as precisely as possible. As described in Chapter 3, a tracking system is employed to reconstruct particle decays geometrically and give information about momentum. A particle identification system gathers information about the mass and the energy of final state particles. Combining this information, one is able to reconstruct entire decay chains up to the primary hadron interaction and gain deeper understanding of the basic interactions that form our universe.
However, the amount of data that 30 million 14 TeV proton bunch interactions per second yield in the LHCb detector is not to be underestimated. The detector readout has to cope with a bandwidth of 4 TB/s. Persisting this data over the course of one year of LHC run time would amount to a size of over 100 EB. Storing this much information is simply impossible given the resources available. It is also unnecessary, as the rate for events that physicists at LHCb consider interesting is below percent level of the incoming rate. We must therefore resort to distilling down these data substantially via a system called the trigger. A trigger uses information extracted from the subdetectors to make an educated guess on whether or not to discard a given event.

4.1 Why upgrade to a full software trigger?

Up to the end of Run 2 of the LHC, LHCb's first trigger selection stage was a hardware trigger, L0. The logic of this trigger stage was very simple and thus quickly evaluable, which meant that it was well suited as an initial filter step. L0 used minimum transverse momentum and minimum energy deposition requirements as selection criteria to reduce the event rate to 1 MHz. Two subsequent software-based High Level Trigger stages (HLT1, HLT2) were able to reconstruct parts of the event to make a more sophisticated decision with a better signal-to-noise ratio. Data corresponding to 9/fb of integrated luminosity were taken between 2010 and 2018.

However, the precision measurements performed by the LHCb researchers require a large amount of beauty and charm statistics. The statistical uncertainty scales of these measurements scales with the inverse root of the total amount of beauty and charm data. The upgraded LHCb will therefore ramp up its instantaneous luminosity by a factor of five to $2 \times 10^{33}/(\mathrm{cm}^2\,\mathrm{s})$ to significantly increase the available dataset. The new detector has to embrace about five *pp* collisions per bunch crossing, a factor five increase in detector occupancy and a much higher stress on the readout and computing systems. The LHCb collaboration aims to collect a dataset of 50/fb integrated luminosity within the next 10 years of operation.

The initial design for the Upgrade LHCb trigger targeted an instantaneous luminosity of $1 \times 10^{33}/(\mathrm{cm}^2\,\mathrm{s})$ and involved a new hardware trigger matching the new readout system, called Low Level Trigger (LLT). It would select similarly simple signatures as the L0 did in previous data taking periods. After the publication of the technical design report (TDR) for the upgrade framework[54], the operational luminosity was decided to be $2 \times 10^{33}/(\mathrm{cm}^2\,\mathrm{s})$. Three designs were studied to determine the feasibility of a trigger working at this luminosity level. The LLT followed by an HLT, a pure software trigger, or a software trigger assisted by FPGAs to offload certain parts of the tracking sequence to. The FPGA solution was discarded as its benefits did not outweigh the additional risk implied when adding complexity to the system.

The LLT solution was rejected because it introduces large inefficiencies in hadronic beauty and charm channels when scaled up to the operational luminosity. Figure 4.1 shows the L0 yield of the B meson decays introduced in Chapter 2.1 as function of luminosity when assuming a constant output event rate. The LLT solution would yield similar efficiencies. The selections performed in L0 and the LLT are not discriminative enough to utilize the increased luminosity. After the feasibility of the software solution was reviewed extensively [55], the LHCb collaboration decided to implement this solution [11]. The full software trigger has the additional advantage that it is much more versatile and can adapt to change in conditions faster and easier than a hardware trigger could. The LLT could be implemented in software as a backup solution.

Efficiency gains due to hardware trigger removal can be showcased with Figure

4.2. It shows efficiencies for a selection of important channels selected by the most important beauty trigger, the topological trigger, or Topo for short. The Topo and the decay channels presented in the plots will be outlined in detail in Chapter 8. The red lines correspond to efficiencies in the Run 1 implementation of the Topo trigger. The blue lines correspond to different output rate scenarios for the Topo. Note that these efficiencies are not directly comparable to efficiencies presented in Chapter 8, because the former are calculated with respect to offline selections for analyses. The gains in efficiency at the 20 and 50 Hz rate scenarios can mainly be attributed to the removal of L0 selections. While already significant in leptonic channels, the efficiency increase in purely hadronic channels is even more remarkable. This is because hadronic channels are mainly selected by the L0 hadron trigger selection, which is the least efficient L0 trigger. A green line representing twice the Run 1 efficiency is integrated into the plots for better comparison. These plots show that it is of major importance to be able to run a software trigger without the LLT in production. Note how the $B^0 \to K^{*}\mu^{+}\mu^{-}$ channel does not gain significantly with respect to Run 1, because the di-muon signature is mainly triggered by the already highly efficient L0 muon trigger.

However, removing L0/LLT implies that the HLT system not only needs to cope with a higher event occupancy, but also with a factor 30 higher input rate and the resulting output bandwidth. This Chapter introduces the trigger system in detail, its workflow and the challenges that LHCb meets with the removal of the hardware trigger.

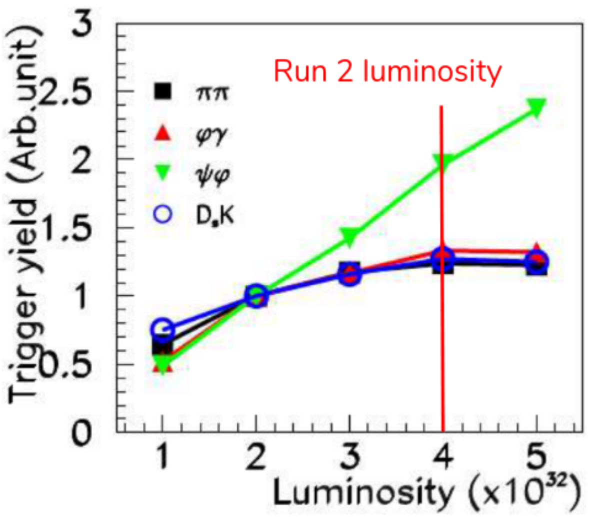


Figure 4.1: L0 yield for different B meson decays assuming a constant output rate [56].

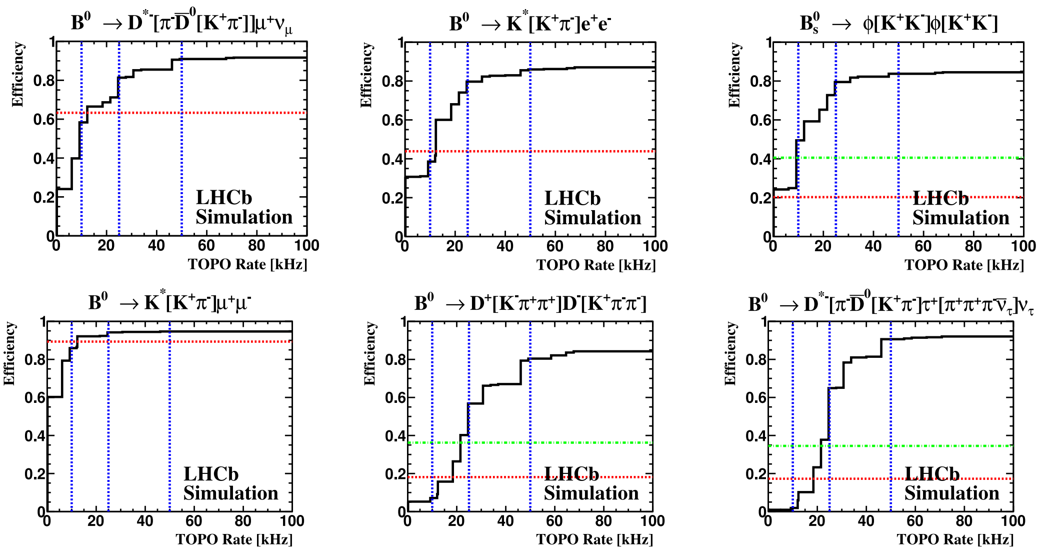


Figure 4.2: Efficiencies of the Topo trigger implementation for the upgrade as of 2014, for several beauty decay channels as function of the output bandwidth. The Run 1 efficiencies are shown as red line. For purely hadronic channels, the green line corresponds to twice the Run 1 efficiency. [57]

4.2 Upgrade trigger workflow

The planned upgrade trigger workflow can be seen in Figure 4.3. Before a trigger can run, the detector is read out and the raw event is assembled from the different subdetectors. HLT1 takes the raw event in small batches and processes in real time. It sends a first loose selection based on a partially reconstructed event to a disk buffer storage. HLT2 asynchronously reads from this storage, reconstructs the complete event and finally decides about the permanent persistence of a given event.

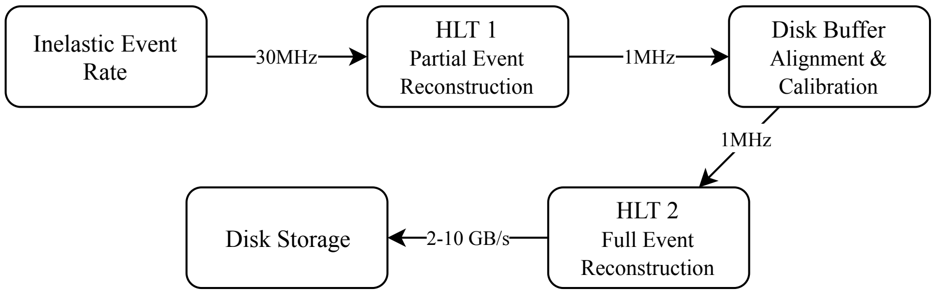

Inelastic Event Rate
30MHz
HLT 1 Partial Event Reconstruction
1MHz
Disk Buffer Alignment & Calibration
1MHz
HLT 2 Full Event Reconstruction
2-10 GB/s
Disk Storage

Figure 4.3: Upgrade trigger data flow. HLT1 has to work synchronously with the LHC, while a disk buffer enables an asynchronously running HLT2 [58].

4.2.1 The first High Level Trigger: HLT1

The goal of HLT1 is the first educated event selection based on information from a partially reconstructed event in real time. The selection is tuned to aim for a factor thirty rate reduction, from 30 MHz incoming event rate to 1 MHz output[11].

Bunch crossings at the LHC have a highly varying complexity. This variance comes from the fluctuating number of interactions per crossing and the high range of possible energies with which the partons collide. Events that have a large multiplicity and a high occupancy in the LHCb detector typically yield lower reconstruction efficiencies and worse signal purities. A global event cut (GEC) is therefore employed as the first step in the trigger. It is designed discard busiest 10 % of all events. The sum of Upstream and SciFi multiplicities is found to be a good indicator for event complexity [11].

The HLT1 reconstruction starts by reconstructing tracks with hit information from the VELO. The VELO is far away from the magnetic field, so only straight lines need to be fitted. The track candidates help to reduce the multiplicity in later tracking stages. Based on these candidates and some information of the current beam line position primary vertices are reconstructed by clustering tracks that have been extrapolated to the beam line. The next HLT1 reconstruction step matches hits in the upstream tracker to a velo track extrapolation through the first part of the detector. A slightly curved line is fit to the hits in the UT and the first momentum estimate is extracted from that curvature. Because of the small magnitude of the magnetic field before and in the UT, the momentum estimate has relative uncertainties of about 15 %. Taking into account the magnetic field model and the first momentum estimate, upstream tracks are further extrapolated into the SciFi region, where the track candidates are matched to SciFi hits. This pushes the momentum estimate to a relative uncertainty of about 0.5 %. A Kalman filter is applied to fit a velo track candidate, taking into account a momentum estimate from the other tracking stations. This decreases the uncertainty of velo track parameters. A full Kalman filter application is too expensive for HLT1. These steps conclude the HLT1 upfront reconstruction that serves as input to almost every selection criterion applied in HLT1. An outline can be seen in Figure 4.4.

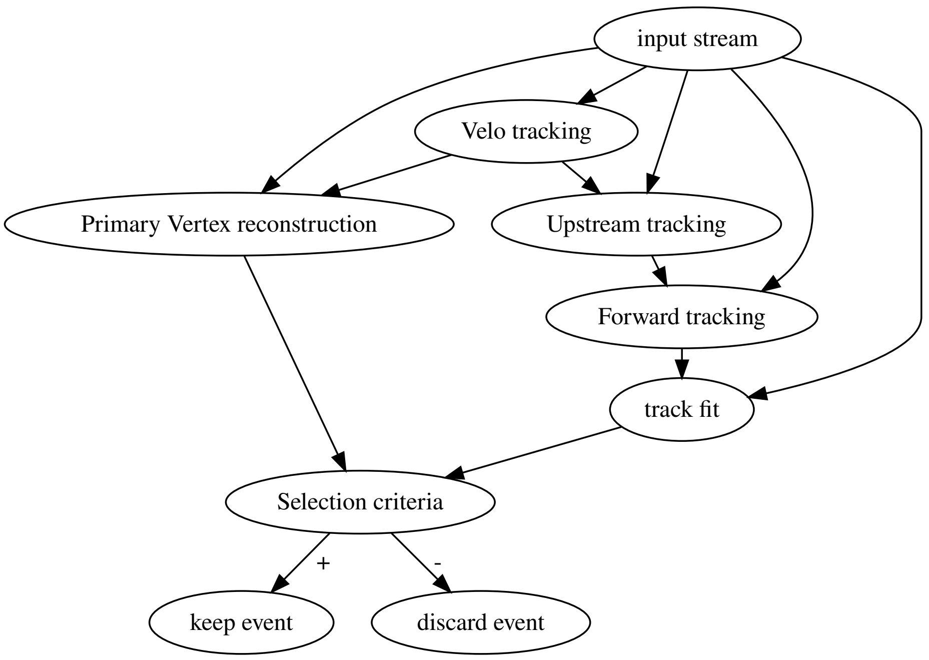


Figure 4.4: A simplified scheme of the standard HLT1 data flow.

Selection criteria in the upgrade HLT1 are very similar to the ones employed in previous data taking periods. We realize the application of these criteria in so-called trigger or selection lines. A line defines the required reconstruction steps and requirements on certain signatures in the event topology to form a final decision. Bandwidth reduction by selections always happens per-event in the first trigger stage. That means that, whenever any selection considers a signature interesting, the full event information will be temporarily persisted into the intermediate disk buffer.

Several lines require thresholds on simple properties of single tracks, others combine tracks to vertices and select based on these. The most prominent single track line is tuned empirically to select tracks that are likely to originate from a bottom or charm decay and it involves requirements on transverse momentum, track fit quality and a minimum displacement from reconstructed primary vertices. A more sophisticated HLT1 selection involves additional reconstruction, specifically the combination of two tracks into a decay vertex candidate and a subsequent vertex fit. This line can base its decision on not only track, but also vertex properties like the vertex quality as result of the fit and the vertex displacement from the beam line. It provides an even more discriminative selection of beauty and charm decays.
LHCb's physics program is not only interested in displaced signatures however. By requiring matching hits in muon chambers, we can select muonic trajectories in HLT1 very purely even if they show no PV displacement. This opens up possibilities like searches for dark photons [59] and also contributes significantly to the efficiency of most selections involving any muonic final state, like $B_s^0 \to \mu^+\mu^-$.

4.2.2 The disk buffer, alignment and calibration

After events pass the first HLT stage with a rate of about 1 MHz, they reach an intermediate persistence step within the disk buffer. The information gathered and processed in HLT1 and stored in the disk buffer is used to perform a real time alignment and calibration of the detector. Already in Run 2 LHCb has operated a real time alignment and calibration system to enable offline-quality reconstruction in the trigger [60]. Examples of real time alignment are the adjustment of the beam line position for a primary vertexing algorithm or the recalibration of the RICH mirror system to maintain high reconstruction performance for the Cherenkov cones.

Most importantly, the disk buffer allows an effective relaxation of the real time processing requirement to a, to some extent, asynchronous operation of the second HLT stage. As we can store HLT1 reconstructed data for up to two weeks, HLT2 can be operated much more flexibly and efficiently based on current demands. To understand why the existence of a disk buffer can lead to increased efficiency, we must keep in mind that the LHC is not continuously running over the course of a Run. One reason is the operational cycle of the LHC [61]. It takes at least two hours to go from initial hydrogen ionization to stable beam conditions and proton beams are held in the LHC for a maximum of about 12 hours. Practical problems result in a higher ramp up and a lower stable beam time, resulting in a stable beam time of less than 50% on average in Run 2. Moreover, there are machine development (MD) periods and technical stops (TS), in which the LHC also does not run. A HLT2 running synchronously would idle whenever there is no stable beam. With the disk buffer however, the work can be distributed over time. Figure 4.5 shows the disk buffer usage in 2016. One can see how during the shaded periods, the disk buffer got emptier as the data was processed by HLT2. In conclusion, HLT2 could utilize processing time corresponding to roughly twice the time in which stable beam conditions were present in the LHC.

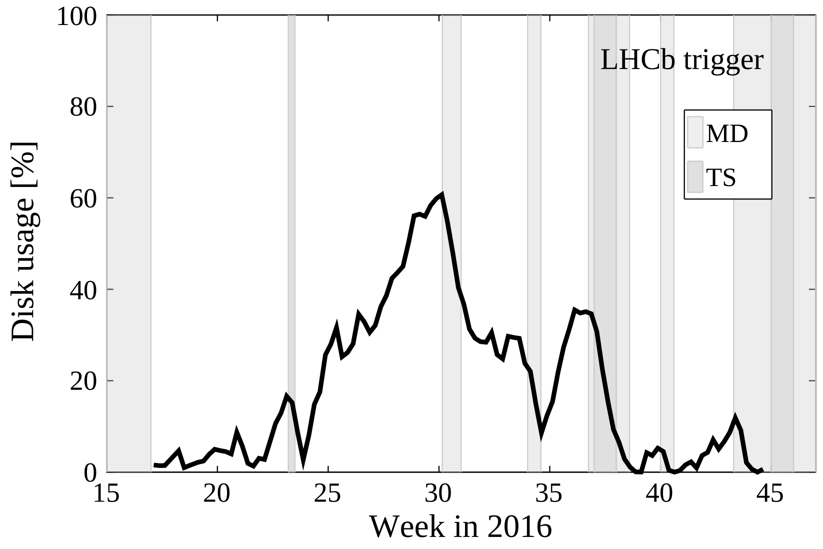


Figure 4.5: LHCb disk buffer usage in % in 2016. The shaded areas mark technical stops (TS) and machine development (MD) [62].

4.2.3 The second HLT stage

As previously mentioned, HLT2 runs asynchronously to the LHC. It takes event input from the disk buffer and aims to perform offline-quality reconstruction and selections to bring incoming event rate of 1 MHz down to a bandwidth of 10 GB/s. Offline quality reconstruction refers to the most precise calculations, taking into account all calibrations and alignment, even though that might cost more computing resources than simplified calculations in HLT1. HLT2 employs similar reconstruction steps as HLT1 and more. Aside from a high precision trajectory reconstruction, the PID system involving the RICH and both calorimeters helps to form particle hypotheses and neutral particle candidates.

Like in HLT1, the output bandwidth in HLT2 is controlled by a set of trigger lines. However, while HLT1 can use O(10) lines to perform the required bandwidth and rate reduction, HLT2 aims to distill the residing data down by another order of magnitude. This can hardly be achieved by the type of inclusive requirements that HLT1 employs. The average type of HLT2 line therefore specializes to select only a very specific decay structure efficiently, involving requirements on track and vertex topologies as well as PID variables. Most trigger lines perform secondary and tertiary vertexing to try to reconstruct the entire decay chain they want to select, as outlined in 3.3. The main disadvantage of exclusive selections is the lack of generalization. To satisfy research interests of all LHCb physicists, upgrade HLT2 needs to account for a total of O(1000) trigger lines that need to make a decision in every event. A union of all trigger line results will then decide over the persistence of a given event. This concludes the high level description of the LHCb trigger system.

4.2.4 Building blocks for trigger software

The framework for LHCb trigger applications defines principles for the workflow that the HLT farm uses for online event processing. It is called Gaudi. As base framework, it implements many generic components, the most important of which will be outlined here. To be able to assemble many different trigger selections with preceding reconstruction in a modular fashion, Gaudi implements the basic building block for processing event data, the algorithm. A Gaudi algorithm is essentially a configurable function with event data input. Every piece of reconstruction or selection software is built upon this base. Algorithms in Gaudi interact with a data store, the Transient Event Store (TES), for their inputs and outputs. This store has the lifetime of one event. All data that is exchanged between algorithms during the event reconstruction resides in the TES. This way, several algorithms can run on the output of another. GAUDI also defines Services that handle operations which are independent of events, like algorithm scheduling, detector alignment, process configuration or provision of metadata like detector conditions.

Configurability and the build model

Although algorithms already represent fairly granular building blocks, the amount of required configurability is often even finer grained, specifically in selection algorithms. GAUDI algorithms therefore enable specific configurable parameters to their implementation. While performance critical code in LHCb is written in the low level language C++, GAUDI exposes user defined parameters to the outside which can be tuned dynamically via a python front end. Over the years of GAUDI's existence users have found more and more places, online and offline, where configurability via python was desirable. Consequentially formed the current GAUDI workflow model:
At compilation time, representations of algorithms and services are exported into a python environment. Algorithm and service developers control the configurable parts of their code by exposing properties as class members to the respective algorithm or service. These python representations may be overwritten or changed at the users leisure in an options file. The parameters set in an options file are then applied to the GAUDI algorithm whenever a configuration step is invoked. Chapter 7 introduces configuration that involves more than mere setting of parameters to be able to achieve more versatile selection algorithms.

4.3 Computing challenges in the Upgrade HLT

This section presents the computing challenges that we face to produce a software trigger system capable of processing 30 MHz of incoming event data and distilling it down to an output rate of 10 GB/s in a signal preserving manner.

4.3.1 Bandwidth requirements

Tuning the upgrade trigger towards the desired output bandwidth efficiently presents a major challenge. The previously introduced concept of exclusive selections alone does not satisfy the requirements. Up to now, the concepts of rate and bandwidth have been used quite interchangeably. Rightfully so, assuming a constant event size and per-event trigger decisions. However, HLT2 employs a more fine-grained decision technique to help with bandwidth reduction, which is the ultimately relevant merit. This kind of selective persistence already existed in Run 2 but will gain much more relevance in the upgrade. The L0 removal is estimated to cause a factor of two effective signal yield. Together with the fivefold increase in instantaneous luminosity this combines to an effective tenfold signal yield per unit time [63]. A factor three increase in event size with respect to Run 2 is estimated based on the average number of interactions per event and the ratio of event size in a Run 2 signal event with respect to the average event size.

Selective persistence

In pre-upgrade HLT2, trigger line authors made a conceptual decision for the persistence of events they deem interesting: Full stream or selective persistence (SP). Whenever a trigger selection configured for full event persistence gives a positive decision, all available information on the bunch crossing is recorded. If on the other hand a line is configured for the SP, only a user-specified subset of information is going to be persisted permanently. The common SP configuration restricts persistence to information concerning the final state trajectories with which an exclusive line built a viable beauty or charm candidate. In 2018, roughly 32 % of all lines went through SP, reducing event size by about 50 %. In the upgrade conditions, event size reduction is much more efficient due to the increased number of interactions per event. In Ref. [63] the event size reduction for an SP event in the upgrade trigger is estimated to be about a factor seven. To achieve the bandwidth requirements of 10 GB/s in the upgrade baseline conditions, we require about 70 % of all lines to persist selectively. A visualization of different scenarios and their respective output bandwidth can be seen in figure 4.6.

A transition from full stream to selective persistence involves considerable effort for every trigger line author. One needs to ensure a consistent, efficient and bug-free selection and assert that relevant information for later analysis does not get lost in the selective persistence step. There were several occurrences

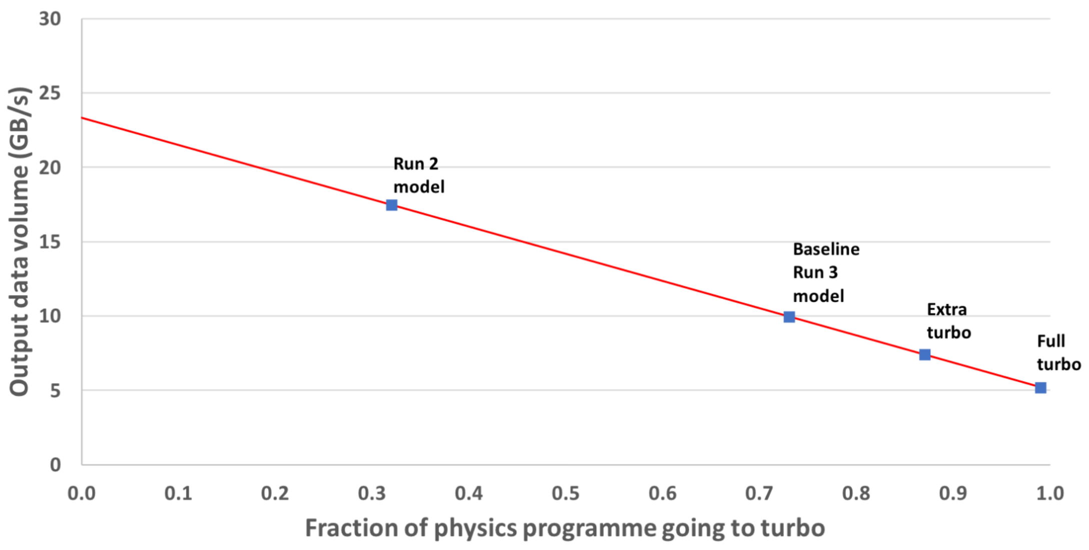


Figure 4.6: HLT2 output bandwidth in upgrade conditions depending on the relative fraction of the physics program using selective persistence [63]

in previous runs where data analysts benefited from information that was not directly related to the signal candidate. As correct reconstruction becomes more important, the need for error-free software is more important than ever. A bug in the momentum-correction due to bremsstrahlung made several electron data unusable in 2016. This would have been correctable in full stream.

Chapter 8 describes possible bandwidth optimizations including selective persistence for the topological B-trigger, a prominent and widely used inclusive trigger line that takes up the biggest amount of the total bandwidth.

4.3.2 Throughput requirements

Until the end of Run 2 the software HLT successfully processed an event rate of 1 MHz as determined by the output rate of L0. As L0 is removed for Run 3, we will observe a thirty-fold increase in incoming event rate into HLT1. The fivefold increase in instantaneous luminosity further increases computational complexity at least linearly. We therefore require an increase in processing power by two orders of magnitude with respect to the Run 2 HLT computing farm. The technical design report for the upgrade trigger [11] laid out a detailed plan in 2014 on how to meet the requirements. More efficient reconstruction algorithms to be used in upgrade HLT1 had already been developed. A total cost estimate of 2.8 MCHF for the HLT processing farm was calculated under the assumption of a computational power per price growth of 1.37 per year over ten years, as seen in Figure 4.7.

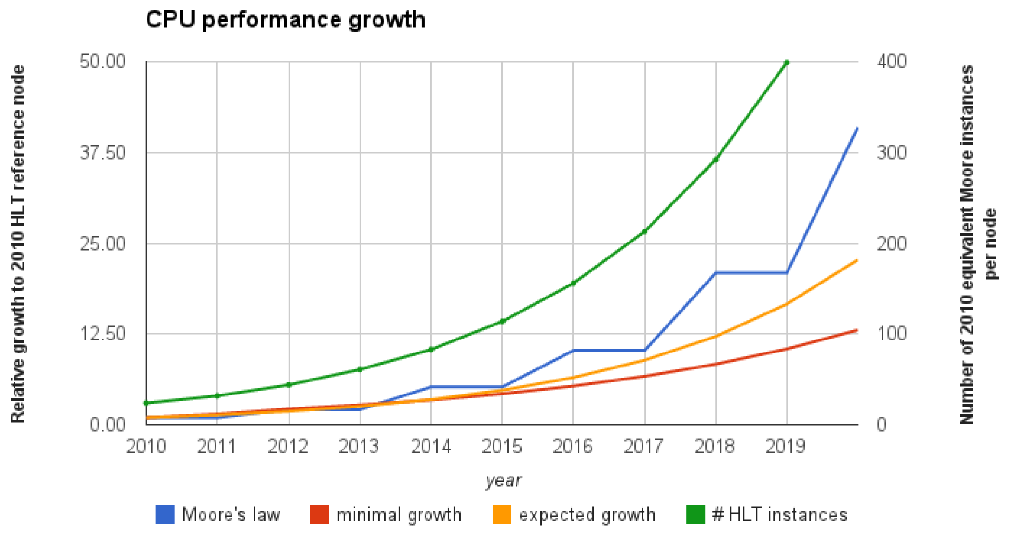


Figure 4.7: Expected CPU computational power growth per time [11].

To general disappointment, the annual growth in performance per price was vastly overestimated. An updated annual growth rate of 1.1 has been calculated in 2017 in [64].

All considerations taken together, the HLT1 stage was a factor six too slow to be operating in 2021 with a budget of 2.8 MCHF. The HLT2 stage was not thoroughly tested in these calculations, but it was assumed to be factors away from its goal.

The task is thus to achieve a factor six in computational efficiency on the given resources via more efficient reconstruction implementations and a low-overhead framework.
The low overhead framework gets much more important when we consider the task that HLT2 has to achieve. In Run 2, there were about 500 HLT2 lines acting on each event at a rate of about 100 kHz. The upgrade HLT2 on the other hand will have to manage about O(1000) trigger lines at 1 MHz input rate. The increased number of lines is a result of the higher degree of required exclusivity to fit into the bandwidth limit. Another contributing factor is the broader physics program that LHCb physicists envision for the higher data rate, making LHCb almost a general purpose detector. Running O(1000) lines at ten times the input rate and

higher luminosity is a challenging task in which computational efficiency of the framework plays a crucial role.

5 Principles for High Performance Computing in the Trigger

There are many methods to decrease runtime of an algorithm, several of which are specific to the algorithm itself. This chapter will first describe some general and higher level principles for code optimization in the dominant CPU architecture of the last decades, the x86 CPU. Specifically, four phenomena will be looked at in greater detail: Cache efficiency, predictability, dynamic memory allocation and vectorization. The last section of this chapter discusses the transition of the trigger software to multi-threading to set context for the following chapter.

Before diving into the details, I would like to mention the most important code optimization technique very explicitly: Identify and restrict yourself to necessary work. All other code optimization techniques only make sense as soon as the logic of an algorithm is minimal to achieve the task that it is designed to do.

5.1 Caching and predictability in CPUs

The first two phenomena are described together because they are often correlated when one successfully optimizes an algorithm. To discuss caching, a model of CPU data storages is shown in Figure 5.1. L1 and L2 caches reside at the core itself, while L3 is shared between multiple ones. Memory and Disk is shared over all CPUs in the machine. While the caches have a much smaller capacity than main memory, they are also much faster to access. When a CPU operation requires data in form of a memory address, the CPU loads this data from disk or memory through the caches into the registers the operation works on. Data is loaded in fixed size chunks to populate the caches. The next time a CPU operation needs data, it may find these data in one of the caches. The closer the memory address is to the one required during the first load, the more likely is a so-called cache hit, meaning required data was found in the cache. A cache hit is very beneficial for runtime, because the cache access times are much smaller than a load from memory and the application can continue processing much faster.

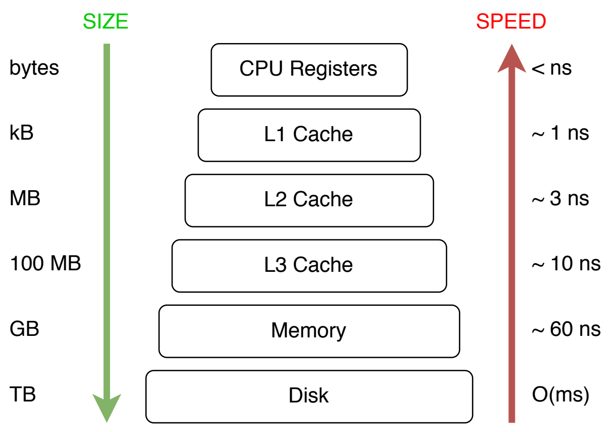


Figure 5.1: Data storage in a modern CPU architecture, ordered by physical distance to the compute units.

The modern CPU tries to prefetch data as much as possible. Whenever it can see that data is going to be required in the near future, it can prepare the data beforehand to ensure a continuous utilization of compute units. However, the CPU goes even further with branch prediction. Applications often branch based on runtime behavior. Every if statement is an example for that. The branches of an if statement may require different data to be loaded. A CPU tries to predict the outcome of the if statement based on historical data and prefetches the data corresponding to the more likely branch. Applications with many branches can greatly profit from correct branch prediction, as data dependencies are effectively reduced and the compute unit spends much less time waiting for data. However, efficient prefetching can only go right if an algorithm is either not branching or if branches have very predictable outcome. To summarize, algorithms with a very linear memory access pattern have both high cache hit efficiency and a high predictability. They can use prefetching to utilize the available compute power well. Random accesses on the other hand kill CPU performance. When the predictor fails to predict the right path, the CPU has to invoke another load and the computation has to stall for a long while, especially when the load has to go to memory. In Chapter 7 a data model is introduced that keeps all necessary information for computation close together in memory to increase cache efficiency. We also remove runtime branching wherever possible to reduce reliance on the branch predictor.

5.2 Dynamic memory allocation

Memory allocation in general refers to the reserve of a block of space in memory to store some data on. There are two types of memory in a CPU, stack and heap memory. The Stack is a Last-In-First-Out (LIFO) structure on which data is simply stacked. Only data on top of the stack can be removed. The size for allocation and deallocation on the stack is defined at compile time. It is therefore referred to as static memory allocation. If one needs a runtime specified amount of memory, one can resort to the heap memory and dynamic allocation. The heap is randomly accessible and as such is much more versatile, but also costly to operate on. It needs to keep track of allocated and free space and a request for a contiguous memory with certain size must search for such a block explicitly. During the course of execution, the heap may become fragmented. This is often a reason for low cache efficiency and longer searches for free blocks. The CPU has means to reduce fragmentation, but these also cost CPU cycles. Repeated dynamic allocation has been and still is a major slowdown in many algorithms in LHCb. Chapters 8 and 6 present algorithm designs that have minimal dynamic memory allocation. If they need it, they will resort to allocation of big chunks at once as opposed to many small ones.

5.3 Vectorization

Modern CPUs have a vast amount of instructions for loading transforming and storing data. Vectorization refers to the usage of a specific kind of instructions: vector instructions. These instructions work on a whole collection (or vector) of floating point or integer data at once. For most scalar instructions like the add operation, there is a vector instruction to perform the same logic on multiple data at once. The effect of a vector operation can be seen in Figure 5.2. Note that a vector instruction has the same latency as a scalar operation, so one can theoretically achieve a speed up by a factor equal to the width of the vectors operated on. In practice, the application needs to be parallelized to the degree that using vector operations make sense. Data to be operated on needs to be contiguous in memory. Structures might require reorganization to get the data in question into the right layout to use vectorization, which often costs more time than vectorization saves. It is thus only efficient to use if multiple instructions can be performed on the contiguous data, because reordering can invoke significant cost. Chapter 7 will introduce a data structure that can use vectorization effectively.

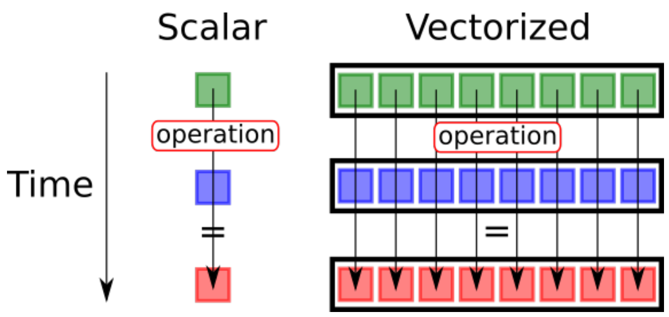


Figure 5.2: Scalar vs. vectorized CPU operation [65]. All entries in the collections on the right need to be consecutive in memory.

5.4 Multi-core utilization with multi-threading

The single core performance of CPUs has risen exponentially, by about a factor two every two years for over 40 years, until around 2005. This exponential growth stopped at frequencies around O(1 GHz), beyond which heat dissipation is too much to be cooled by air or water coolers. The key to continual performance growth was then found in parallelization. The first CPU with a second core was released around the mid 2000s. That trend of increasing parallelization has continued to this day. Figure 5.3 summarizes the key indicators for CPU performance in a 42-year span until 2017. AMD has released its first commercial 64 core CPU in 2019. Additionally, a technique called hyper-threading came up in 2002, which enabled further parallelization by assigning two processing threads to one CPU core. Two threads share the physical core, but can interleave each other whenever tasks on the other thread temporarily uses only parts of the resources. This typically yields a 30 % to 40 % performance boost.

The LHCb trigger application is a great candidate for parallelization as we assume independence between consecutive events. Independent work is an essential prerequisite for efficient parallel execution. In Run 1 and Run 2 the trigger application launched multiple independent processes, each processing portions of the full incoming data to make use of the available computing cores. This involves a large memory footprint as every process has to execute the entire application and no process can share memory with any other. Another disadvantage is inefficient

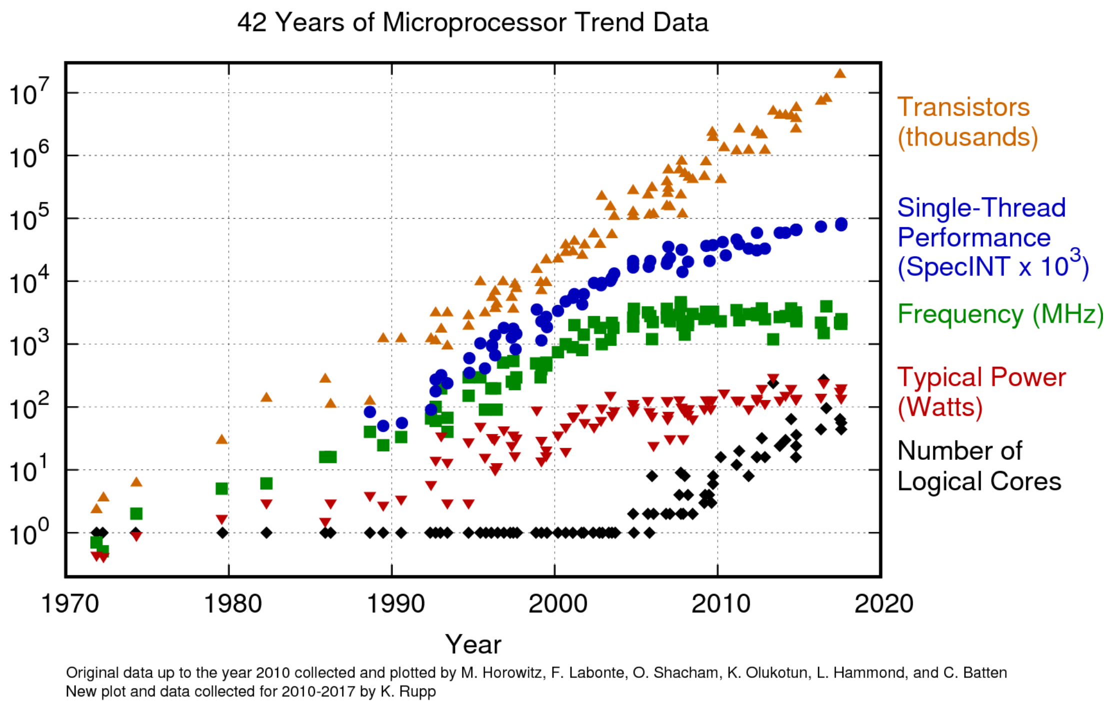


Figure 5.3: 42 years trend of computational power in CPUs [66]

handling of CPU blocking operations like IO [67]. Large efforts have been put into the development of a Gaudi application that allows multi-threading [68]. This results in a cheaper thread spawn, a higher cache efficiency and a smaller memory footprint due to shared memory.

When developing a multi-threading application, one must handle the state of the application carefully when it can be mutated by every thread. Because execution order of threads is not deterministic, incautious handling of shared state causes race conditions that result in hardly reproducible problems. Pre-upgrade algorithms in the Gaudi application used shared state in their class instances for processing. In a multi-threaded application this does not allow for one algorithm instance to run on multiple events at once. As creation of new instances for every event cost time and memory, Gaudi transitions to an algorithm model where one instance allows processing over multiple events at the same time. This algorithm model is called ”Gaudi::Functional”.

Gaudi::Functional

As every algorithm may contain (configurable) state, access in a multi-threading environment should only happen without mutation of said state. To enforce this crucial requirement as rigorously as possible for developers of reconstruction and selection algorithms, a constrained algorithm base implementation exists in upgrade Gaudi: Gaudi::Functional. Inspired by functional languages, Gaudi::Functional mimics the invocation of a pure function. Pure functions are defined by two properties:

1. Given the same inputs, it always yields the same output.
2. An invocation of a pure function may never mutate state beyond its own scope, it is said to have no side-effects.

Gaudi::Functional enforces both requirements as much as the C++ language allows. Moreover, a functional algorithm can be read much more intuitively. Input and output data are now part of the call signature. Inputs and outputs to legacy algorithms may only be found within the function body.

Gaudi::Functional implements many possible input and output structures and names them accordingly for increased readability. Figure 5.4 displays the most commonly used categories:

- The **Consumer** receives data and returns `void`.
- The **Producer** generates data without inputs.
- The **Transformer** receives and produces data.
- The **Merger** takes a runtime-specified amount of same-typed inputs.
- The **Splitter** generates a runtime-specified amount of same-typed outputs.

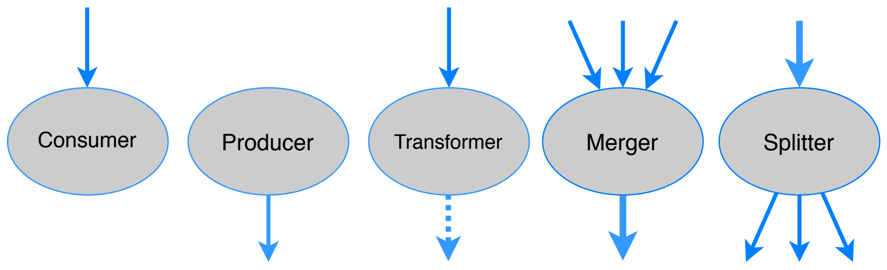


Figure 5.4: Functional algorithm types in Gaudi.

The next chapter introduces a fast scheduling algorithm that multi-threads and makes use of the features introduced with Gaudi::Functional.

6 A Scheduling Algorithm for the Upgrade Trigger Regime

The most significant recent change in the GAUDI framework has been the utilization of multi-threading as opposed to multi-processing. This change required the redesign of many low level components to meet requirement of thread-safety. In the heart of the framework environment sits the algorithm scheduler defining the logic for event processing. This scheduler must decide when to run which tasks to achieve the result. A multi-threading application requires the scheduler to dispatch tasks to different threads such that efficient parallelization over CPU cores is possible. The first implementation of such an algorithm was defined in Ref. [69]. This chapter starts by defining some concepts and nomenclature. The existing scheduler will then be outlined shortly. Afterwards, a new and vastly different implementation to meet the LHCb Upgrade requirements better will be presented in greater detail. Most general ideas for the new algorithm were worked out in a team of 4 people. The lower level ideas, details, and the implementation were provided by me. It has been published in Ref. [70].

6.1 Control and data flow

Two terms used excessively within this and the next chapter are control flow and data flow. Data flow refers to the path data takes through a processing environment. It describes how algorithms depend on each other via the means of input and output relationship between them. An algorithm is data dependent on another when its input is produced from the other. It implies that one algorithm has to run before the other.

Control flow refers to the mechanism of controlling in which direction an application goes. A basic form of control flow is a branch, for instance implemented via an if statement. This statement defines that under some condition the algorithm takes either one of two paths. An equivalent, but more implicit control flow is used often throughout this chapter: Short circuiting. A boolean operation can often be evaluated without all operands included in the process. Consider a binary boolean conjunction: $A\&B$. If A turns out to yield false, then the output of B does not matter for the result, it is false either way. The equivalent happens for a boolean disjunction $A|B$ when A yields true. Short-circuiting (or lazy) boolean operations evaluate as soon as possible without requesting unnecessary data. If A and B both require processing to yield an output, the processing for B would not happen.

6.2 The baseline: A multi-threaded event scheduler for Gaudi

Traditionally, sequential Gaudi implemented very basic control flow. Whenever one algorithm yields a negative control flow decision, the sequence would simply terminate. The development of a multi-threaded Gaudi involved the creation of a new event scheduler [69]. The so-called Avalanche scheduler implements task-based parallelism over and within events. Via inspection of data and control flow dependency graphs of a given task collection the scheduler can maximize intra-event occupancy, thus minimize the need for inter-event concurrency. Maximized intra-event concurrency proved to maximize peak throughput of the pre-upgrade LHCb reconstruction.

The Avalanche scheduler ranks algorithms by their importance in event processing graph given data and control flow precedence. At any given decision step, all available ready-to-run algorithms are scheduled in order of descending importance. The measure of importance is motivated by the critical path (CP) of the precedence graph, that can be calculated using the critical path method [71].
The calculation of the CP involves knowledge of the runtime of algorithms in the graph. Unfortunately, bunch crossings at the LHC vary greatly in occupancy and topology. As reconstruction algorithms depend on these characteristics, a upfront calculation of algorithm runtime is nearly impossible. The Avalanche scheduler therefore assumes a uniform time distribution across all algorithms. Under that assumption, the critical path is automatically also the longest. The only relevant importance of a given algorithm becomes the eccentricity in the precedence graph. Eccentricity of a vertex in a graph is defined as the maximum distance to any other vertex. Note that in this case it is defined with the precedence direction.

Only paths along the same direction as the arrows are considered. Ranking according to eccentricity has been shown to yield the best CPU occupancy and therefore throughput results in pre-upgrade LHCb reconstruction. A visualization of the reconstruction precedence graph with encoded node eccentricity can be seen in Figure 6.1. The more red an algorithm is colored, the higher the rank in the scheduling logic. The implementation of this scheduler in GAUDI yielded a 34 % increase in intra-event throughput on pre-upgrade LHCb reconstruction with respect to previous, reactive scheduling techniques [72].

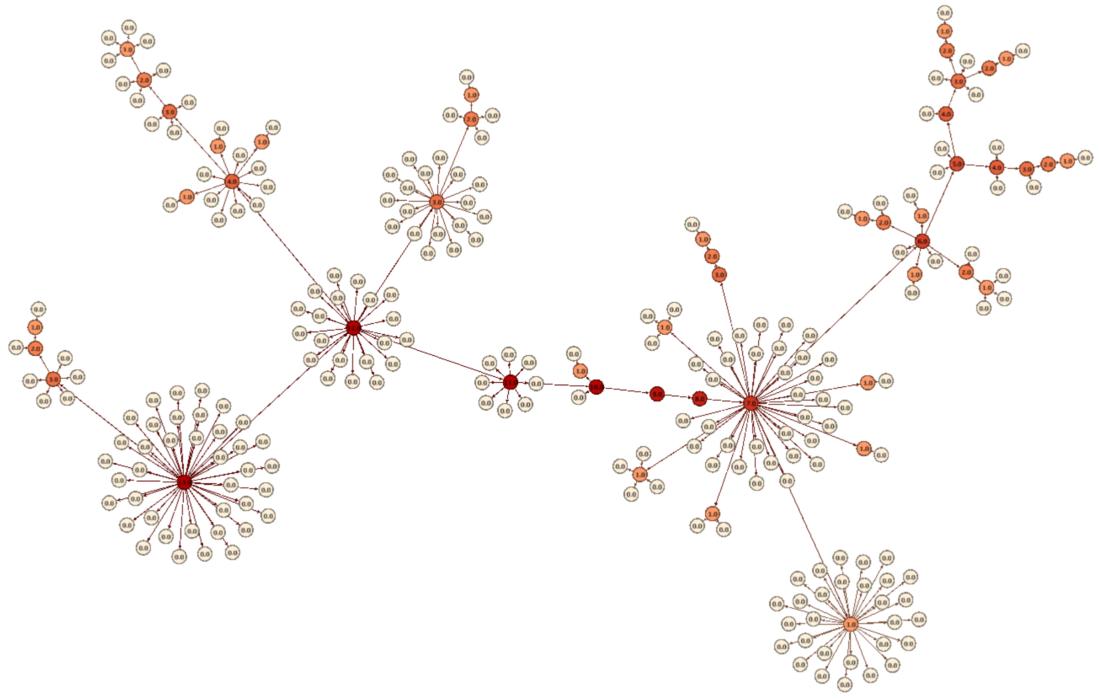


Figure 6.1: Pre-upgrade LHCb reconstruction in a precedence graph. More intense colors correspond to greater eccentricity [69].

While this scheduler is suitable for performance regimes present in Run 1 and Run 2 LHCb, it is no option for the upgrade HLT. The dynamic aspect of this scheduler involves much runtime bookkeeping, like the gathering of algorithms that have their control and data flow dependencies met at O(100) time steps throughout the execution of one event. The decision-making time per processed event, given about 250 tasks, is at least 2.5 ms. This scales roughly linearly with the number of tasks in the event.

A quick estimation on CPU time per event in upgrade conditions shows that we have little time to spare for extensive scheduler logic: Assuming roughly 1000

CPU nodes in the HLT processing farm, HLT1 has to run at 30 kHz per node. Let us assume a typical node in the farm to comprise about 30 physical cores. The CPU time per event calculates as follows:

$$T_{\text{event}} = \frac{\#(\text{nodes}) \cdot \#(\text{cores per node})}{R_{\text{tot}}} \approx \frac{1000 \cdot 30}{30\,\text{MHz}} = 1\,\text{ms}, \qquad (6.1)$$

where R_{tot} is the inelastic collision rate going into the HLT. With these assumptions each event in HLT1 has about 1 ms CPU time to be processed completely. Even though these assumptions may be off by several factors, the tension between 2.5 ms decision time and 1 ms processing time renders this scheduling algorithm unsuitable for HLT1. HLT2 has about a factor of 30 more time. When extrapolating the linear decision time of 250 tasks to several thousands per event, the infeasibility of employing this scheduler becomes apparent in the HLT2 environment aswell. Even when assuming only 1000 tasks, the 10 ms decision time per event would take about one third of the entire processing time in HLT2. In conclusion, the Avalanche scheduler is not suitable for upgrade HLT conditions.

6.3 A new scheduling application

6.3.1 From intra- to inter-event concurrency

The decision time for in-flight scheduling is a show stopper in the Upgrade HLT regime. One could reduce the decision time by bundling tasks together and thus effectively reduce the number of decisions made per event. In fact, the most reasonable task bundling that comes to mind is bundling full events. That means that scheduling logic within events has to be done statically. However, this also means that task scheduling decisions become trivial as there are no dependencies between events. The eccentricity of each task is 0. We call this approach inter-event concurrency as opposed to the intra-event concurrency in the avalanche scheduler. The new algorithm outlined in the rest of this chapter builds on this idea.

6.3.2 The high level workflow

When considering only inter-event concurrency, events can be scheduled in a fixed manner without computationally intensive checks or calculations during runtime.

A high level workflow for the operation of a new scheduler implementation could then look as follows:

1. Define the control flow and the data flow of the desired application
2. Generate an order of execution at configuration time that meets all the precedence rules extracted from the graph
3. Cooperatively push full event tasks to worker threads, where the given instruction order is executed with minimal runtime overhead.

6.3.3 The trigger control flow anatomy

The trigger application is essentially a sophisticated filter algorithm. It receives event information as input, reconstructs properties of interest and then decides based on these properties. On the highest level, the trigger can essentially be summarized as in Figure 6.2.

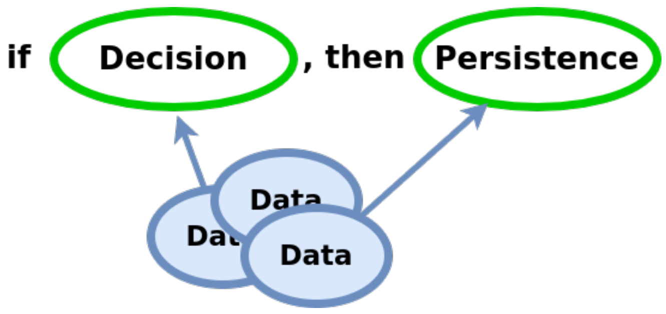


Figure 6.2: The trigger, from very far away. The blue arrows represent data flow.

The decision is ideally true for everything that physicists in LHCb consider worthy of persistence and false otherwise. As we did not yet manage to find a discriminative enough heuristic to define ”worthy”, the decision is a union of many parts, mostly exclusive trigger selections (or trigger lines) to reconstruct a specific set of decays. There were over 500 trigger lines in Run 2. We want to save an event if any of these lines considers the event interesting. Summarized, one might depict the trigger work flow as done in Figure 6.3.

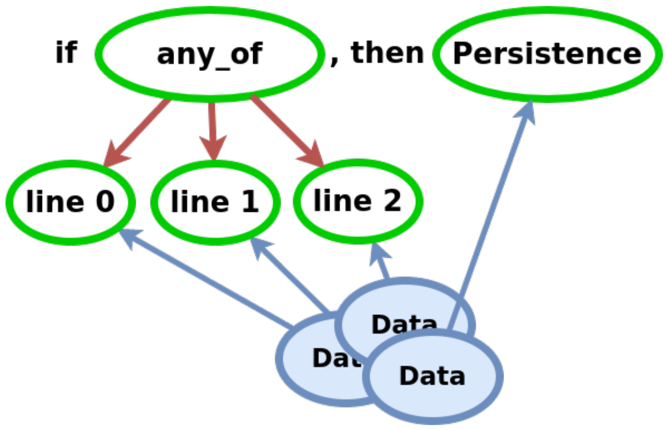


Figure 6.3: The trigger, from not quite so far away. The red arrows indicate control flow dependency.

When zooming in even further, we will see that lines themselves are made up of smaller elements of boolean decision, see Figure 6.4. It becomes apparent that a tree representation of control flow for the trigger is a suitable choice. The trigger decision is the first example of a higher level tree node composed of and exclusively defined by the boolean results of all trigger lines. We will call these nodes ”Composite Nodes”. They combine decisions made by their child nodes. The nodes filled green in Figure 6.4 correspond to ”Basic Nodes”, they represent leafs in the control flow tree. These basic nodes make decisions from reconstructed data by invoking decision-making algorithms and their data dependencies. A trigger line designed to select $B_s^0 \to \mu^+\mu^-$ could define its control flow as follows: First, check if there are muons that pass some quality and displacement requirements. If there are any, see if combinations of these can build vertices with reasonable resolution. Only if both nodes yielded some candidates should the trigger line tag the event positively.

To be able to define any control flow, composite nodes need to be able to implement several types of combination logic. We choose *lazy-and*, *lazy-or*, *eager-and*, *eager-or* and *not*. Lazy logic corresponds to short-circuiting execution outlined in Section 6.1. Eager logic schedules all children unconditionally. ”not” corresponds to boolean inversion, and requires exactly one child.

We further define optional notion of order. Composite nodes may or may not force children evaluation order. A forced order of evaluation is interpreted transitively. Two composite nodes that act as children of an ordered mother imply pairwise precedence rules on all of their children and grandchildren.

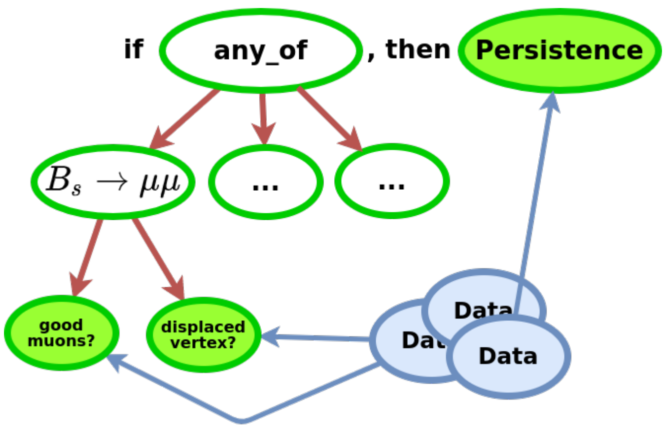


Figure 6.4: The trigger, a closer view.

We can thus summarize our two types of nodes:

- **A basic node** governs one algorithm that makes a control flow decision and the data dependencies for this algorithm.
- **A composite node** defines an execution logic and whether it forces evaluation order on its children, which can be either basic or composite nodes themselves.

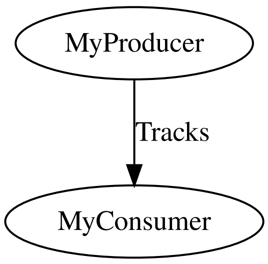
With the control flow design established, the next task is the representation of data flow.

6.3.4 Representation of data flow

Data flow in LHCb is implemented through the TES interaction. Every event interacts with a TES instance that is local to that event. This is crucial for practical event independence as we do not have to consider multi-threading synchronizations in the data flow. Whenever an algorithm produces data, ownership over that data is transferred to the TES, and they remain there for the processing time of that event. TES access is done via string key lookup. Algorithms in LHCb and GAUDI define these keys as configurable via python. Data flow is expressed as edge of the directed data flow graph by matching strings in output of one algorithm and input to the other:

```
track_location = "/Event/ProducedTracks"

a = MyProducer(OutputTracks = track_location)
b = MyConsumer(InputTracks = track_location)
```



The implementation of the pre-upgrade trigger did not allow for automation of data flow resolution, as TES interaction happened implicitly within the execution of every algorithm and there was no syntactical difference between properties that define data flow and miscellaneous parameters. This was changed as part of the general overhaul of the Gaudi framework to fit future needs.

Thus, implicit data flow is turned explicit via the use of so called ”Datahandles”. Datahandles serve multiple purposes, one of which is that they are distinguishable as special properties concerning the definition of data flow. They also provide information of the direction of dataflow, whether they represent output or input. It is possible to inspect algorithms for their participation in data consumption and production.

By doing so, data dependencies can automatically be deduced for every decision that a trigger selection author wants to use. By matching Datahandle Input to Output, an ordered collection of data dependencies can be built for every algorithm. The order is such that for every data flow edge between two algorithms, the data producer appears before the consumer in the collection.

With these ingredients at hand, the initialization and the runtime step of the scheduler implementation can be defined.

6.4 Event loop preparation - Initialization

One of the biggest differences in this scheduler implementation with respect to the Avalanche scheduler is that the order of execution is fully defined before running over events. This is a key feature to achieve low runtime overhead.

Firstly, the control flow tree needs to be defined in the python interface. The following produces an example control flow that runs two algorithms in a short-circuiting fashion within a logical conjunction:

```
bn1 = BasicNode("BN1", decision_maker1)
bn2 = BasicNode("BN2", decision_maker2)

root = LAZY_AND("root", children=[bn1, bn2], ordered=True)

register_cf_tree(root)
```

The root of the tree is registered and traversed through the children to gather all nodes that one can reach from root. This tree is translated to C++ via the configurable interface. From that information the algorithm creates C++ instances of all the nodes and save them with fixed positions in memory. Data dependencies in the basic nodes are filled by matching the Datahandles as discussed in the previous section. Both composite and basic nodes receive pointers to their children and parents to be able to traverse the tree upwards and downwards. For all ordered composite nodes, control flow precedence rules are built pairwise between all children, represented as control flow edges. A typical two-line configuration may look as depicted in Figure 6.5. The red arrows represent control flow precedence, generated from the ordered mother of the basic nodes.

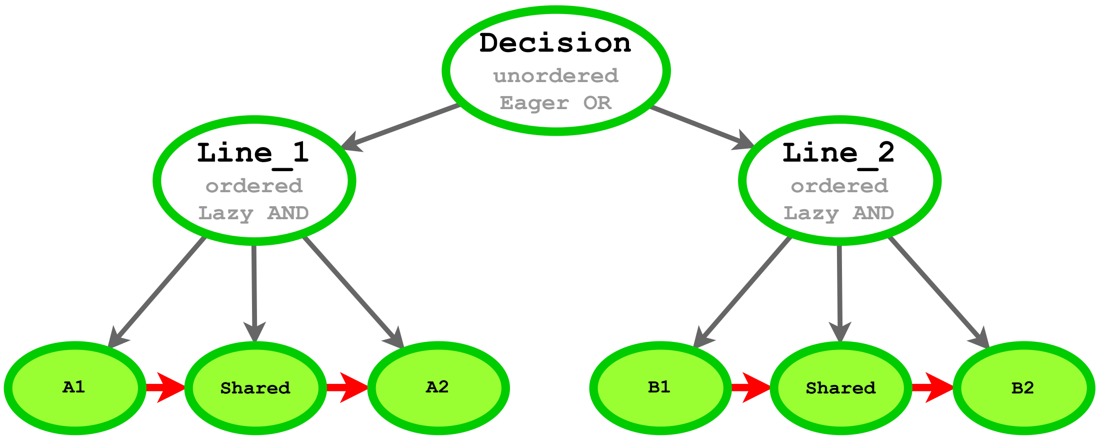


Figure 6.5: The control flow tree of a minimal two line configuration, with one shared basic node. Red arrows represent control flow edges.

Next, a valid order of execution to use at runtime is produced. All basic nodes are gathered into a set and then ordered to meet all precedence rules as defined by the composite nodes. This is done by continuously looping over the initial set and moving nodes into an ordered collection whenever all nodes that precede this node are already part of the ordered collection. This way control flow precedence circles can be detected. A full loop through the set without any move operation results in error.

We end up with an ordered collection of basic nodes with their data dependencies as can be seen in Figure 6.6.

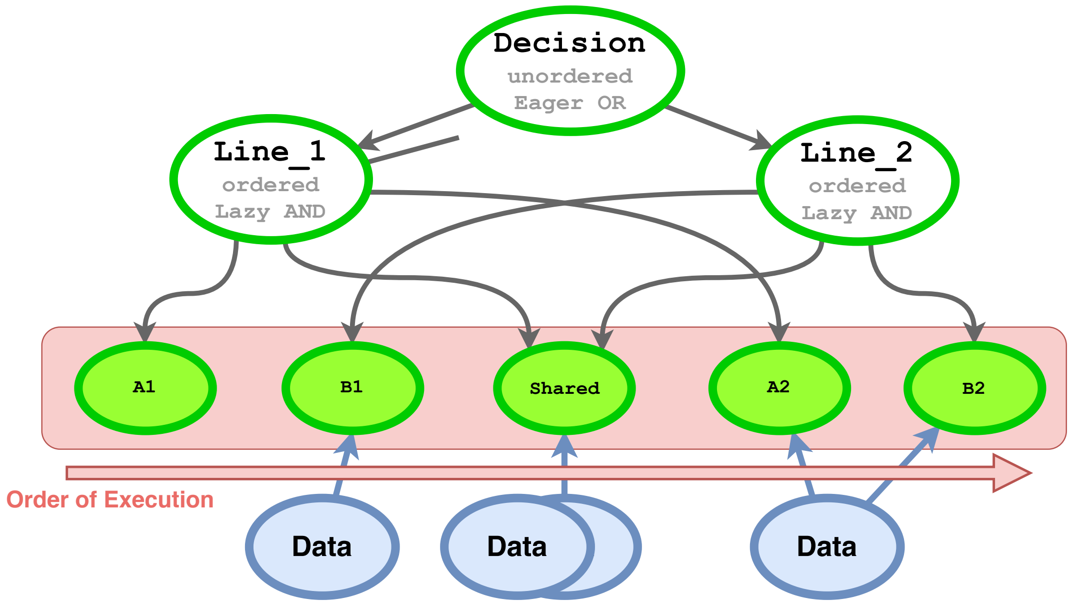


Figure 6.6: The control flow tree with the basic nodes ordered as they will be executed during runtime.

During runtime, the algorithm needs to maintain the state of every control flow node to ensure correct execution. Specifically, each node's state is specified by one integer and one boolean. With the boolean we track the control flow decision that every node evaluates, either from the underlying algorithm or from the logical combination of other nodes. The execution counter will be further explained in the next section. Notably, the node states are not tracked as members of node instances. This would require a copy of the control flow tree instance for each event in flight. Therefore, node states are represented as part of an ordered collection, and each node is assigned an index in the collection. It is copied into each event task to enable independent processing.

Although not strictly required, the same trick is performed for algorithm execution states. At any time during the event, one can inspect algorithms for their control flow decision and their execution state by querying an algorithm state service that holds thread local variables to track the states. This is a feature needed for the avalanche scheduler, but thread-local access is unnecessary overhead in this case, as the scheduler can work on local instances.

Finally, a user defined number of threads is initialized. This collection is often called thread pool. These threads take care of processing events.

6.5 The event loop - Runtime

6.5.1 Task packaging

The event loop is the performance crucial part of the scheduler, as everything that will be calculated within the loop needs to happen 30 million times per second in HLT1 and 1 million times per second in HLT2.
One thread, the master thread, takes care of the actual loop over incoming events. It prepares tasks which represent the processing of one event each. Every event task receives a context with which it finds metadata about the event. The context also carries a fresh copy of the node state vector that has been prepared in the initialization. Event tasks are then pushed into a queue.
Meanwhile, idle workers within the previously initialized thread pool wait for incoming event tasks. When one arrives, any idle worker is notified, it takes the task from the queue and immediately starts processing. If there is no free worker, the first one that finished processing directly takes up the next task. As soon as a task is done, the worker notifies the master thread about the finished event. This notification enables cooperative scheduling. Instead of filling the worker queue indefinitely, the user can define the number of events in flight. That is the maximum number of unfinished event tasks at any given time. Unconditional filling would consume more memory than necessary. By just slightly overcommitting the queue, idle workers are avoided. On the test node with 20 hyper-threads, the speed optimum is reached with 20 workers and about 24 events in the queue.

6.5.2 Sequence execution

Within the event task, execution of the ordered sequence of basic nodes may now begin. To explain the algorithm, we define nomenclature:

1. A node is ”retired” as soon as its decision is evaluated. For lazy nodes, this happens as soon as the decision of their children allows it. One negative

(positive) decision in a *lazy-and* causes retirement. For eager nodes, retiring happens only after all children are retired themselves.

2. A node is ”requested” if there is at least one path through the tree from this node to the root where none of the parents in that path are retired.

The basic node sequence is looped over in the following manner:

1. Is the node requested? If not, continue with the next node
2. Every data dependency in the list assigned to the node is executed if that has not happened before as part of another basic node. The same is done for the decision maker algorithm. Note that it is possible that a decision maker is a data dependency of another node.
3. The control flow decision of the decision maker is propagated to all parents of the node.
4. All parents update their own state depending on their composite node type. If one parent is retired, this information is propagated to its parents in turn.

Every request for state goes into the local state collection that has been copied into the context for the event task. So if the execution state of a node is needed, the node provides the index to look for in the state collection. The same goes for algorithm states.

The outlined method of execution guarantees minimal work, and the introduction of the scheduler into the LHCb software stack has immediately revealed unnecessary steps in the reconstruction, because data was produced that was not consumed anywhere.

6.6 The control flow barrier - Sharing work

In the LHCb track and vertex reconstruction the very same task is sometimes performed on different selections of tracks or vertices. These selections are not necessarily disjoint, in which case duplicate work is done on the intersection. We can explore different methods of avoiding duplicated work. One method could be an upfront reconstruction that considers all tracks or all vertices. This may save computation in case the intersection is bigger than the amount of data you additionally consider when not selecting previously.

Here another method is considered: A work sharing barrier. Its job is to gather the union of all selections to perform the computationally intensive task on, then execute the task only once and afterwards scatter the data back into multiple selections accordingly. A sketch of the barrier workflow is depicted in Figure 6.7. Columns represent independent data flows merged at the gather step. In this example, the first data flow stopped early, as for instance control flow did not require further execution of the corresponding line. In order to handle these cases correctly, the barrier requires the introduction of optional data dependencies. The gather step needs to work regardless of how many data flows make it to that point. Only if all executions terminate early, the gathering step should not be invoked.

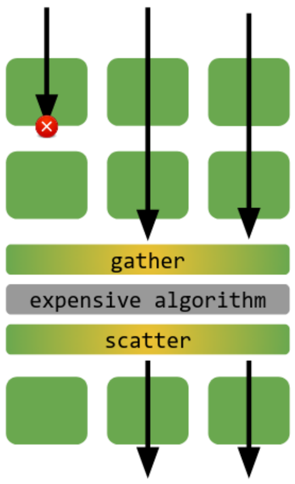


Figure 6.7: The gathering concept. Black arrows indicate dataflow. The gather algorithm only considers inputs of active dataflows (second and third column from the left).

The way this scheduler handles data dependencies and control flow precedence allows such a barrier by introducing additional precedence rules and strategically placing the gathering algorithm. A working barrier configuration is shown in Figure 6.8.

Firstly, optional data dependencies are modelled by never explicitly considering data dependencies of the gather algorithm. When data dependency lists are build for basic nodes, the gather algorithm is considered as pure producer. That way one avoids accidentally invoking a selection algorithm that only serves an already retired line, like the first column in Figure 6.7. In the example in Figure 6.8, the gray dotted arrows depict optional dependencies.

Secondly, explicit pairwise precedence rules between the inputs to the gatherer and the receivers of the barrier output data need to be introduced. Otherwise, one line might be scheduled before the other, and the expensive algorithm runs before all inputs have been produced. These precedence rules are the reason this is referred to as a barrier concept. All inputs to the barrier need to have completed or retired beforehand before the gathering can begin. In other words, a synchronization step is introduced. The example in Figure 6.8 introduces four precedence rules, pairwise between Selection 1/2 and Filter 1/2.

All inputs to the gatherer need to be scheduled explicitly as decision maker of a basic node as opposed to being executed as part of a data dependency resolution. There are two reasons for that:

Firstly, explicit precedence is part of control flow which is modelled via edges between tree nodes.
Secondly, these inputs mostly serve the gatherer exclusively. However, inputs to the gatherer are not considered as part of the data dependency chain due to their optional nature. Thus, unless explicitly requested as part of a lines control flow, these inputs would not run at all. With the proposed setup, they are explicitly requested as long as the parent line has not yet retired. Notably, since these data producers now participate in control flow as decision makers, they simply yield an unconditionally positive decision.
Lastly, our barrier should not be part of the explicit control flow, but only play a role as data dependency of any one of its successors in the data flow. If all lines retired before reaching any consumer of the output data the expensive algorithm provides, invocation of the barrier would be unnecessary work.

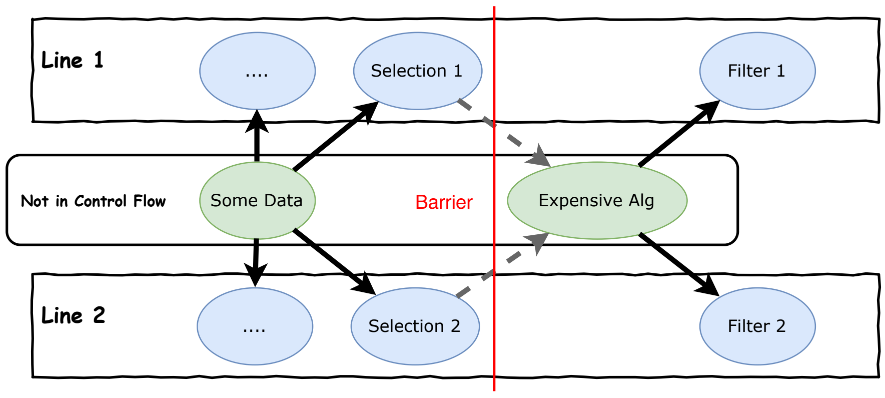


Figure 6.8: The barrier concept built into the work flow configuration. Black arrows represent data dependency, while the gray dotted arrows denote optional nature. Blue blobs represent basic nodes and green blobs show data producers.

6.7 Scheduler performance

Performance measurements were performed for multiple scenarios to ensure sufficient computational efficiency even in complex scenarios.

The first test involves the execution of the HLT1 reconstruction chain. As it represents a linear sequence, it can be scheduled explicitly with a minimal logic scheduler implemented in Gaudi. This scheduler invokes a full event stop whenever a negative control flow decision is encountered. No throughput difference could be observed between both schedulers after averaging several runs, meaning that the logic introduced in the new implementation causes negligible overhead in this case. This is not a representative stress test however, as the HLT1 reconstruction only involves linear control flow with about 15 algorithms.

For following tests, the Intel V-Tune Amplifier [73] is employed to estimate function execution times in multi-threaded environments without significant overhead to the application. Fully fledged HLT environments are mocked with algorithms of configurable busy execution time.

When calibrating the application to the nominal HLT1 throughput requirement of 30 kHz with 10 lines and 5 nodes per line on 40 threads, the schedulers relative consumed CPU time stays well under 0.5 %. Doubling the number of lines at the same throughput level brings the contribution to execution time up to nearly 1 %. However, other framework overheads surpass the scheduling overhead.

A HLT2 like scenario is also tested to ensure scalability of the implementation. 500 lines with 10 nodes each are calibrated to a throughput of about 1 kHz. The relative time contribution stays well under 2 % here. Unfortunately, other framework overheads scaling badly with increasing number of algorithms slow down the application much more. Further framework bottlenecks need to be investigated until we are ready for a 1 MHz HLT2 application.

6.8 Summary and outlook

I have presented the first production ready scheduling algorithm that meets the feature requirements for the upgrade HLT. All in all, the undertaken tests show promising results that runtime overhead is sufficiently small for use in the online HLT environment. As of this day, it is the baseline scheduler used in all CPU upgrade configurations. The GPU implementation for HLT1 in Allen is currently adopting an implementation that has been derived from this one. It enables the option of packaging multiple events into one task.

The only real test for the overhead of the scheduler implementation will be the full HLT2, once it is fully implemented. Until then, there are some changes that can marginally speed up the scheduling time.

In the HLT many selection algorithms have a long tail of data dependencies that are similar for many nodes, invoking the full track reconstruction to get the particle inputs. Even though these algorithms are not actually run multiple times per event, there are still lots of unnecessary if-statements to go through. One can therefore explicitly schedule these common reconstruction steps upfront and invoke the following optimization: Whenever a basic node dominates another, all shared data dependencies can be removed from the latter. ”A dominates B” means that execution of B implies previous execution of A.

Secondly, most of the explicit scheduling time is spent in checking whether a basic node is requested, an operation that has to go up the control flow tree. Conceptually, an invocation condition in dependence of other basic nodes could be deduced at configuration time already. This would effectively avoid tree traversal for that particular operation.

7 Selections and Combinatorics in Upgrade HLT2

After having written and optimized parts of the underlying application framework for the upgrade trigger in the last chapter, this chapter focusses on two specific algorithms of HLT2 itself, where significant improvements have to be achieved in order to get the desired runtime efficiency. After an introduction to the use case for these algorithms in Section 7.1, the performance of status quo is discussed in Section 7.2. Section 7.3 discusses these algorithms in detail. The following sections discuss several solutions to optimize the algorithms in question. Lastly, a summary on the current state and an overview on future work are given. Note that the work presented in this chapter is the outcome of shared efforts between Olli Lupton and me. Several other people in the LHCb collaboration helped forming the ideas.

7.1 Selection algorithms in the trigger workflow

The outlined HLT1 workflow in Figure 4.4 is conceptually similar to HLT2. As most trigger lines require a common set of reconstruction algorithms, control flow can be slightly simplified by invoking an unconditional upfront reconstruction step. Although the specifics of the reconstruction in HLT1 and HLT2 are different, both trigger stages have one thing in common. After upfront reconstruction, most of the leftover work is done by selection algorithms, represented as the "selection criteria" in Figure 4.4.

In HLT2 one mainly differentiates between two types of selecting algorithms. The first type is the plain selection that takes a collection of particles and returns a selected collection. These are called "Filters". An example of a Filter is one that collects muons with a high transverse momentum from all the input particles.

Algorithms of the second type are called "Combiners". These algorithms combine two or more particles to form a candidate for a decaying particle. Although combiners reconstruct vertices, they will be referred to as selection rather than reconstruction algorithms, because selection criteria with control flow impact are applied at multiple points during the combination process.

Many Combiner and Filter instances in HLT2 apply tight requirements such that most invocations end up with an empty output collection, meaning that no particle passed all selection criteria. Control flow decisions for the trigger are based on whether at least one particle passed the algorithm. The typical workflow of a trigger line selecting $B^0 \to \pi^- K^+$ can be seen in Figure 7.1. It is assumed that charged final state particles are already reconstructed and classified via PID information. To build $B^0 \to \pi^- K^+$, one first needs pions and kaons. Because they should come from a beauty decay, one may filter based on displacement from the primary vertices and a high transverse momentum. This is what the Filters are designed for.

The following workflow is comprised within the implementation of Combiners. If there are pions and kaons matching the criteria, the next part of the workflow can be invoked. The selected final state particles can be combined and selections can be made on these combinations. Examples would be the distance of closest approach between both particles. If there are good combinations, a decay vertex for the B^0 candidate can be fitted. If this vertex passes some more requirements, a B^0 candidate is selected for final persistence. The decision of the Combiner depends on whether any candidate passed all previous criteria.

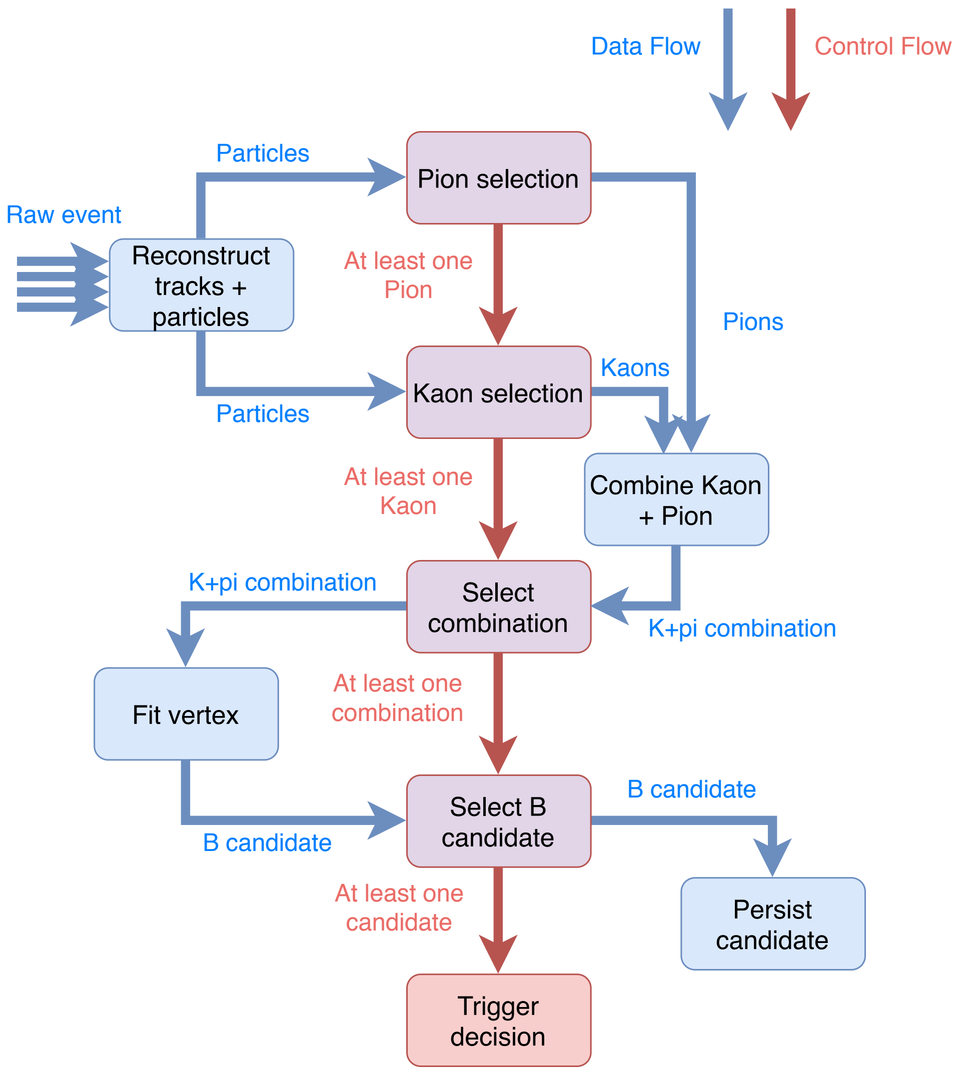


Figure 7.1: The reconstruction workflow in a trigger line selecting $B^0 \rightarrow \pi^- K^+$. Blue arrows mark data flow and red arrows correspond to control flow.

7.2 Runtime performance in status quo

Significant throughput improvements have to be achieved to reach the throughput goal of 1 MHz in HLT2. Exact numbers are hard to come by, as only about 300 out of an estimated O(1000) trigger lines are currently available in upgrade software. Current benchmarks with the existing lines show that HLT2 throughput is off by at least a factor of 2. The Combiners and Filters this chapter focuses on contribute about 25% to the overall runtime.

The benchmark in Figure 7.2 compares this to the Run 2 HLT2 implementation on HLT1 filtered upgrade simulation. On a single core of an Intel Xeon E5-2630-v4 CPU, this configuration processes events at about 2 Hz. Filters and Combiners together use about 23% of the total runtime, and this configuration only consists of about 450 lines as opposed to the O(1000) lines expected for the Upgrade.

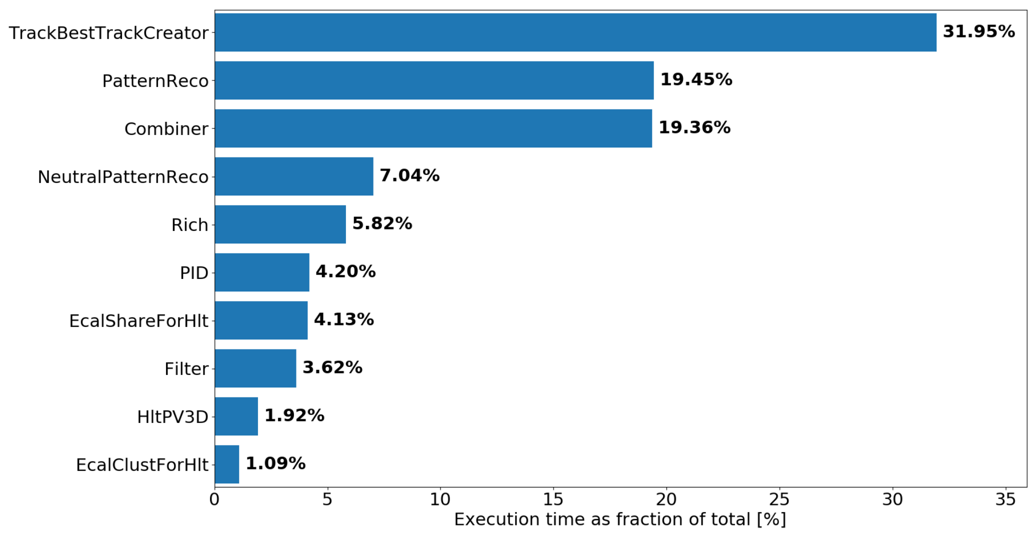


Figure 7.2: Relative execution times of the top 10 algorithms in the Run 2 HLT2 on upgrade HLT1 filtered simulation. The Combiner and Filter bars sum all combiner and filter instances. All other algorithms refer to parts of the reconstruction. Specifically, the TrackBestTrackCreator is the algorithm performing the track fit.

7.3 The baseline algorithms

The baseline Filters and Combiners in HLT2 are called "FilterDesktop" and "CombineParticles". Both act on collections of particles as inputs. FilterDesktop returns a selected subset of the input particles, while CombineParticles creates new candidates from the combination of the inputs.
As these are very generic algorithms, several aspects are configurable through the python interface, like the decay to be reconstructed in the combiner and all selection criteria.

7.3.1 Filtering with LoKi

Being able to define generic selection criteria from a python interface to the C++ code is a non-trivial task. In the early days of LHCb, criteria would still be hard coded into the algorithms, with configurable parameters only. It was then possible to change the minimal transverse momentum threshold from the configuration, but not the fact that it is a minimum transverse momentum cut without editing C++ code. A dramatic improvement to usability came with the introduction of the LoKi (Loops & Kinematics) framework [74], which provided the implementation of so called LoKi-Functors. These allow the definition of almost arbitrary selection code via specification though a python string or directly within a GAUDI algorithm. With LoKi, selection code can be defined as follows, from python:

```
kaons = reconstruct_kaons()
MyKaonSelector = FilterDesktop(
                    input=kaons,
                    Code="(PT > 1*GeV) & (IP > 1*mm)"
                 )
```

A python interpreter will interpret the `Code` string at initialization time of the application. `PT` and `IP` are python aliases for C++ classes that are accessible from python via the cppyy interface [75]. Different kinds of unary and binary operations on these objects are overloaded to yield composed LoKi functor types. For instance, `PT` > `5*GeV` constructs a class with a call operator returning a boolean, checking whether the input has a transverse momentum of more than 5 GeV. Compositions via mathematical operations like +,−,∗,/, boolean compositions &,|,~ and more are supported in the LoKi framework. Functors can be transformed to a C++ representation of themselves that is accessed from the filter and combiner algorithms to perform the desired selection. It is thus possible to create arbitrary selection predicates without any additional compilation, sufficient for almost all use cases in the LHCb selection framework.

7.3.2 Basics of combining

The process of combining particles to a common parent has been introduced with the $B^0 \to \pi^- K^+$ example in Section 7.1. This section will go into more technical detail, starting with a summary of the workflow:

1. Take m input particle collections, where m is the number of different particle types.
2. Select the children collections with reasonable criteria for the given decay.
3. Build a n-body combination from these particles, where n is the number of children of the parent to be reconstructed. This is a $O(k^n)$ scaling operation.
4. Select combinations with criteria to match a good candidate. These criteria do not only involve user-specified predicates based on the decay, but also a pairwise overlap check. When two particles share underlying components like tracks, a combination can not yield a valid signal candidate, so these are preemptively discarded.
5. Fit the combination to yield a decay vertex position and a parent momentum with respective uncertainties.
6. Select good vertices and label them according to the decay requirement.

This workflow is performed in almost every HLT2 line in LHCb, often multiple times per line execution. It is notable that the number of operations involved scale exponentially with the number of children involved, so critical path efficiency is crucial for a successful online application. Efficient evaluation of selection criteria becomes very important as these have to be applied to every combination.

7.3.3 Workflow in the baseline combiner

CombineParticles has worked as the default combiner algorithm throughout all operational years of LHCb. It is based on the Run 2 HLT2 particle implementation: LHCb::Particle. LHCb::Particle is a structure that holds multiple properties like particle ID information and possibly a decay vertex or a track. As the class is not specific to the type of particle, a charged final state particle has a nullpointer for the decay vertex and a composite particle has a nullpointer for the track. The design of LHCb::Particle is discussed in more detail in Section 7.5.1. CombineParticles can receive either selections or vectors of particles to be agnostic on whether a previous selection has been applied to the collection. The algorithm is subdivided into three stages: Child-, Combination- and Vertexing-stage.

At first, the algorithm parses a decay descriptor. This is a user specified string that encodes the decay to be reconstructed by the combiner. An example decay descriptor is given here:

```
"[B_s0 -> D_s0 pi+ pi-]CC"
```

It encodes several important pieces of information for the combiner: The number of children (3), the particle id of combined particle (B_s^0), the children particle ids and their charge configuration. The `CC` serves as shortcut for charge conjugation. Wrapping the decay descriptor with brackets and `CC` means that the charge conjugated decay is also to be reconstructed in the same combiner invocation. CombineParticles generally allows for an arbitrary long list of decay descriptors to be worked off at the same time. Reconstructed parent particles from all decay descriptors will later be put into a common collection to be stored on the TES. Decay descriptor parsing and LoKi cut generation are the two main things that happen before or during initialization, they do not contribute to runtime costs in the event loop.
Selection criteria logic for cuts on input particles works as follows: The user provides a map of particle ID (PID) to criterion. When there is no entry for a PID that occurs in a decay descriptor, the criterion will fall back to either the criterion of the charge conjugated PID, or if neither exist, to no selection criterion at all.

A user can give an arbitrarily long list of input particle collections to the combiner. In the child stage, input particles are merged into a single container per PID. According to the decay descriptors and the child cut dictionary, relevant particles will be selected for consideration in the combination stage.

The combination and vertexing stages happen within a loop over all decay descriptors. A n-dimensional loop over the relevant input particles given the current descriptor is built. It corresponds to the cartesian product of the input vectors. The first check on each combination is for uniqueness. It is trivial for combinations with disjoint PIDs because particles with different PIDs are per definition different. However, it is important for any decay descriptor in which one PID occurs twice, like `'B+ -> pi+ pi+ pi-'`, where one does not want to consider both (`pi+`[x], `pi+`[y], `pi−`[z]) and (`pi+`[y], `pi+`[x], `pi−`[z]). x,y,z represent indices into the pion containers. The uniqueness check asserts that only combinations are considered in which particles with the same PID are ordered in some way. The order criteria comprise kinematic and pointer comparisons. The result for selected combinations is depicted in Figure 7.3. Note that checks for unique entries in a collection usually scale $O(n^2)$ with the number of elements, while this order requirement can be applied with $O(n)$ scaling. Producing a complete collection of unique combinations with the order requirement can only work correctly if the containers of which the combinations are made are exactly the same. This is the reason for the merge of all inputs according to PID in the child stage. The combiner functionality is therefore restricted to one cut per PID, not one per input collection.

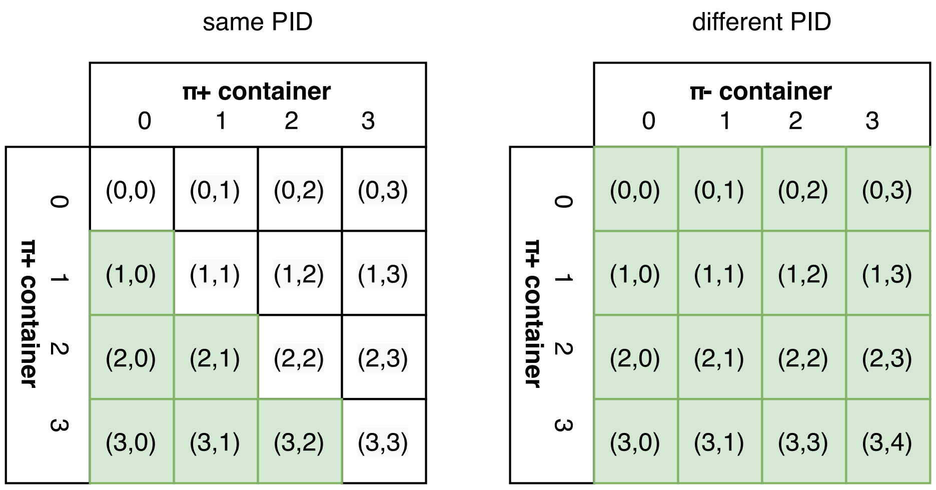


Figure 7.3: Loop structure for combinations of containers with either same (left) or different (right) PID in a 2D subspace. Assuming the containers are ordered according to the aforementioned kinematic criteria, green fields represents considered combinations.

Next, all combinations with an overlap are filtered out. There are multiple algorithms that define what overlap means, but conceptually one tries to identify whether multiple particles in the combination have been made from the same track or with the same calorimeter clusters. Nothing stops two algorithms from defining two different particles from the same track by assigning two different particle IDs. No track or cluster information in the detector should go into the reconstruction of two disjoint particles.

The algorithm proceeds to apply the combination cut, one of previously defined selection criteria. In HLT2, trigger line authors often require a small distance of closest approach (DOCA) between the combined particles or that at least one input particle had a high transverse momentum. The application of selections on the particle combination serves the purpose of reducing combinatorial load on the subsequent vertex fit.

The last stage in CombineParticles is the vertexing stage. The few particle combinations that have made it through all previous checks and selections are now fitted into a common intersection point, or vertex. Out of a successful vertex fit comes a vertex position and a covariance and a fit χ^2, which is then associated to a new particle, the parent candidate for all children in the combination. Its PID is set to the parent PID in the decay descriptor. These candidates are then selected again with a user specified criterion, most often based on the vertex χ^2, invariant mass and vertex position. Particles passing the entire selection chain are persisted into the TES for further studies.

Next to CombineParticles, there is another set of algorithms called ”NBodyDecays”. These algorithms are, unlike CombineParticles, specialized on the number of children to be combined. They enable another set of user specified criteria that we will call sub-combination criteria in this chapter. When combining N children, cuts on the [2..N-1] combinations may also be specified to reduce the combinatorial phase space early on. A simplified three body combination loop might look as follows:

```
for p0 in particle_container_0:
  for p1 in particle_container_1:
    # do not even enter the third loop if
    # the two body combination is ”bad”
    if not two_body_cut(p0, p1):
      continue
    for p2 in particle_container_2:
      if three_body_cut(p0, p1, p2):
        do_vertex_fit_and_cut(p0,p1,p2)
```

7.4 Improving upon the baseline with new selections

The first version of LoKi has been introduced as the framework for composable and configurable selections over a decade ago. The functional, multi-threading framework is however ill-suited for the use of many LoKi features that rely heavily on internal state. As many people want and require configurable selections, a rewrite of major parts of the LoKi framework would be necessary for use in the upgrade HLT. However, benchmarks within HLT1 selections have uncovered that LoKi selections often are the bottleneck of the application.

We therefore decided that developing a completely new selection framework is more sensible. This framework, dubbed ThOr (Throughput Oriented) is being designed with different levels of parallelism in mind. It is fully functional to ease multi-threading, and its type system makes the use of vectorization within selections easier than LoKi does.

While ThOr aims to provide similar functionality, its C++ backend is different to LoKi's. LoKi works with cppyy's python bindings and it resorts to type erasure on multiple levels to be able to compose functors at runtime without any additional compilation. While this is a very flexible approach, it comes with a non-negligible runtime overhead. Every composition adds more costly runtime polymorphism.

The backend of ThOr on the other hand does not try to avoid additional compilation. With the cling compiler [76] that has been developed after the introduction of LoKi, Just In Time (JIT) compilation is available in the LHCb software stack natively. Rather than stacking layers of costly polymorphism, ThOrs functors can get JIT compiled and linked at the initialization time of the HLT application. There is only one layer of type erasure that enables the use of ThOr functors in any GAUDI algorithm.

Cling is a JIT compiler currently based on Clang 5. Compared to the currently used compilers in LHCb software, Clang 9 and GCC 9/10, Clang 5 is fairly old and does not support the newest C++ standards. This fact in itself poses challenges due to incompatibility, but that is not the only problem. Code compiled with cling often executes many factors slower than code compiled with modern gcc or clang versions, the reason for which is currently under investigation. Moreover, changing compiler flags to enable different levels of vectorization is highly non-trivial in cling as it uses pre-compiled headers of ROOT libraries that it depends on. JIT compilation of ThOr criteria is thus done exclusively on a scalar basis, i.e. no vectorization enabled.

In summary, JIT compilation with cling is currently not efficient for high throughput scenarios, but it works fast enough for the offline use without the need to recompile the full software. For the high throughput production application, ThOr employs a second mode of compilation: The functor cache.

All functors that are defined in python during build and compile time will be compiled into a cache to be used natively in the application. The usage of these functors does not need any JIT compilation.

Note that LoKi is also capable of using a functor cache, but the performance difference is much less pronounced than with ThOr. This has two reasons: Firstly, the slowdown that Cling introduces with its JIT compilation only applies to ThOr, as LoKi does not necessarily need to compile anything. Secondly, the extensive amount of type erasure that enables composability without compilation is still present in compiled functors. Figure 7.4 summarizes the possibilities to build configurable selections and their runtime qualitatively. The speed differences have not been quantified for the two slower options as they can be avoided during production. Detailed comparisons between compiled LoKi and ThOr are found later on in this Chapter.

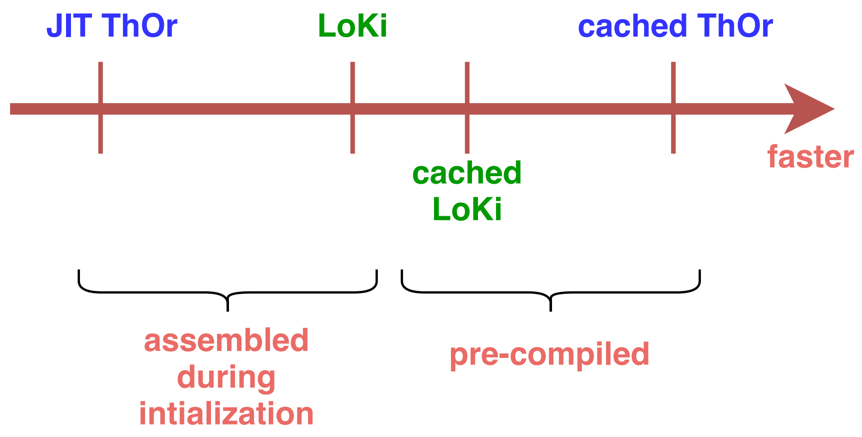


Figure 7.4: Functor implementations sorted by speed (single-core only, qualitatively).

ThOr implementations of functors are already put to practice in HLT1 selections and yield good results there. For the first tests in HLT2, ThOr functors are implemented to work with LHCb::Particle.

Generally, ThOr functors are designed to be agnostic to input and output type to be flexible on what they operate on. They only assume the existence of certain accessors needed to compute the desired result. This approach usually referred to as duck typing. `instance.momentum()` or `instance.pid()` are basic examples. Exceptions are sometimes made for more mature models like LHCb::Particle as API changes there are much more convoluted and would require changes throughout the entire downstream codebase. In this case, the functor input type is detected at compile time and another code branch is executed.

Comparisons of runtime in this chapter are all single core tests, as LoKi is not capable of multi-threading. First order Linear scaling in the number of physical cores is expected, as there is no inter-event dependency in any of these algorithms.

The baseline filtering algorithm FilterDesktop using LoKi is compared to an equivalent filtering algorithm using ThOr called PrFilter. PrFilter is a generic filtering algorithm which can be specialized on the type of input container. For this comparison it is specialized on a non-owning masked collection of LHCb::Particles. Benchmarks of ThOr, LoKi, combiner and filter algorithms depend on the configuration that is tested. Therefore, performance numbers are not to be considered representative for the overall HLT application, although the benchmarked selections closely resemble actual existing HLT2 selections. Note that all benchmarks are performed with the fastest configurations which are not necessarily the default.

Table 7.1 shows two benchmarks for each filter type. These and all following benchmarks in this chapter are performed over 10000 simulated events that resemble actual production data as closely as possible. These simulations are referred to as ”minimum bias” in LHCb. Charged basic particles are filtered according to two different criteria. The first number corresponds to a selection with only one functor, `ISMUON`. This functor simply dereferences some pointers to check whether a particle has information from the muon stations. This simple comparison is helpful to identify framework performance differences. Since many of more complicated functors are implemented differently, the throughput differences in these originate not only from choosing between ThOr and LoKi, but also by functors themselves. The other criterion is fairly realistic preselection to many beauty and charm trigger lines that selects high momentum displaced tracks. Specifically, this preselection selects based on momentum, transverse momentum, χ^2(IP) and PID. The speed up in this case can not only be attributed to the increased framework efficiency. Specifically the calculation of χ^2(IP) differs. The ThOr implementation is explicitly specialized and optimized for charged basic particles while the LoKi employs a more general, but also more costly algorithm making up about 60% of the 171 µs.

	ISMUON	displaced presel.
FilterDesktop	11.6 µs	171 µs
PrFilter	4.6 µs	12.8 µs

Table 7.1: A benchmark comparing two selections with two different algorithms. Both Filters work without data copy, only with indices and pointers. PrFilter employs functors implemented in the ThOr framework.

Overall, there are many places throughout LoKi functor implementations where performance is sacrificed for generality, convenience or not strictly necessary features. For instance, several frequently visited branches in the ISMUON calculation yield atomic increments of logging counters. In the benchmarked scenario where one selects muons from all long tracks, counter increments happen about 10 % of the time. If not explicitly disabled, these increments drive the execution time from 12 µs to 45 µs.

7.4.1 Combining with ThOr

It is now established that ThOr functor implementations work well in filter algorithms and yield significant throughput improvements, aided by the increased efficiency of the ThOr framework itself. The next step is thus to try to apply ThOr in the other selection algorithms, the combiners. The implementation presented here is the first multi-threading capable combiner, implemented in the GAUDI functional framework. Combiners make up a major portion of the HLT2 runtime and therefore represent a crucial component for optimization.

The algorithm described here is referred to as ThOrParticleCombiner and its differences to CombineParticles are mainly framework based. It is a Merger as described in 5.4. It gathers over multiple particle input containers, combines particles from these according to a decay descriptor and then creates parent particles in a vector collection. ThOr functor specializations are used for selection. This combiner specializes on the number of children at compile time, similarly to NBodyDecays. Up to the functor implementations, the children selections are implemented in the same way as the LoKi combiners. The differences are to find in the combination loop:

1. NBodyDecays allows for N-1 combination criteria for an N-Body combination, concerning the first M children, where M goes from 2 to N. A 4-Body combination allows for combination cuts on children (12), (123), (1234). The new algorithm allows also for combination cuts on children (13) and (14). This helps with reducing the combinatorial load even more than in NBodyDecays. Other combinations of criteria, like (23), make less sense to implement since they would have to be evaluated as often as the (123) combination due to the loop order. One could hoist the evaluation out of the loop by preemptively applying the (23) criterion to the full input collections and save the result in a 2-dimensional structure, but as combination cuts are generally expected to have a rather small efficiency, only little information in the 2d structure would actually be used.

2. The information on whether there are multiple instances of the same PID in a decay descriptor is only available at runtime. However, the PID check is performed in LoKi combiners for every combination. The new combiner lifts this information out of all loops and thus reduce the branching in critical paths.

The overlap check during combination is done with the same tool as in the LoKi combiners. The vertexing stage also employs the same logic as LoKi combiners, which is simply a filter step. Table 7.2 shows how CombineParticles, NBodyDecays and ThorParticleCombiner compare in runtime when using a common $D^+ \to K^+\pi^+\pi^-$ selection with similar, but looser selections than in the Run 2 HLT2 $D^+ \to K^+\pi^+\pi^-$ trigger line. The combinations are made from kaons and pions selected with the previously benchmarked displaced filtering criteria from Table 7.1. The combination selection involves minimum transverse momentum, distance of closest approach (DOCA), invariant mass, summed children momentum, and several more displacement requirements. After the vertex fit, selection criteria are based on vertex quality, parent flight distance and direction angle. The big performance gap between CombineParticles and NBodyDecays is explainable by the fact that CombineParticles does not allow for sub-combination cuts and thus has to evaluate many more combinations than the other algorithms.

	$D^+ \to K^+\pi^+\pi^-$ execution time
CombineParticles	256 µs
NBodyDecays	77.1 µs
ThorParticleCombiner	38.8 µs

Table 7.2: A benchmark comparing a typical $D^+ \to K^+\pi^+\pi^-$ combination with different combiners. CombineParticles and NBodyDecays use LoKi functors, ThorParticleCombiner uses ThOr.

Overall, the implementation logic is very similar in all combiners that have been introduced up to now. Many aspects of the logic are constrained by the LHCb::Particle model. In benchmarks of the algorithms it turns out that the overlap check is a bottleneck in the baseline combiners. This is even the case in the combination of basic particles where the implementation boils down to a couple of pointer comparisons. Dynamic memory allocation appears in every overlap-check invocation, regardless of the particle type. As overlap checking is part of the critical path, this contributes a lot to the algorithm runtime.

7.5 A new particle model

The performance improvements in the previous section are promising and motivate to go further. Several key design choices in the HLT2 data model seem to hinder performance. Changes in the data layout have already been found to unlock large gains in HLT1 [77]. These changes mostly concern the minimization of dynamic memory allocation, the respect of cache locality and a data layout to enable parallelism on the level of vectorization. As long as the right accessors are implemented, ThOr functors can work on any data layout.

7.5.1 Data layouts and LHCb::Particle

When designing a model in the object-oriented paradigm, a class usually represents an object to be described.

```
struct Point3D {
  float x,y,z;
};
```

When describing multiple points at once, one then resorts to collections over `Point` instances:

```
using N_Points = std::array<Point3D, N>;
```

This data layout is referred to as Array-of-Structures (AoS). For contiguous collections, the memory layout of such a collection of points looks as depicted in the upper half of Figure 7.5. All features of one point are contiguous and spatially displaced from features of other points.
The LHCb::Particle is designed as such a structure, displayed here in a simplified manner:

```
struct Particle {
  ParticleID m_pid;
  Vertex* m_decay_vertex;
  std::vector<Child*> m_children;
  Track* m_track;
  ...
};
```

The second, less intuitive layout is called Structure-of-Arrays (SoA) and is shown in the bottom half of Figure 7.5. Instead of representing multiple points by a collection of Point class instances, one can transpose the layout to three collections, one for each feature of a point:

```
struct N_Points {
  std::array<float,N> x;
  std::array<float,N> y;
  std::array<float,N> z;
};
```

This achieves contiguous features in memory as opposed to contiguous structures in AoS.

Multiple LHCb::Particle instances are always represented in AoS fashion, which brings advantages as well as disadvantages.

The biggest advantage is intuition and readability. Encapsulating and combining properties into structures that have meaning is a much more natural step for a human than packaging features independently. A particle means more to somebody reading the code than a collection of x positions. Moreover, it is more natural to represent a single particle in AoS than in SoA, which is useful for many use cases in LHCb, like traversing a particle's decay chain.

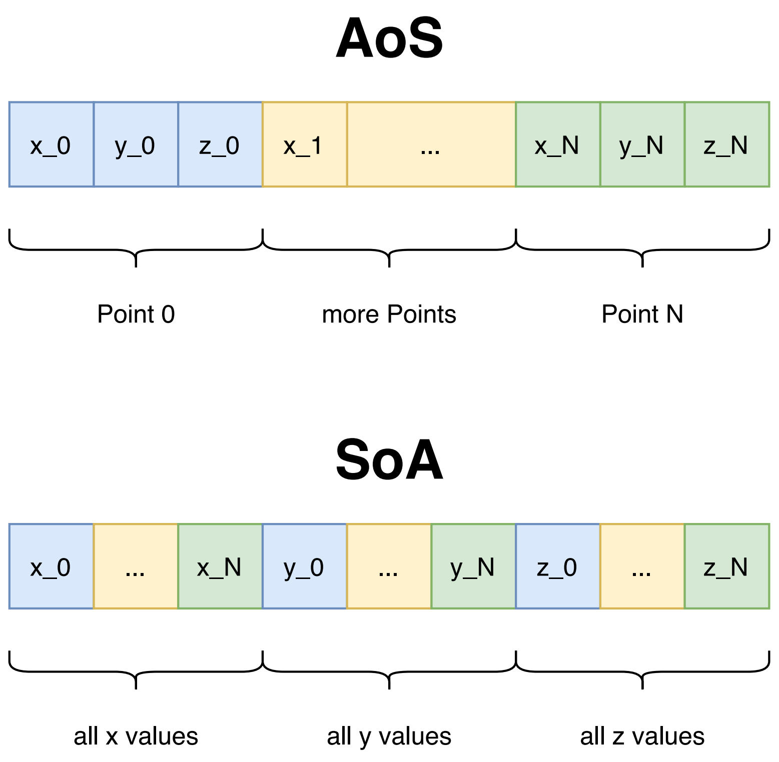


Figure 7.5: Array of Structures (AoS) vs Structure of Arrays (SoA).

The biggest disadvantage is SIMD performance. SIMD, or Single-Instruction-Multiple-Data, is a programming paradigm where the task can be formulated as a single sequence of instructions that has to be applied to a lot of data, element-wise:

```
//AoS
std::array<Point, N> points, other_points;
array<int, N> x_sums;
for (int i = 0; i < N; ++i) {
  x_sums[i] = points[i].x + other_points.x[i];
}

//SoA
N_Points points, other_points;
array<int, N> x_sums;
for (int i = 0; i < N; ++i) {
  x_sums[i] = points.x[i] + other_points.x[i];
}
```

While these operations look very similar and accomplish the same thing, there is one crucial difference. The SoA loop accesses contiguous elements as it evolves. For each new `i`, one just adds 4 bytes to the previously used memory addresses. The AoS loop does not access memory contiguously and has to jump the size of one `Point` every time to get to the next `x` value.

There are at least two reasons to prefer the SoA loop. Firstly, the cache hit rate is very high. The AoS loop on the other hand uses only about one third of the data received by one cache load, and it gets worse the more members a structure has.

Secondly, CPU vector instructions operate on contiguous memory. Because the SoA layout naturally ensures the dense memory structure in each feature, that loop can be transformed to use vector instructions quite natively. As these instructions are almost as fast as scalar instructions, speed-ups of up to 8-fold can be achieved with modern CPU vectorization. Real life use cases naturally bring their caveats, but they can often be outweighed using vectorization in a sufficiently parallel task.

Since selections and combinatorics clearly describe a SIMD paradigm, the task ahead is to create a particle model that lays out its data in a SoA fashion.

7.5.2 The SoA particle

The design of the new particle model presented in this section is based on two dimensional tables and operations on them. Columns represent features, and one row represents the corresponding particle. One should be able to

1. create a column
2. merge columns into a table
3. add more columns to an existing table
4. read a row
5. read multiple rows at a time for vectorization

In LHCb::Particle, many of the data members were invalid, depending on what type of particle is currently represented. For instance, a basic particle does not have a decay vertex, and composite or neutral particles do not have a corresponding track. Every optional member, in this case represented by pointers, needs to be checked for existence whenever acted on. To implement these checks is easy to forget and nullpointer access results in segmentation violations that can be difficult to debug. Moreover, every existence check is an additional runtime branch. To avoid these caveats, every particle type in the new model is represented by a distinct type. The absence of an accessor yields a compiler error that gives much more comprehensive information than a segmentation fault.

Charged basic particles are modelled by a very thin wrapper upon a number of features that define the particle. There are multiple different algorithms that assemble the features needed to define ChargedBasics throughout different reconstruction steps. These features are zipped together after reconstruction. Zipping gets its name from the python `zip` operation and corresponds to merging the feature columns into a table that, when iterated over, yields entire rows.

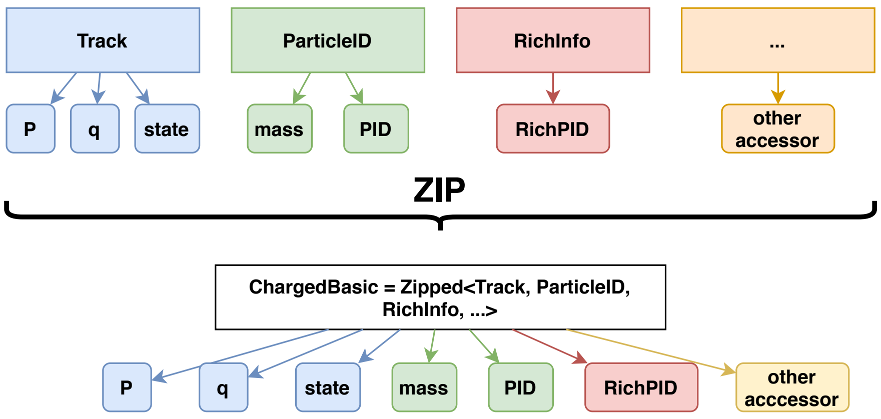


Figure 7.6: Zipping columns together to mimic charged basic particles.

Every feature column defines a proxy type that acts as front end to the column, exporting all necessary accessors. The zipped structure then automatically defines its own proxy that inherits from all columns proxies, thus accumulating all accessors. A sketch of the zipping operation can be seen in Figure 7.6. A custom iterator yields proxy instances with an offset and a width. Proxy accessors can either yield scalars or vectors of any width a vector instruction supports. Instead of looping over particles, the loop happens implicitly over chunks of columns with accessors to make it look like an AoS layout:

```cpp
// yield width 8 vectors
for (auto particle : zipped_particle_structure.with<avx2>()) {
  auto mask = particle.pt() > 5*units::GeV;
  store_if(mask, output, particle);
}
```

This code is explicitly vectorized. `particle.pt()` is an accessor that is made available by the track that has been zipped into the structure. It yields a custom vector type that overloads all common arithmetic operations to use appropriate vector instructions in the background with the intel intrinsics library [78]. Up to control statements like `if`, the code has the readability of AoS code but can still run explicit vectorization and benefit from the cache locality that contiguous memory access provides. Control statements can not be applied natively, because the condition for the control statement might be different for different data. Every branch that dispatches based on information of a single particle needs transformation into a masked operation. `store_if` will only store indices where the mask entry is not zero.

Although zips are non-owning views, they can invoke a copy operation onto their storage to be able to create an equivalent, but modifiable structure. This is mainly used to select and consolidate particles after they were created in the reconstruction.

Composite Particles are created within combiner algorithms. They are not assembled column by column and are thus not modelled by a zipped structure themselves, although they can be used within zips whenever additional per-particle information is needed. Composite particles are modelled to minimize dynamic memory allocation and maximize locality. The Composite uses a specific underlying storage called ”SOACollection”. It mimics a table just like a zip, but with all columns next to each other in memory. The SOACollection only has one heap memory block, and all columns are fit into it right after each other. This has the disadvantage that one copy per column has to be performed whenever the size of the container has to be extended, but only one allocation happens for all columns. Reallocation is avoidable because the number of particles coming out of a HLT2 combination is limited.

To further mitigate dynamic memory allocation bottlenecks, a memory pool is employed with a custom allocator. Every event task has its own pre-allocated memory pool and every structure with the custom allocator can tap into that pool to avoid the costly dynamic allocation. The SOACollection supports the allocator and thus allocates very cheaply.

Just like basic particles, the composites export a proxy with accessors that can yield information of variable width to enable vectorization. The new composite model currently supports a decay vertex information, kinematic properties, child relations over indices, and covariance matrices.

The zipping infrastructure, the SOACollection and the custom allocator were initially developed for HLT1 and can now be reused in HLT2 selections with great success.

7.6 Filtering and combining with the SoA particle model

This section presents filter and combination algorithms to exploit the advantages that the new particle model provides.

Everything needed to efficiently filter particles is already provided by the model backend. It is sufficient to specialize the PrFilter algorithm to work with the new particle types. A loop iterates in vector strides and applies a compressed store operation on the corresponding output storage. Unlike the LHCb::Particle filters, these explicitly copy and consolidate particle information to maintain a contiguous memory structure from which one can vectorize natively. This comes with a cost, but it simplifies following operations if one can assume contiguous inputs. A benchmark of all filter implementations is shown in Table 7.3. The performance numbers suggest that most of the time is used to consolidate the particles. Firstly, the major difference between the two PrFilter implementations is the consolidation. Secondly, the performance difference between the simple (ISMUON) and the more complicated selection is negligible in the SoA particle scenario.

	ISMUON	displaced presel.
FilterDesktop	11.6 µs	171 µs
PrFilter	4.6 µs	12.8 µs
PrFilter with SoA Particle	10.4 µs	10.5 µs

Table 7.3: A benchmark comparing two selections with all implemented filter algorithms. This extends Table 7.1 by PrFilter operating on the SoA particle.

However, the real gains are expected to be found in a combiner algorithm on the SoA particle model. Vectorization can now be employed efficiently as ThOr and the new model are designed to support it.

7.6.1 A Combiner for the SoA Particle

A new combiner algorithm dubbed ”ThOrCombiner” is implemented to work with ThOr functors and the SoA particle model. It is implemented generically to be able to handle the different particle types as uniformly as possible.

In contrast to the previously outlined combiners, this one does not implement a child stage. Child cuts complicate the implementation and have only little benefit as they can be performed by preceding Filters. Additionally, only one decay descriptor is allowed per combiner instance. The only exception to this rule is a global charge conjugation, which results in two descriptors. Child containers need to be provided in order according to the given decay descriptor. Whenever there are two children of the same particle ID, the corresponding containers must match exactly. The typical selection is expected to be invariant over input permutation. If really needed, any asymmetry of input particles can be expressed with a combination cut. With this assumption the $O(n^2)$ checks for unique combinations outlined in Section 7.3.2 can be avoided without having to merge inputs of same particle ID as the other combiners do. The following sections describe the algorithm in greater detail, once with pseudocode and once on a higher level with pure text.

The algorithm logic

The combiner logic is best displayed with pseudocode. Here, ”←” assigns, ”←←” adds to a collection and ”#” evaluates the size of a collection. More details on the logic can be found in the following subsections.

```
• W ← vector width for vectorization
• N ← the number of children in the decay descriptor (at least 2)
• CS ← empty list. It stores for combination indices in flight. One store for
  every n ∈ [2,N]. CS[i] refers to (i+2)-body combinations.
• CANDS ← empty list. It stores the fitted candidates.
• loop: decay ← decay descriptors
    – indices ← containers of indices into particle containers that match the
      decay descriptor
    – loop: p0 ← indices[0] (indices of first particle container)
        ∗ (triangular) loop: p1s ← W elements of indices[1]
            1. CS[0] ←← combine p0 with every element of p1s, filter them
               with an overlap check and the two-body combination cut.
            2. if #CS[0] < W: next p1s loop iteration
            3. nextCS ← move W elements from CS[0]
            4. if #indices == N:
               CANDS ←← vertex fitted and selected nextCS
            5. else: (triangular) loop: p2 ← indices[2]
                · CS[1] ←← nextCS with p2 added to each combination,
                  filtered by overlap check and three-body combination cut.
                · if #CS[1] < W: next p2s loop iteration
                · nextCS ← move W elements from CS[1]
                · if #indices == N:
                  CANDS ←← vertex fitted and selected nextCS
                · else: (triangular) loop: p3 ← indices[3] with the equivalent
                  logic as the loop on indices[2] ... This is implemented
                  recursively!
    – every CS[i] for i < (N-2) still contains [2..N-1]-body combinations due
      to the loop break on #CS[i] < W. These are now extended to N-body
      combinations and then fitted and appended to CANDS in the same
      way as above.
• CS[N-2] still contains (less than W) N-body combinations. These will be
  fitted, selected and appended to CANDS.
```

Combining

The combiner algorithm starts by selecting indices from the particle containers according to the charge and particle ID specified in the decay descriptor. Next, combinations need to be built from these indices. Conceptually, this corresponds to a N-dimensional cartesian product where N is the number of children in the decay descriptor. Combinations are built recursively starting from 2 indices. More indices are added to every combination until the correct number of children is reached.

When the same particle ID appears multiple times in the decay descriptor, other combiners applied a kinematic ordering to avoid combination duplications. In this combiner, entries in the particle input list are assumed to be unique. One can select combinations solely based on indices, considering only the lower triangle of the cartesian tensor. This yields the behavior that is outlined in the left half of Figure 7.3, independently of ordering in the container. This ”triangular” logic is implemented by starting the second particle loop from the first loops current index + 1 instead of from 0. That is simpler and cheaper than selecting based on kinematic properties, and the unique entries assumption is very reasonable in the normal HLT2 workflow, as most operations select from a unique collection of reconstructed particles.

The algorithm always works on a chunk of combinations of a size equal to the vector width W. These W-sized chunks of [2..N]-body combinations are selected via a [2..N]-body combination cut and overlap checks in each respective loop. This causes chunk size reduction. To maintain a high vector instruction utilization, these chunks are refilled with new combinations to at least size W before being processed by the next vector instruction.

Overlap checks are trivial in the currently implemented combiners covering combination of charged basic particles. A simple comparison of underlying tracks can be performed without much runtime branching, as a charged basic type guarantees the existence of a track. Composite children are envisioned to carry a flat set of object IDs, and overlap checks can then be implemented by checking the intersection over these sets. This is computationally much more efficient than creating these sets on the fly with dynamic allocation in the inner loops of the combiner.

Vertexing

Whenever a full vector width of valid N-body combinations is produced, the vertex fit functionality is invoked in a vectorized fashion. A purely functional linear algebra library was specifically designed to allow vectorized matrix and vector operations in the LHCb environment. It is utilized to perform the fit in a fully vectorized fashion. After the vertex cut is passed, the composite output particles are filled with all necessary information and added to the output storage. The vertex fit is always performed on size W combination chunks until the very end, where the residual (#combinations mod W) combinations are fitted. The residuals are fitted after the loop over the decay descriptors to ensure the highest possible vector instruction utilization.

7.6.2 Benchmarks on combining with the SoA Particle

The success of the particle model and the new combiner is immediately apparent when benchmarking the $D^+ \to K^+\pi^+\pi^-$ selection from Section 7.4.1. A collection of all results can be seen in Table 7.4. As the new combiner can optionally vectorize if vector instructions are available, multiple possibilities are shown here: Scalar execution without vectorization, Vectorization with width 4 (SSE instruction set) and with width 8 (AVX2 instruction set). While vectorization brings down the runtime by 35%, it is not the biggest contributor to the increased speed. This is due to the fact that there are often too few combinations to utilize the vector instructions well enough. The reduced dynamic memory allocation due to the specialized overlap check is among the biggest contributors to speed up between the scalar ThorCombiner and the ThOrParticleCombiner. When selection criteria are loosened, vectorization becomes more effective.

When the performance displayed in these benchmarks extends to other trigger selections in a similar manner, the 20-25% contributions from the current selection algorithms can brought down to percent level. Trigger selection authors need to utilize the [2..N-1] combination cuts more rigorously than in the past, where they have often been neglected. When selections implemented in the ThOr and SoA particle environment are properly tuned, I am confident that the contribution of selections to the overall runtime stays in the single digit percent level.

Implementation	$D^+ \to K^+\pi^+\pi^-$ execution time
CombineParticles	256 µs
NBodyDecays	77.1 µs
ThorParticleCombiner	38.8 µs
ThOrCombiner Scalar	10.2 µs
ThOrCombiner SSE	7.5 µs
ThOrCombiner AVX2	6.9 µs

Table 7.4: A benchmark comparing a typical $D^+ \to K^+\pi^+\pi^-$ combination with different combiners. Extends Table 7.2 by the ThOrCombiner implementation working on the SoA particle and ThOr functors.

7.7 Conclusion and Outlook

In this chapter several new features for particle selection and combination with the potential to significantly increase the throughput in the HLT2 application were presented. Although the outlined benchmarks are not exemplary for all the HLT2 use cases, the drastic throughput increase suggests overall superiority of ThOr and the SoA particle model. Moreover, the algorithms working on the new model scale much better with an increased combinatorial load as they can make better use of vectorization. This is important since the most expensive combiners in Run 2 HLT2 were those with high combinatorial load.

The implementation of additional features into ThOr, the model and the combiners is a top priority in the imminent future. There are still several missing features to make these new frameworks suitable for a fully operational HLT2. Missing features include the combination of composite children, an implementation for neutral basics, related particle information, composite overlap checks and several ThOr functors.

The functionality used in the benchmarks has been verified during development and testing. All configurations tested show equivalent efficiencies for all selections involved. The vertex fit has been verified to yield the same results as the baseline algorithms up to floating point differences. All validation of physical correctness beyond efficiency comparisons is yet to be performed.

If time proves too short to cover all these tasks, the ThOr combiner working on LHCb::Particle can already give significant speed-ups with a particle model that has been validated for multiple years. Properly employing sub-combination cuts makes a crucial difference, and these have not been used very regularly in Run 2 combiners. The only thing missing to make that combiner fully functional are less commonly used ThOr functors.

In conclusion, the work presented here lays the groundwork for a highly efficient selection framework suitable for a 1 MHz HLT2. Actions to transfer existing lines to this framework are already being taken. ThOr functors at the very least, but likely also the SoA particle model will soon become the new baseline for HLT2 production.

8 The Topological Trigger with Selective Persistence

The previous chapters concerned the increase of throughput for the HLT application for online production. This chapter will concentrate on the second biggest problem of the HLT production environment, the output bandwidth. With the amount of lines anticipated and the expected physics output for Run 3, the upper limit for output bandwidth of 10 GB/s represents a tight constraint.

The work presented in this chapter will try to mitigate bandwidth bottlenecks by scrutinizing the most prominent physics selection in HLT2, the so-called topological trigger, or Topo for short. It aims to select a wide range of beauty decays with an inclusive selection focussing on the spatial and kinematical decay topology. The Topo works by combining an inclusive list of basic particles with sufficient momentum and impact parameter into vertices and selecting them based on vertex quality and displacement. It works in two modes, a two- and a three-body combination. The selections are designed such that also (N>3)-prong decays can be captured. After combinations have been built, a multivariate classification algorithm is employed to further distill the selection down to an acceptable bandwidth. Due to its inclusive nature, the bandwidth output of this trigger line is orders of magnitude larger with respect to the usual exclusive HLT2 lines. This chapter aims to construct an algorithm to select a wide list of beauty decays with high efficiency while maintaining the lowest possible output bandwidth. Two strategies are employed to achieve this:

1. Select signal with a high efficiency while rejecting background events with high probability. The precision, that is the number of true positives over the number of all selected, is to be maximized.

2. Employ selective persistence. The inclusive nature of the topological trigger makes the selection of relevant decay parts a much harder task than in exclusive lines, where the decay nature is known beforehand. The task is to identify parts of the full event that are relevant for future analyses.

Other parts of the event can then be discarded to lower the bandwidth even further.

Note that the work presented in this chapter was performed mostly by me, but the work on selective persistence is based on a similar study performed by Alex Pearce and Vladislav Belavin [79].

8.1 Input data

The studies presented in this chapter work exclusively with simulated samples to model the upgrade detector and HLT conditions. Default upgrade collision and detector conditions have been used. As the Topological trigger aims to select a wide range of decay signatures inclusively, a collection of decay samples is used as representation of the signal. The motivation for choosing most of these channels is given in Chapter 2.1. Next to choosing decays of high physical relevance it is also important to choose a set of decays that reflects all topological signatures that one aims to select. This collection contains two-, three- and four-body vertices, different parent particle flavors, neutral particles and semi-leptonic decays, where non-reconstructible neutrinos take part. A classifier dealing with this collection also has to cope with electrons with higher reconstruction uncertainties due to the increased amount of emitted bremsstrahlung in the detector. For every decay, the charge conjugated decay is implicitly considered as well. Parentheses are enclosed around decay products that decay into something specific themselves. For instance, $B_s^0 \to (\phi \to K^+K^-)(\phi \to K^+K^-)$ refers to the B_s^0 decay into two ϕ mesons which each decay into K^+K^-. The samples comprise simulations of the following decays:

- $B^0 \to K^+\pi^-$
- $B^0 \to (D^+ \to K^-\pi^+\pi^+)(D^- \to K^+\pi^-\pi^-)$
- $B_s^0 \to (\phi \to K^+K^-)(\phi \to K^+K^-)$
- $B^0 \to \pi^+\pi^-(\pi^0 \to \gamma\gamma)$
- $B^0 \to e^+e^-(K^*(892) \to K^+\pi^-)$
- $B^0 \to (D^{*-} \to (\overline{D}^0 \to K^+K^-)\pi^-)\mu^+\nu_\mu$
- $B^0 \to (D^{*-} \to (\overline{D}^0 \to K^+K^-)\pi^-)(\tau^+ \to \pi^+\pi^+\pi^-\overline{\nu}_\tau)\nu_\tau$

- $B^0 \to (D^{*-} \to (\overline{D}^0 \to K^+K^-)\pi^-)(\tau^+ \to \mu^+\nu_\mu\overline{\nu}_\tau)\nu_\tau$
- $B_s^0 \to (D_s^- \to K^+K^-\pi^-)K^+$
- $\Lambda_b^0 \to (\Lambda_c^+ \to pK^-\pi^+)(D_s^- \to K^+K^-\pi^-)$

Minimum bias refers to a type of simulation that aims to resemble the data read out during production as closely as possible. Measured decay probabilities as defined by the PDG [80] are used for every known process. Minimum bias simulation is used to estimate rates and to serve as background for classification purposes.

Simulation of a specific decay at LHCb means that a full event is simulated where at least one of the present decays involve the listed one. For example, the first listed signal simulation has at least one $B^0 \to K^+\pi^-$ decay in each event, but the other b quark from the pair production might have hadronized and decayed differently. There are cuts on the simulation samples making sure that the all final state particles of the signal decay are in detector acceptance.

All samples in this chapter are filtered by HLT1. They only contain events that passed at least one HLT1 trigger line. More information on the simulated samples can be found in Appendix A.2.

8.2 Optimization of the topological event selection

The topological trigger, aims to select beauty decays solely based on topological information so as to be able to generalize over specific particle IDs. Beauty flavored particles have hundreds of known and unknown decay modes. The Topo is in place to avoid having to write exclusive lines for each of the known channels and also capture possibly unknown or forgotten channels implicitly. Due to the distinctive signatures and comparably low rate of beauty hadrons, the rate of an inclusive beauty line is acceptable at high signal efficiencies. Due to the higher cross-section of charm hadrons, an inclusive charm line is not possible with an acceptable overall efficiency. Ideally, every beauty decay is to be captured and selected by the Topo trigger line. The workflow is organized as follows:

1. Combine charged basic particles without specific PID requirement into a 2,3-body combination to produce B candidates.

2. Select B candidates based on kinematic properties of the children and the candidate itself, and vertex fit information.
3. Persist an event whenever there is at least one candidate passing all requirements.

First, the metric for optimization will be defined. In Section 8.2.2 the combination and preselection stage of the Topo is outlined. After Section 8.2.3 introduces some theory on the used algorithms, Section 8.2.4 outlines the baseline classifier and how to improve upon it. Section 8.2.5 shows the performance of the improved algorithm on the full input data set given a fixed output bandwidth.

8.2.1 The metric for optimization: Trigger On Signal efficiency

To optimize selections in this chapter, the metric for optimization must first be defined. The final goal is to efficiently select signal decays from the input samples while keeping the output rate and bandwidth at a defined upper limit. One could consider a signal decay selected as soon as the event containing the signal decay is selected by the Topo trigger. This implies that enough information of the event is persisted to reconstruct the full signal decay if that has not happened. However, when choosing for trigger requirements to use in an analysis, one needs to be able to model possible backgrounds appearing in the data to be able to extract a physical quantity of interest. This might pose a significant problem. If the selection depends on quantities of the event besides the signal decay itself, the event itself has to be modelled. To avoid this, most analysts chose to only use events in which the trigger line selected based on the signal candidate itself. This is referred to as Trigger On Signal (TOS).

A signal candidate is defined as one in which all basic particles belong to decay products of the signal decay. This information is produced by matching simulated to reconstructed particles. The true decay chain of the simulated particle can be inspected and considered for evaluation of ground truth. The decay is different for every simulation sample. Minimum bias only consists of background candidates.

In Section 8.2.4 the baseline selection algorithm is introduced. This algorithm is optimized to select candidates where all children come from the same beauty hadron as signal. This definition will be referred to as ”FromSameB”. It is different to the definition for a signal candidate because the signal decay may not be the only beauty decay in the event. Simulation generation guarantees one generated $b\bar{b}$ pair. Changing this optimization goal is one reason for a defining a new algorithm.

TOS efficiencies in this chapter are calculated under the assumption that there is at least one candidate in each event as every simulation sample contains at least one signal decay. This assumption is arguable as the final state particle of the signal might not be reconstructed. However, algorithms might manage to reconstruct and make use of decays using only a subset of the final state particles. Optimizing for the maximization of this definition of efficiency is not harmful. Efficiencies for trigger selections have been optimized against common offline selections in the past [57]. However, as Upgrade LHCb is a completely new detector, efficiencies should not be biased against possible future offline selections that are based on experience with the current detector. Thus, every selection efficiency in this chapter is defined as follows,

$$\epsilon = \frac{\#\text{events with a signal-matched candidate}}{\#\text{events going into the selection}}. \tag{8.1}$$

8.2.2 Topo candidate combination

The events considered for this trigger line are ones which passed the HLT1 filter. Here, specifically at least one of three trigger lines needs to be triggered on signal:

1. Hlt1TrackMuonMVA
2. Hlt1TrackMVAVLoose
3. Hlt1TwoTrackMVAVLoose

These single track lines select displaced tracks with high transverse momentum. In this chapter displacement is measured with respect to primary vertices. The muons are processed separately as they do not need as tight displacement cuts to control bandwidth. The two-track line makes a combination and selects based on the χ^2 of the vertex fit (χ^2_{vtx}), displacement and summed combination transverse momentum. The exact configuration of these trigger lines can be found in Appendix A.1.

The Topo combination starts from long tracks with a χ^2 (IP) of at least 4. This selection excludes tracks that can be closely extrapolated back to a PV. χ^2 (IP) is calculated by calculating the χ^2 on a single iteration fit of the track onto the closest primary vertex. It serves as measure for displacement from the beam. Tracks that come from beauty mesons mostly exhibit high χ^2 (IP) values because they extrapolate back to the displaced beauty decay vertex instead of any PV. A distribution of χ^2 (IP) in $B_s^0 \to \phi\phi$ events can be seen in Figure 8.1.

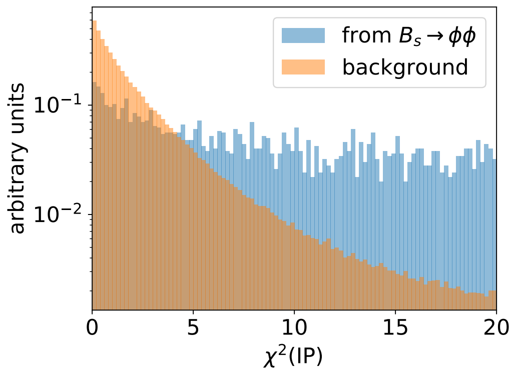


Figure 8.1: χ^2 (IP) distribution of long tracks in $B_s^0 \to \phi\phi$ simulated events. The orange distribution corresponds to tracks that do not originate from beauty hadrons. The blue distribution contains tracks which come from $B_s^0 \to \phi\phi$.

The tracks are transformed into particles and combined into B candidate. The two-body combination is selected to have a distance of closest approach of less than 1 mm. If the tracks do not extrapolate to a common point, it becomes much less likely that they originated from the same decay vertex. Because beauty mesons have a relatively high mass, the combined transverse momentum of the tracks in the combination is expected to be larger than for the average combination of tracks. Therefore, a combined transverse momentum of over 1 GeV is required for the two-body combination.

The candidates' selection comprises the following requirements:

1. Flight distance χ^2 greater than 16. This indicates high displacement of the decay vertex and primary vertices. Because beauty hadrons have high lifetimes, this cut has high separation power.
2. cos(DirA) with respect to the closest PV greater than 0. DirAis the opening angle between the candidate momentum and the line between decay vertex and best PV. This value might differ from 0 significantly if not all tracks are captured by the combination. However, an angle over 90° is unlikely for signal decays as this would imply a very large amount of missing momentum.

3. Pseudorapidity of the interpolation between best PV and decay vertex with respect to the beam line between 2 and 5. This is equivalent to the LHCb detector acceptance. Almost every combination passes this cut.

To motivate the use of cos(DirA) and flight distance χ^2 in the preselection, the distributions of these variables in $B_s^0 \to \phi\phi$ events are shown in Figures 8.2. While the cuts are loose, they can already get rid of a large portion of the background.

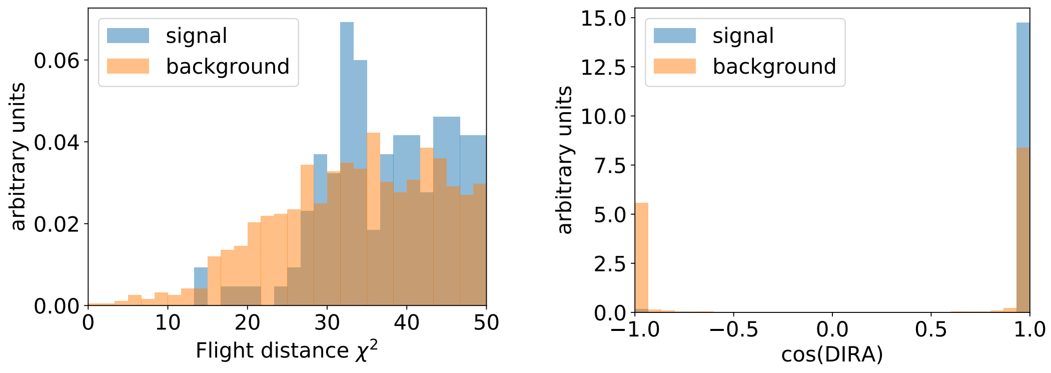


Figure 8.2: Flight distance χ^2 (left) and cos(DirA) (right) distributions two-body combinations in $B_s^0 \to \phi\phi$ simulated events. The blue distribution corresponds to signal candidates, truth matched to the signal decay. The orange distribution shows background candidates.

For the two-body selection, the Topo candidate is required to pass any of the previously listed HLT1 lines. TOS in single track lines is defined such that either one of the tracks in the Topo candidate should have passed the lines requirements.

The three-body combination builds upon the two-body combination. After another cut on the two-body combination, χ^2_{vtx} smaller than 10, these are combined with another charged basic particle. The three-body combination is required to have a combined transverse momentum of over 2 GeV. The same distance of closest approach requirement and the same vertex cuts as in the two-body case are applied. Again, at least one child has to pass one of the HLT1 trigger lines. Note that the two-body combinations considered for the three-body combination do not need to pass the HLT1 lines by themselves, only in combination with the third track.

Efficiencies for the outlined combination and selection are displayed in Table 8.1. The definition of the efficiencies is given in Section 8.2.1. Two-body decays like $B^0 \to K^+\pi^-$ exhibit a lower efficiency due to combinatorial chance. Reconstructing exactly the two right tracks is a two-body Topo is less likely than combining two out of N for a N-body decay with N > 2.

Table 8.1: TOS efficiencies for the preselection.

Decay	Efficiency [%]
$B^0 \to K^+\pi^-$	35.9
$B^0 \to D^+D^-$	79.1
$B^0_s \to \phi\phi$	73.7
$B^0 \to \pi^+\pi^-\pi^0$	48.1
$B^0 \to K^*(892)e^+e^-$	63.6
$B^0 \to D^{*-}\mu^+\nu_\mu$	33.0
$B^0 \to D^{*-}(\tau^+ \to 3\pi\overline{\nu}_\tau)\nu_\tau$	46.6
$B^0 \to D^{*-}(\tau^+ \to \mu^+\nu_\mu\overline{\nu}_\tau)\nu_\tau$	46.3
$B^0_s \to D^-_s K^+$	75.9
$\Lambda^0_b \to \Lambda^+_c D^-_s$	78.2

8.2.3 Boosted tree ensembles

The B candidate selection after reconstruction and combination is a multivariate binary classification on a relatively small feature space. Ensemble methods, specifically Boosting, have proven to be very effective in these tasks. Gradient boosting algorithms have been dominating the Kaggle [81] classification scene for several years now. LHCb has a history of employing boosting for classification tasks, mostly in data analyses. The in-house implementation of several ML algorithms including boosted decision trees, TMVA [82], has been used for over a decade. Boosted trees have been used to train the topological selection for Run 2 and we will continue to do so here.

The basics of boosting will shortly be outlined in the context of Classification And Regression Tree (CART) ensembles. Firstly, a CART is a binary tree structure that splits inputs based on their properties, recursively. Figure 8.3 shows an example tree with weights in the leaves. Regression trees have continuous and classification trees have discrete output. An input to the tree will traverse it by applying the selections defined in the nodes. If the input meets the requirement, it proceeds along a green arrow, otherwise a red. The output is the weight in the leaf the input ends up in. If weights and criteria are well-adjusted, one could interpret the output of this tree as an estimation for signal probability for muons from a B meson. By increasing the depth and the number of leaves, more sophisticated selections can be encoded in a decision tree. Trees are usually built by iteratively introducing new nodes. In each step the criterion that can best optimize an objective function is chosen to form a new node. This is called greedy optimization.

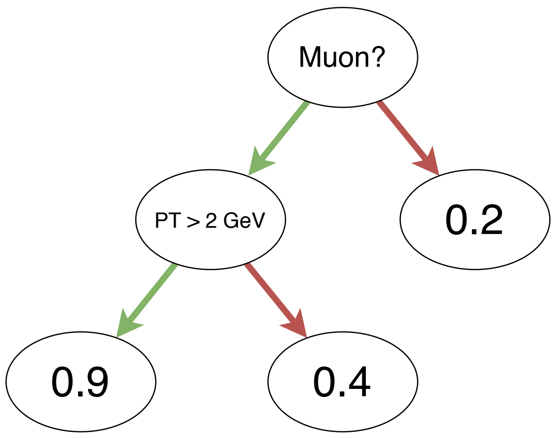


Figure 8.3: A simple tree with 3 leaves and a depth of 2. It assigns a number to the input based on which leaf it ends up in. A green arrow corresponds to a pass of the criterion defined by the node, a red arrow means fail.

To model more complex functions, a single tree needs a huge number of parameters. Because of this property, they are often referred to as ”weak learners”. This is where boosting algorithms come in. Instead of considering one tree, they build ensembles of trees, the weights of which are added to achieve arbitrarily good function approximations. Most boosting algorithms iteratively fit weak learners to rectify miss-predictions of the previous iteration. Gradient boosting will be used in all new Topo implementations described in this chapter, so is quickly outlined here:

1. Training set: N Samples and truth $(x_i, y_i \in \{-1, 1\})$
2. A differential loss function $L(y, \hat{y})$.

3. Initialize the model $F_0(x_i) = const.$
4. for $k = 1..K$

    a) Compute residuals (errors) of the current model:

    $$k_i = -\frac{\partial L(y_i, F_{k-1}(x_i))}{\partial F_{k-1}(x_i)}$$

    b) Fit a weak learner $h_k(x)$ to model the residuals.

    c) Choose a weight γ_k to minimize the loss on the current model:

    $$L_k(y, \hat{y}) = \sum_i L(y_i, (F_{k-1} + \gamma_k h_k)(x_i))$$

    d) Update the model $F_k(x) = F_{k-1} + \gamma_k h_k(x)$

Gradient boosting can be interpreted as generalization to AdaBoost, the algorithm used in previous implementations of the Topo [83]. It uses an exponential loss function $L(y, \hat{y}) = \sum_i e^{-y_i \cdot \hat{y}_i}$. There are many parameters in a gradient boosting classifier that can be tuned to increase performance. For instance, one can only consider subsets of the data in each step, or use different factors γ_k to adjust how fast and how likely classifiers reach local minima.

8.2.4 Topological classification - Baseline comparison

The heart of the topological trigger is the multivariate selection that acts on the previously produced and selected Topo candidates. As opposed to other, exclusive trigger lines, the selections described in Section 8.2.2 are very loose and yield a high output rate. The multivariate selection aims to optimize efficiency on reconstructed beauty decays while maintaining a relatively low output rate.

The Run 2 Topological trigger applies the multivariate classifier to each candidate individually. The resulting trigger decision corresponds to whether at least one of the candidates passes the classification. The current baseline for the upgrade Topo as well as the first new implementation described here use the same workflow. A gradient boosting algorithm proves to be very efficient for the classification task.

As shortly discussed in Section 8.2.1, the training data for the baseline algorithm uses the ”FromSameB” definition for classification. If all children in the candidate are matched to a simulated basic particle that comes from the same simulated B hadron, a candidate is considered to be part of the signal data set. This does not necessarily need to be the signal hadron of the simulated sample. Although it increases the statistical power of the signal data set, it does not fully represent the metric to optimize for, namely the efficiency of the signal decay. This is the first of four motivations to train a new Topo classifier. The second motivation is the input data. The baseline algorithm is trained on a smaller and less representative input collection. Thirdly, the input decay signatures are not weighted equally, but simply uniformly and thus proportionally to the different number of input samples available for each decay. Lastly, the TMVA implementation boosted tree ensembles trains rather slowly compared to modern implementations. From personal experience, it requires significant manual tuning to reach competitive performance. Instead, the Catboost [84] library is employed for the new algorithms. During the course of these studies, Catboost and LightGBM [85] were compared in different scenarios and with different hyperparameters and always yielded very similar results. Finally, Catboost was chosen because it has the option to export most of their models to native Python and C++, which greatly eases the integration into the LHCb software stack.

This section compares the different optimization methods and implementations. New algorithms are trained with the same signal definition as the baseline uses. Only the algorithms and data preprocessing cause performance differences.

In Section 8.2.5 algorithms are optimized for the selection of the signal decay itself.

Input data to the baseline approach comprises the following decays:

- $B^0 \to (D^+ \to K^-\pi^+\pi^+)(D^- \to K^+\pi^-\pi^-)$
- $B_s^0 \to (\phi \to K^+K^-)(\phi \to K^+K^-)$
- $B^0 \to \pi^+\pi^-(\pi^0 \to \gamma\gamma)$
- $B_s^0 \to (D_s^- \to K^+K^-\pi^-)K^+$
- $B^0 \to X_c\mu\nu_\tau$
- Minimum bias

Each entry in the training sample comprises features presented in Table 8.2 and the signal label. Distributions of three input features with high separation power are shown for illustration in Figures 8.4. Signal in $B_s^0 \to \phi\phi$ events are plotted against background candidates reconstructed in minimum bias events. Although these features show significant differences in signal and background, they also motivate the use of a multivariate classifier. None of these features can be used for rectangular selections without sacrificing portions of the signal. The boosted tree ensemble will does a much better job of retaining signal while discarding background due to exploitation of correlations between these features.

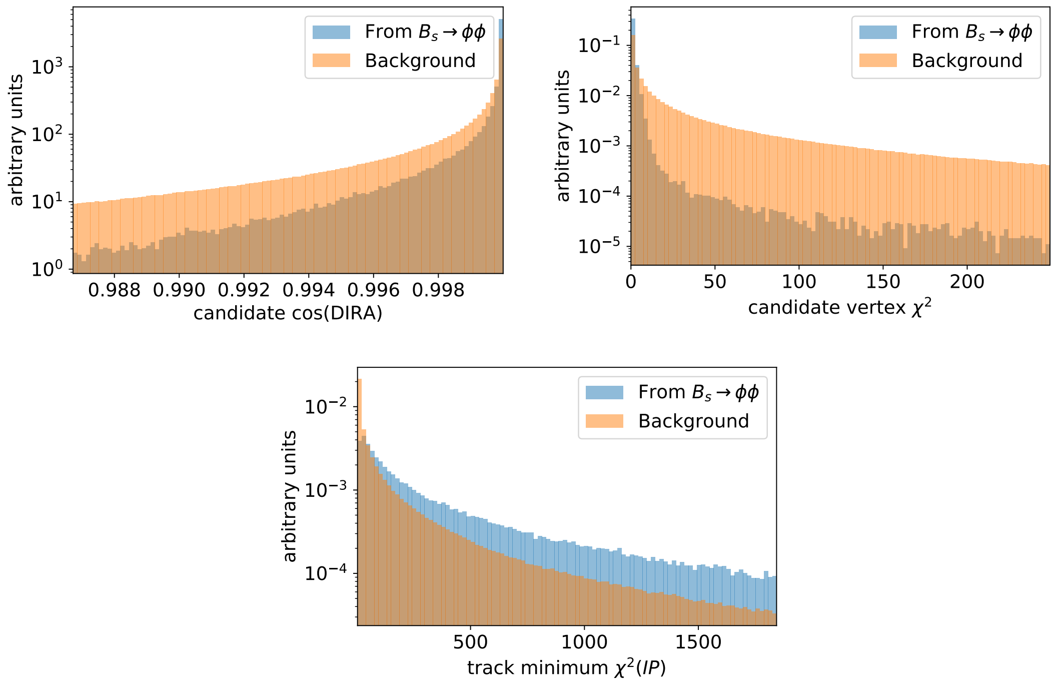


Figure 8.4: Distributions of candidate cos(DIRA), vertex χ^2 and track minimum χ^2(IP) for $B_s^0 \to \phi\phi$ signal candidates versus minimum bias background.

To measure and compare performances of different classifying solutions, the True Positive Rate (TPR) is plotted against the False Positive Rate (FPR). This is known as Receiver Operator Characteristic (ROC) and is a commonly used for performance visualization in binary classification tasks because it is independent of the number of background and signal samples and their ratio. The area under that curve (AUC) can give a single number score without the need to choose a cut point. For final evaluation, several realistic cut points will be chosen to measure efficiency given output rate.

The baseline implementation used the AdaBoost implementation in TMVA with 100 weak learners. The new models use gradient boosting on binary cross-entropy

Table 8.2: Input features to the Topo training

Feature	Explanation
(2,3)-Body CORRM	Corrected invariant mass $\sqrt{M^2 + (p_T^m)^2} + p_T^m$. p_T^m is the missing transverse momentum to match the momentum vector and the linear extrapolation between corresponding PV and decay vertex.
(2,3)-Body DOCAMAX	Maximum distance of closest approach between children.
(2,3)-Body FD χ^2	Measure for displacement of decay (d) and primary (p) vertex: $(\vec{x}_d - \vec{x}_p)^T(\text{cov}_d + \text{cov}_p)(\vec{x}_d - \vec{x}_p)$, where $\text{cov}_d(p)$ are the covariance matrices for decay or primary vertex position.
(2,3)-Body P(T)	(transverse) momentum of the reconstructed mother
(2,3)-Body cos(DIRA)	cos of the direction angle of the reconstructed mother with respect to the PV
(2,3)-Body χ^2 (IP)	χ^2 (IP) of the reconstructed mother
(2,3)-Body χ^2_{vtx}	χ^2_{vtx} of the reconstructed mother
child χ^2 (IP)	χ^2 (IP) of the children
child P(T)	(transverse) momentum of the children

loss with up to 3000 weak learners. With such a large number of weak learners, overtraining might become a problem. Overtraining names the phenomenon where a model becomes sensitive to the statistical fluctuations of its training data, thus lacks generalization and performs much worse on unseen samples.

Four methods are employed to counter overtraining: Firstly, a threefold cross validation is used. The data is split three-way, and training is performed on two thirds while validation happens on the third, unseen sample. This is done thrice, each time with a different third as validation sample. The model is considered fine if all three resulting models behave similarly. Secondly, the L2 generalization term in the loss function adaptively reduces the weights in the leaves when they get much larger than average. Thirdly, a technique called early stopping constraints the number of weak learners. When the model does not continue improving on the validation set, early stopping will stop the training progress and effectively reduce the number of weak learners. Lastly, the random subspacing helps with generalization because the model is forced to consider not only the features that appear very predictive on the training set.

Considering multiple candidates in a Topo event at once might yield more exploitable correlations for the classification algorithm. Therefore, a second training workflow was considered: Use the three candidates with the highest prediction in the previously trained model as one input sample to get a better event wise discrepancy. This workflow yields an event wise prediction, while the other configurations yield candidate wise predictions. The problem with event wise decisions is that a TOS efficiency cannot be defined. However, the classification performance can still be compared when ignoring the TOS requirement. To do so, candidate predictions are translated to event predictions by taking the highest prediction of all candidates in the event. An event wise label is defined as whether at least one candidate is present in it. Note that the approach with the three best candidates is not pursued further within this thesis as the final goal is defined to be TOS efficiency. It rather serves as showcase for the fact that information besides the signal candidate helps in selecting signal events more efficiently.

As both two- and three-body classification should run in the trigger, the important metric to compare is the combined performance. Therefore, predictions on both datasets are combined by taking the maximum prediction of any two- or three-body candidate within the event. This maximum aggregation is only negligibly worse than a brute force threshold optimization for best efficiency given fixed output rate in a two-dimensional space. The combined performance for baseline and new models is compared in Figures 8.5.

The data used for these scores represents a slice of the input data that has not been seen by any of the new models trained within the cross validation scheme. Because three models have been trained during cross validation, the unseen data will be classified by taking the mean prediction of all three models. Averaging rather than predicting with one model changes the scores negligibly, which suggests that generalization works well. Each plot presents a specific input decay for signal, but always minimum bias data for the background.

The comparison plots suggest that the newly trained models perform better in the general case. Specifically the $B_s^0 \to \phi\phi$ and $B^0 \to \pi^+\pi^-\pi^0$ decays are captured up to factors of 2 better, depending on the output bandwidth. The $B^0 \to D^+D^-$ sample is a factor 3 larger than all others. Overtraining might explain the performance differences over the samples. It is also apparent that considering multiple candidates in an event wise decision generally outperforms the simple maximum aggregation that the default Topo algorithm employs.

Note that Figures 8.5 show the classification performance in terms of the "FromSameB" definition, but these do not necessarily reflect the signal efficiency as defined in Section 8.2.1. Figures 8.6 therefore show signal efficiency given output rate relative to the input rate. Here, every simulated signal event is counted as signal since the simulation guarantees at least one signal decay in each simulated event. The rate is estimated on minimum bias events. One can see that the increased performance of the new algorithms is reflected in higher signal efficiency per output rate.

8.2.5 Topological classification - Full input data

The new models showing improved performance can now be applied to the wider input decay selection described in section 8.1. The training in this chapter optimizes TOS efficiency of signal events as defined in 8.2.2. Samples are weighted to ensure balance between the different samples. The performance of the combined two- and three-body classification algorithms given a minimum bias output rate can be seen in Figures 8.7.

The output bandwidth envisioned for the Topo amounts to 3 GB/s [83]. To estimate bandwidths given selection efficiencies, multiple factors need to be determined. The output bandwidth of the default Topo is determined as follows:

$$b(t) = r(t) \cdot \beta(t) \tag{8.2}$$

where b is the bandwidth, r the output rate, t the threshold configuration and β the average bandwidth per minimum bias event that passes the Topo selection. Because the number of Topo candidates vary with the thresholds, β is a function of t. To determine $\beta(t)$, the bandwidth of all pieces of information to be persisted have to be assessed. In the default persistence setting where all information of the event has to be persisted, β comprises the following parts:

$$\beta(t) = \beta^{\text{downstream}} + \beta^{\text{upstream}} + \beta^{\text{neutrals}} + \beta^{\text{long}} + \beta^{\text{topo}}(t) \tag{8.3}$$

The contributors to bandwidth are downstream, upstream and long tracks and neutral particles. Additionally, each reconstructed Topo candidate has some information to be persisted. The output bandwidth of each contributor has been estimated by running the baseline Topo as defined in 8.2.4. The persisted size of final state particles is determined to be

$$\beta^{\text{downstream}} = 10.2\,\text{kB}, \tag{8.4}$$

$$\beta^{\text{upstream}} = 11.5\,\text{kB}, \tag{8.5}$$

$$\beta^{\text{neutrals}} = 13.8\,\text{kB}, \tag{8.6}$$

$$\beta^{\text{long}} = 55\,\text{kB}. \tag{8.7}$$

These numbers represent an average size of all tracks of the specified type in an event. As the number of Topo candidates depends on the threshold configuration, the size of these are determined on a per-candidate basis:

$$\beta^{\text{topo}}(t) = \beta^{\text{topo}}_{\text{candidate}} \cdot n^{\text{topo}}_{\text{candidate}}(t) = 360\,\text{B} \cdot n^{\text{topo}}_{\text{candidate}}(t), \tag{8.8}$$

where $n^{\text{topo}}_{\text{candidate}}$ is the average number of Topo candidates per event, which depends on the selection thresholds. Thresholds are optimized for highest average TOS efficiency over the input samples given the bandwidth limit of 3 GB/s. Table 8.3 shows the TOS efficiencies given these thresholds.

Table 8.3: TOS efficiencies for Topo classifiers. For a total efficiency, these are to be multiplied with the preselection efficiencies in 8.1.

Decay	Efficiency [%]
$B^0 \to K^+\pi^-$	92.2
$B^0 \to D^+D^-$	35.4
$B^0_s \to \phi\phi$	75.6
$B^0 \to \pi^+\pi^-\pi^0$	55.3
$B^0 \to K^*(892)e^+e^-$	56.0
$B^0 \to D^{*-}\mu^+\nu_\mu$	43.3
$B^0 \to D^{*-}(\tau^+ \to 3\pi\overline{\nu}_\tau)\nu_\tau$	26.2
$B^0 \to D^{*-}(\tau^+ \to \mu^+\nu_\mu\overline{\nu}_\tau)\nu_\tau$	26.8
$B^0_s \to D^-_s K^+$	53.1
$\Lambda^0_b \to \Lambda^+_c D^-_s$	38.0

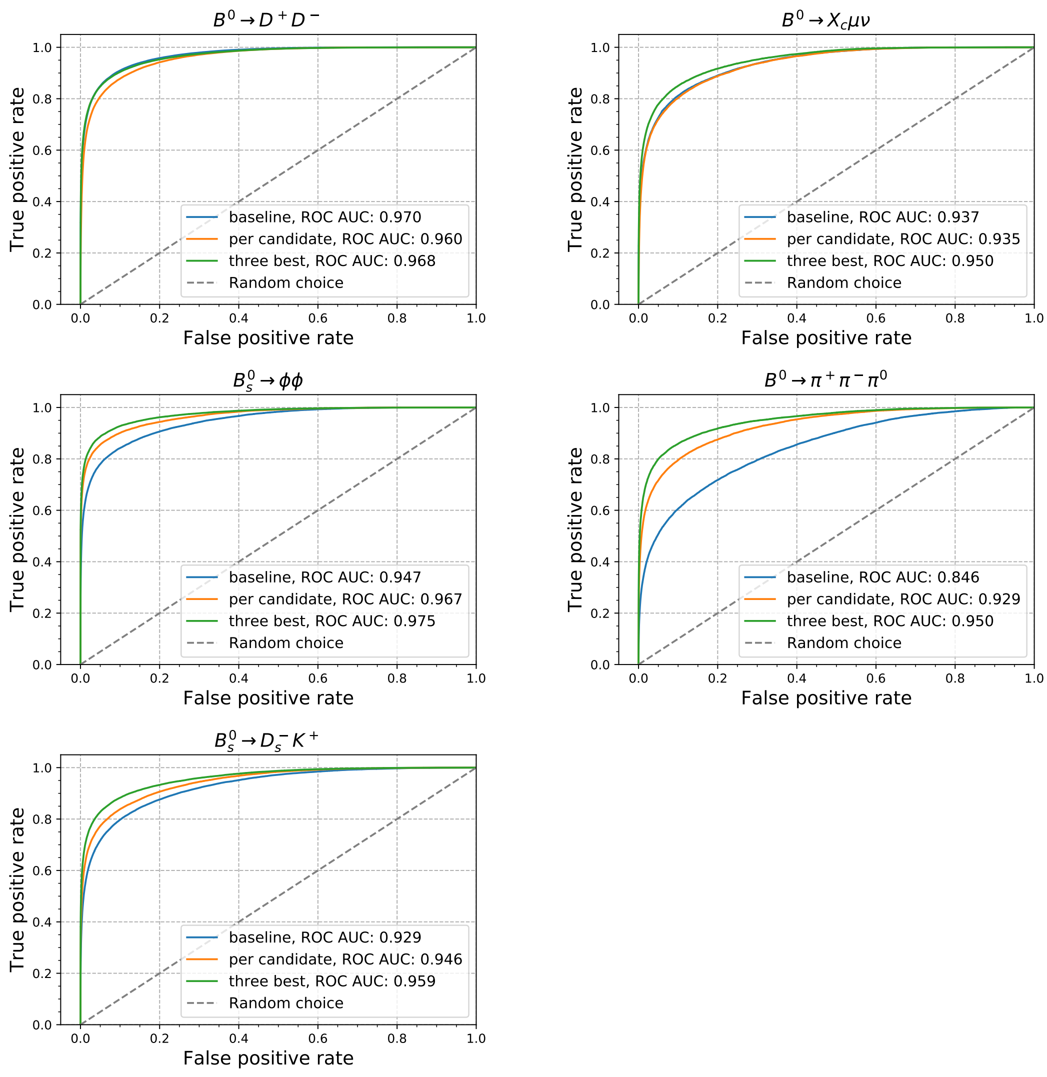


Figure 8.5: ROC curves and scores for the baseline Topo classifiers and the new models trained on the same input data. The plots differ only by the signal sample considered. Each plot considers the classification of one of the input samples against minimum bias background. Blue represents the baseline model. Orange and green curves represent two different training methods employed to train the new models. The blue and orange models are trained on a per-candidate basis.

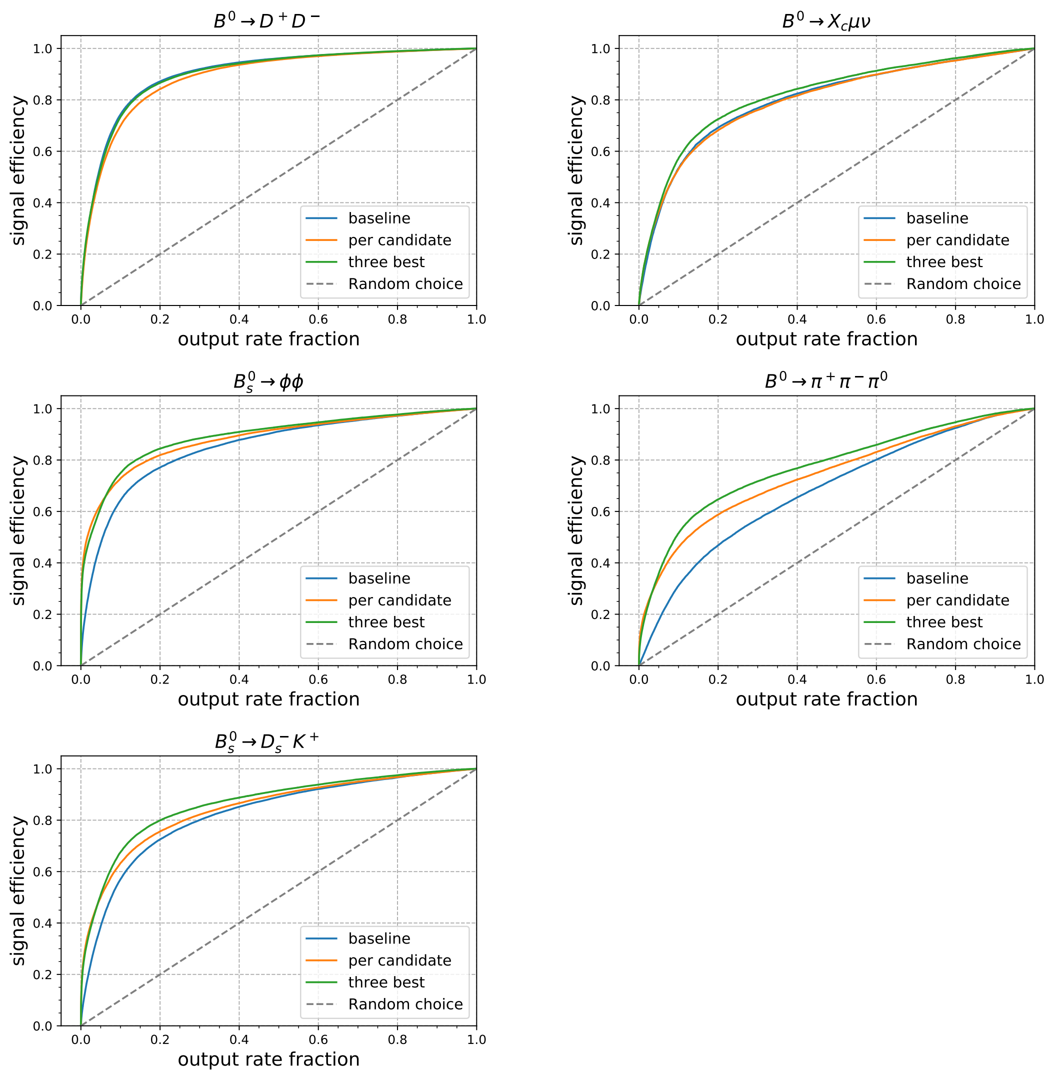


Figure 8.6: Signal efficiency against output event rate using combined two- and three-body event predictions. Signal efficiency is here defined as the number of events of the signal decay samples passing the selection. Background is defined by the fraction of minimum bias events that pass.

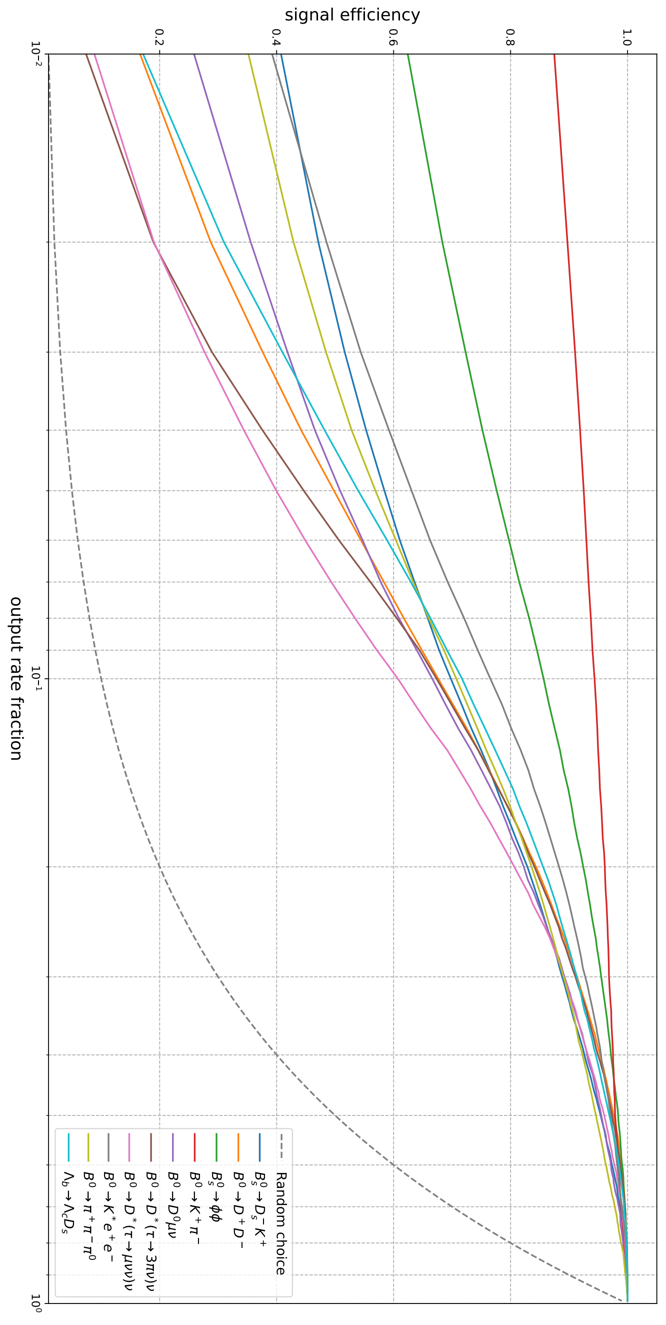


Figure 8.7: TOS signal efficiency against output event rate using combined two- and three-body event predictions for all input samples using the maximum of two- and three-body prediction. The x-axis is in logarithmic scale.

8.3 Selective Persistence in the topological trigger

The goal of this chapter is the reduction of persisted size of an event selected by the topological trigger. This can be achieved via Selective Persistence (SP) as opposed to the Full Persistence (FP). Selective Persistence in exclusive lines has been utilized for several operational years now. Line developers in Run 2 had the option to persist anything between only the candidate decay chain to the entire event, depending on physics analysis needs. Figure 8.8 sketches the possible choices and their impact on output bandwidth.

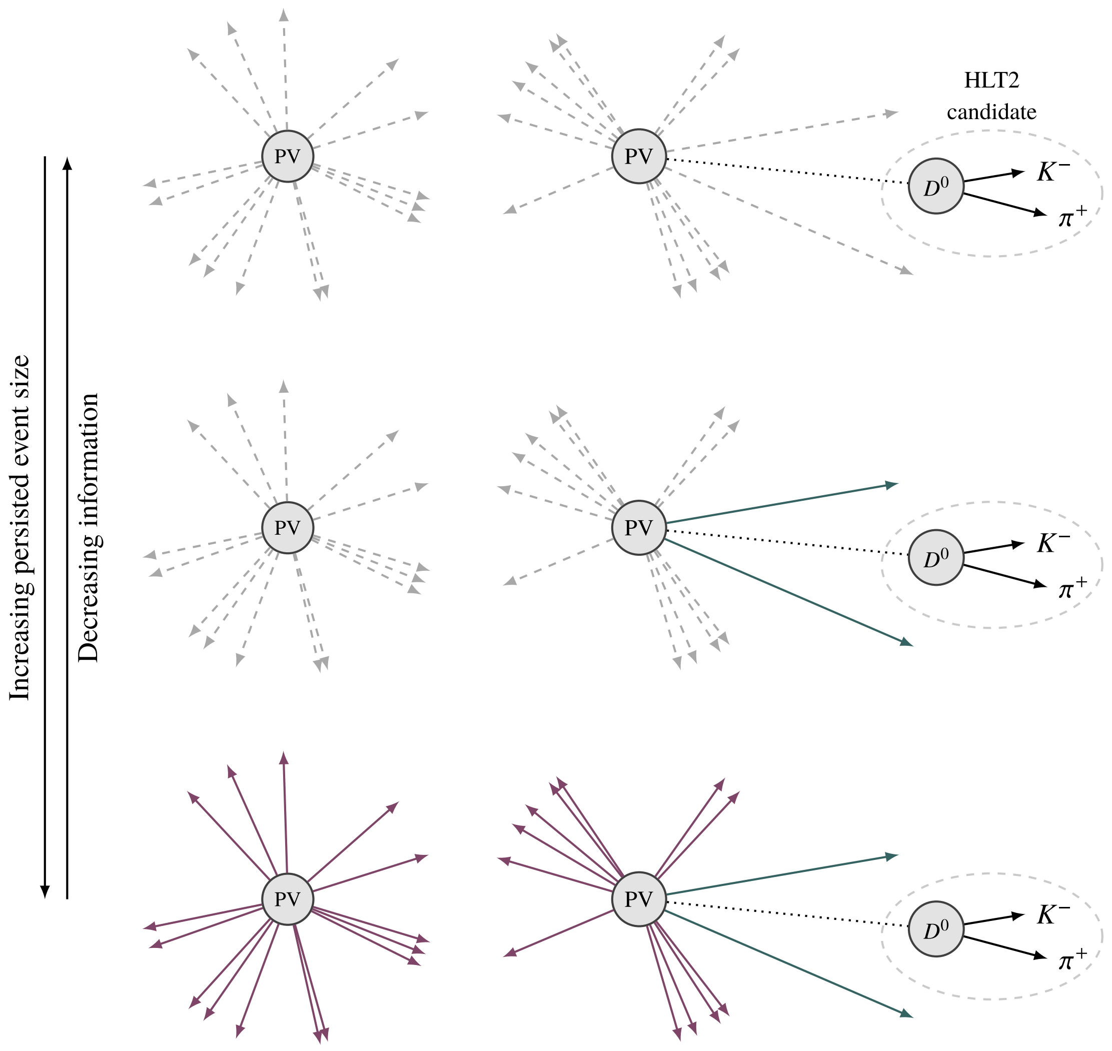


Figure 8.8: Different choices for signal event persistence. The minimum needed is the signal candidate and its children. One can optionally save more related information up to every signature in the event [86].

Employing SP on the Topo output is significantly harder for two reasons. Firstly, the inclusive nature of this line implies that it is supposed to be used for many different analyses. Therefore, a definition of relevant information for offline use might be too restrictive. Secondly, the decay signatures selected by the Topo are often more than the reconstructed two- and three-body candidates used for the actual selection. Unlike in an exclusive line, the decay chain itself is not specified.

To address the first problem, the following assumption will be made: Relevant information for offline analyses concerns trajectories that come from a beauty decay chain. This implies the tracks that come from the other beauty quark in the $b\bar{b}$ pair production. Due to the many degrees of freedom and unknowns in a pp-interaction, information that is not part of the $b\bar{b}$ pair final states is uncorrelated to the beauty decay chain and can therefore not contribute any more information to the signal decay. The approach of trying to select all trajectories coming from beauty decays automatically tackles the second problem as well, as these tracks are a superset of the tracks coming from the signal beauty hadron.

The problem now reduces to: How can one efficiently select these tracks? Two approaches are discussed here.

Selection based on primary vertex relations

The simplest idea utilizes the fact that in the typical upgrade event there are more than one primary interaction point. However, the $b\bar{b}$ pair of interest may only produce tracks that either extrapolate back to that PV very precisely or tracks that come from a displaced decay vertex. The sketch in Figure 8.8 displays this. As displacement of tracks is measured by minimum distance to any of the PVs, a problem arises when displaced tracks extrapolate towards a different from their actual PV by chance. Concretely, the tested selection criterion to select tracks is build as follows: Take any track that extrapolates back to the same primary vertex as the Topo candidate with a small impact parameter to that PV. Additionally, take all sufficiently displaced vertices as defined by the minimum impact parameter with respect to all PVs. The performance of this approach can be measured as every classification problem: A class probability is assigned based on the outlined logic. The signal probability is assigned as follows:

$$p_1 = \min(1, \delta(\mathrm{PV}(\mathrm{track}), \mathrm{PV}(\mathrm{candidate})) + \sqrt{\chi^2(\mathrm{IP})(\mathrm{track})/100}), \qquad (8.9)$$

where represents the Kronecker delta. The PV function yields the best PV for a given candidate or track. Conceptually this function should assign high values to prompt tracks from the same PV as the candidate and displaced tracks of all sorts. The scaling factor of 100 is chosen rather arbitrarily and defines the maximum χ^2 (IP) for which the prediction still differs. Everything above an IP of 100 mm will just be clipped to 1. Because a classification task only depends on differences in predictions rather than the absolute value, the specific choice for the scaling factor makes little to no difference as long as it is much larger than a common value for prompt tracks, because otherwise even prompt tracks might yield a high prediction. The square root as monotonic function makes no theoretical difference. It helps when binning the χ^2 distribution as is done for the performance plots.

With a floating point prediction and truth values for whether a track originates from a beauty decay, a signal efficiency vs. average retention fraction plot gives information about the prediction quality for all possible working points. The average retention fraction is estimated on minimum bias samples while the signal efficiency is based on the respective signal sample. These curves for the different decays in the input samples are shown in Figures 8.9. Note that the calculation of p_1 may be performed multiple times per track in each event, as it has to be done for every Topo candidate. For a per-track prediction, the maximum value over all Topo candidates in the event is chosen. Specifically, if two- and three-body selections yield Topo candidates in the same event, the maximum is taken over both types of candidates. Note also that events with more than one $b\bar{b}$ are discarded from the evaluation. The point of this study is to capture the tracks coming from the $b\bar{b}$ pair the Topo candidate refers to.

A high per-track efficiency might not be good enough if one of the tracks that was not selected carried the crucial information. Therefore, all performance plots in this section show a second curve made from a per-event prediction. By assigning an event the minimum prediction value of all signal tracks, we can estimate the fraction of events where all relevant tracks are being selected with a given threshold. Notably, tracks which are part of any Topo candidate are always selected. This explains the offset at the lower left of the performance plots.

Selection based on trained classification

A trained multivariate classifier usually performs better than a manually defined prediction as Equation (8.9). To train such a classifier, features from tracks and

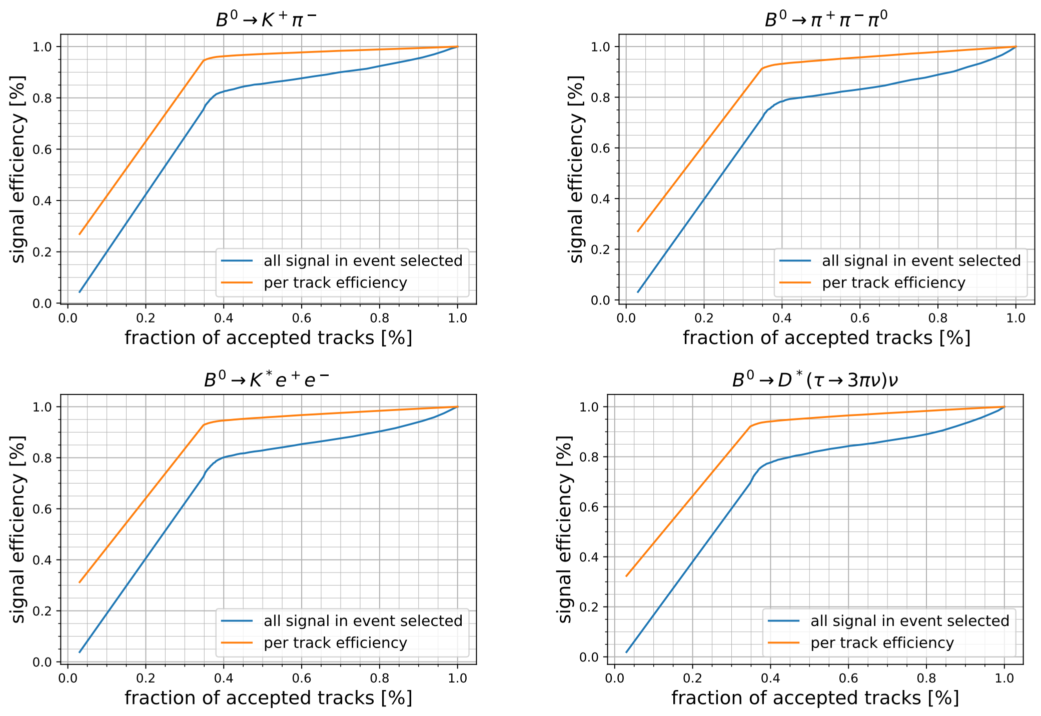


Figure 8.9: Prediction performance of p_1, visualized via long track acceptance fraction vs. efficiency for combined two- and three-body selections. The orange curve displays the fraction of tracks from beauty decay chains that are accepted by the classifier given a total track retention. The blue curve shows the fraction of events where all tracks from beauty decay chains were captured.

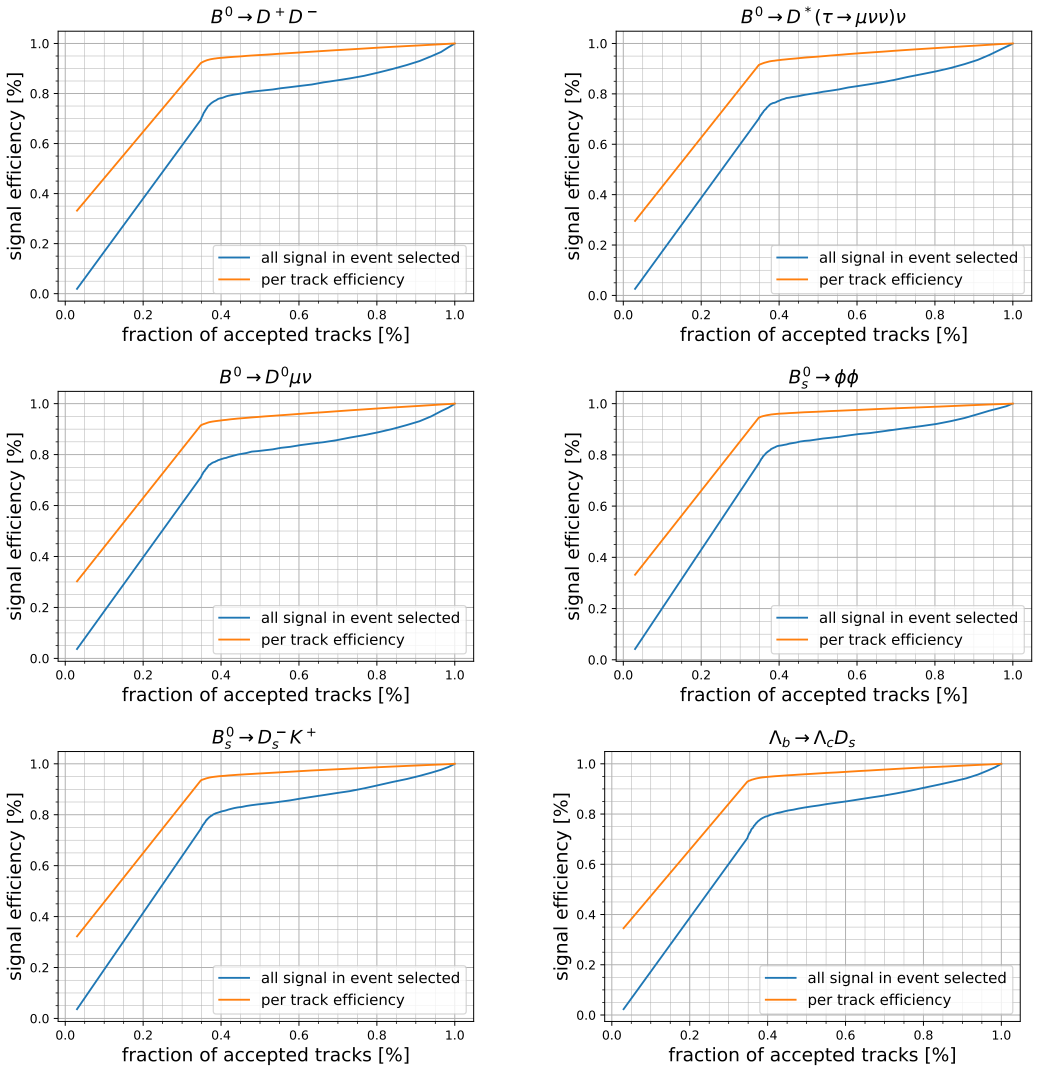


Figure 8.9: Prediction performance of p_1, visualized via long track acceptance fraction vs. efficiency for combined two- and three-body selections. The orange curve displays the fraction of tracks from beauty decay chains that are accepted by the classifier given a total track retention. The blue curve shows the fraction of events where all tracks from beauty decay chains were captured.

Topo candidates can be extracted to base a prediction p_2 on. A selection based solely on track features has briefly been explored to yield non-competitive results, and is therefore not further described here. Instead, the data is prepared by combining each Topo candidate that comes out of the topological trigger line and its own classifier with each track in an event. For each of these combinations, features to base the classification on can be determined. The relevant ones include features related to the candidate, the track, the combination of both and the primary vertices. Specifically, the features described in Table 8.4 are used for classification training. Note that two types of classifiers are trained, because features for two- and three-body Topo candidates come with different features used in the classifier. The gradient boosting implementation in LightGBM is used as it trains faster in a CPU environment with the type of data used here. Classification performances of Catboost and LightGBM implementations yield negligible differences. For final integration into the LHCb software stack, one may switch back to Catboost for easy translation to C++. All previously outlined measures to counter overtraining are taken here as well.

The performance plots for the combined predictions given by both classification algorithms is shown in Figures 8.10. The overall superiority of the trained classification is apparent. Using the shown method of selectively persisting tracks in the Topo trigger line enables one of two things: Either the Topo output rate can be increased to yield a higher overall signal efficiency or the freed up bandwidth may be used for other lines struggling with output rate control. For a 50% track retention working point the event loss would stay under 5% in all signal decay samples, even in the worst case scenario where only those events are usable in which all tracks from beauty decay chains have been captured by the classifier. One can use these bandwidth reductions to change thresholds of the Topo selection. The efficiency gain is visible in Table 8.3. Otherwise, the freed up bandwidth could be used to ease requirements on other important trigger lines. A global optimization has to be conducted to determine a good trade-off for all physics interests the trigger has to cover.

When removing 50% of the long tracks the Topo thresholds can be adjusted to increase the overall efficiencies of the signal decays, unless the freed up bandwidth is used in other trigger lines. Efficiencies for thresholds in both scenarios are shown in Table 8.5, once with and once without discarding 50% of the long tracks. The thresholds are optimized for highest TOS efficiency within the bandwidth limit.

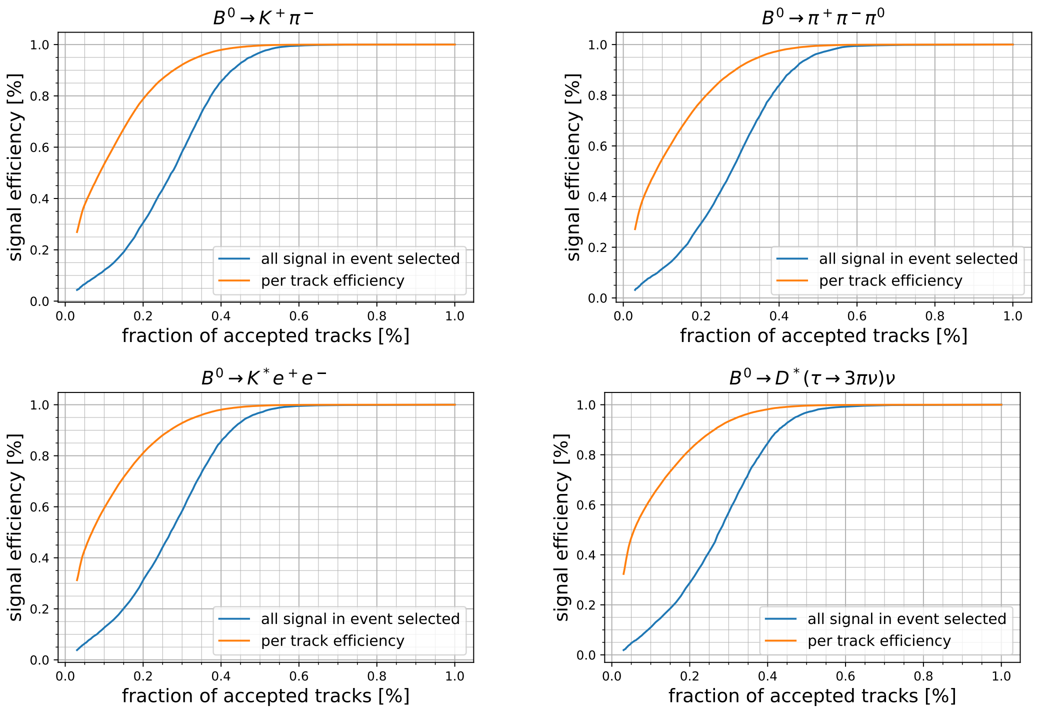


Figure 8.10: Prediction performance of p_2, equivalent to Figures 8.9.

Table 8.4: Input features to the track classification training

Feature	Explanation
Topo-Track-DOCA (χ^2)	Distance (χ^2) of closest approach between the candidate and the track
Topo-Track-P(T)	(Transverse) Momentum of the combined Topo and track candidate.
Topo-Track-η	Pseudorapidity of the combined Topo and track candidate.
Topo-Track-χ^2_{vtx}	χ^2_{vtx} of the Topo and track candidate if a vertex fit between them converged, otherwise an error value.
Topo-Track-IP (χ^2)	Impact Parameter (χ^2) between Topo candidate and track.
Topo-Track-$\mathit{\Delta} R^2$	$\mathit{\Delta}\eta^2 + \mathit{\Delta}\phi^2$ between Topo and track. ϕ is the angle in the x-y-plane of the detector
(2,3)-Body DOCA χ^2	Maximum distance of closest approach χ^2 between children.
(2,3)-Body FD χ^2	Measure for displacement of decay and primary vertex.
(2,3)-Body MIP	Minimum IP of Topo candidate with respect to PVs.
(2,3)-Body VZ	z-position of associated primary vertex.
(2,3)-Body χ^2_{vtx}	χ^2_{vtx} of the Topo candidate.
(2,3)-Body PRED	Topo classifcation prediction.
Track P(T)	Track (transverse) momentum.
Track η	Track pseudorapidity.
Track MIP (χ^2)	Impact Parameter (χ^2) between track and its associated primary vertex.
Track χ^2	χ^2 of the track fit.
Track GHOSTPROB	Predicted probability for the track being a ghost. This is determined by a neural network.

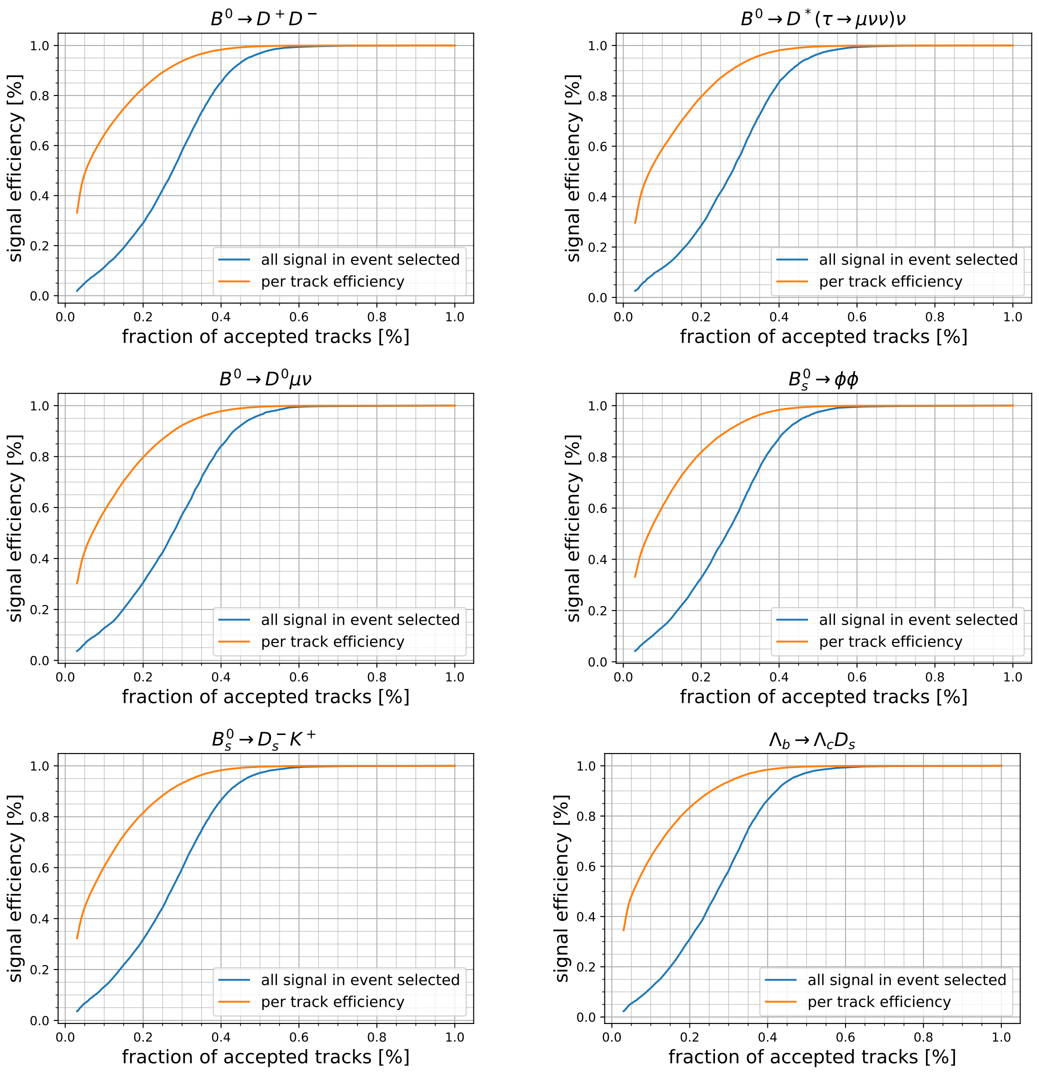


Figure 8.10: Prediction performance of p_2, equivalent to Figures 8.9.

Table 8.5: TOS efficiencies for Topo classifiers given different bandwidth scenarios. For a total efficiency, these are to be multiplied with the preselection efficiencies in 8.1. The left column is the same as in Table 8.3

Decay	Efficiency [%] @ 100% persistence	Efficiency [%] @ 50% long tracks
$B^0 \to K^+\pi^-$	92.2	93.2
$B^0 \to D^+D^-$	35.4	42.8
$B_s^0 \to \phi\phi$	75.6	79.9
$B^0 \to \pi^+\pi^-\pi^0$	55.3	62.1
$B^0 \to K^*(892)e^+e^-$	56.0	62.8
$B^0 \to D^{*-}\mu^+\nu_\mu$	43.3	50.9
$B^0 \to D^{*-}(\tau^+ \to 3\pi\overline{\nu}_\tau)\nu_\tau$	26.2	36.2
$B^0 \to D^{*-}(\tau^+ \to \mu^+\nu_\mu\overline{\nu}_\tau)\nu_\tau$	26.8	36.6
$B_s^0 \to D_s^-K^+$	53.1	58.0
$\mathit{\Lambda}_b^0 \to \mathit{\Lambda}_c^+D_s^-$	38.0	46.3

8.4 Summary and outlook

This Chapter presented an optimized two- and three-body Topo classification algorithm to efficiently select a broad range of signal decay channels. It optimizes TOS efficiency as opposed to overall signal efficiency, because analysts prefer TOS selections to be able to model backgrounds accurately.

The Topo traditionally persisted all reconstructed information in an event because the usual implementation of SP is dependent on a fully known reconstructed decay. In an inclusive line like the Topo, SP is much harder. One wants to generalize over multiple decay channels, so any SP approach has to do the same. To try to reduce the output bandwidth per retained event, an approach for SP using PV association was proposed. Because this approach shows significant losses even at a high track retention, a second algorithm was proposed to perform SP. A classification algorithm that predicts whether tracks come from a beauty decay chain is employed. This algorithm can discard 50% of all long tracks while retaining a close to 100% efficiency. With the freed up bandwidth, one can either enable other physics selections in the HLT2 or increase the efficiency of the Topo.

However, SP has only been tried on long tracks as these are the biggest contributors to bandwidth. The next step is to evaluate the given method on upstream tracks. As these also exhibit VELO tracks, the topological information close to the decay vertex can be used in the same way as with long tracks. Downstream tracks and neutral particles will need to be handled differently. For Downstream tracks one might select based on whether they combine to a K^0_S or a $\mathit{\Lambda}$ particle. Under the rough assumption that 50% of all particle and track types can be discarded with minimal loss, the efficiencies under the 3 GB/s bandwidth constraint would look as shown in Table 8.6. Larger efficiency improvements can be achieved in decays with a lower Topo efficiency. For instance, the efficiency of $B^0 \to D^{*-}(\tau^+ \to 3\pi\overline{\nu}_\tau)\nu_\tau$ decays could increase by almost a factor of two.

Table 8.6: TOS efficiencies for Topo classifiers for all scenarios. The first scenario assumes full persistence of all basics. The second scenario assumes that 50% of all long tracks are discarded. The rightmost scenario assumes that 50% of all basic particles can be removed with small loss.

Decay	Efficiency [%]	Efficiency [%]	Efficiency[%]
	@ 100% persist.	@ 50% long tracks	@ 50% all basics
$B^0 \to K^+\pi^-$	92.2	93.2	93.7
$B^0 \to D^+D^-$	35.4	42.8	51.4
$B_s^0 \to \phi\phi$	75.6	79.9	82.6
$B^0 \to \pi^+\pi^-\pi^0$	55.3	62.1	65.9
$B^0 \to K^*(892)e^+e^-$	56.0	62.8	68.5
$B^0 \to D^{*-}\mu^+\nu_\mu$	43.3	50.9	57.3
$B^0 \to D^{*-}(\tau^+ \to 3\pi\overline{\nu}_\tau)\nu_\tau$	26.2	36.2	48.3
$B^0 \to D^{*-}(\tau^+ \to \mu^+\nu_\mu\overline{\nu}_\tau)\nu_\tau$	26.8	36.6	46.7
$B_s^0 \to D_s^-K^+$	53.1	58.0	62.2
$\Lambda_b^0 \to \Lambda_c^+D_s^-$	38.0	46.3	55.6

9 Conclusion

LHCb is currently performing a significant detector upgrade involving a five-fold increase of instantaneous luminosity and the removal of the hardware trigger stage. The work presented in this thesis tackles some of the challenges implied by these changes. Specifically, two challenges are addressed: Firstly, the event throughput of the current HLT application is factors too low to be processing 30 MHz of input rate efficiently within the given budget. Secondly, LHCb needs to fit its broad physics program into an output bandwidth limit of 10 GB/s.

The first contribution described in this thesis is the implementation of the first feature-complete scheduling algorithm that is fast enough to run in the production environment without causing significant overheads to the application. Static scheduling logic is prepared before event evaluation and runtime decisions are limited to an absolute minimum. The algorithm can operate arbitrary control flow and implements a barrier concept to be able to share work between algorithm instances. This scheduling algorithm marks an important contribution to the first fully functioning HLT1 implementation. It is also the new baseline implementation for the Upgrade HLT2 solution.

The second contribution is the renewal of the selection framework in HLT2. Algorithms needed for decay chain reconstruction after trajectory reconstruction prove to be one of the major bottlenecks for the HLT2 application throughput. About 1000 different selection sequences will be using these algorithms in production. By overhauling the selection system and the particle event model, new implementations described in this thesis achieve up to 10-fold throughput improvements in decay chain reconstruction and particle selections. The transition to these algorithms in production will reduce the contribution of selection lines in HLT2 to under 5% relative contribution.

The third contribution tackles the second big challenge of the Upgrade HLT: The output bandwidth. With the constraint of 10 GB/s output bandwidth to permanent storage, many trigger lines have to transition to a selective persistence model. The Topological beauty selection is an inclusive trigger line with exceptionally high bandwidth requirements. Due to its inclusive nature it cannot apply the usual methods of selective persistence. Methods of optimizing the Topo selection to either lower bandwidth requirements or increase efficiency are explored. Selective persistence can be applied by classifying tracks based on whether they originate from a beauty flavored particle. At more than 95% worst-case efficiency, 50% of the long tracks in an event can be removed to lower output bandwidth or increase efficiency accordingly.

Summarized, crucial contributions for the operation of an efficient, fully software based HLT were presented. As such, they mark important milestones towards the collection of data corresponding to 50/fb integrated luminosity with unprecedented efficiencies in beauty and charm decay channels.

To showcase the statistical power of the dataset to be collected in the next 10 years, we compare efficiencies and production rates in several channels that were considered throughout the thesis. Statistical uncertainties in measurements at LHCb usually scale roughly with the square root of the number of signal events retained. The 50/fb dataset already increases the amount of produced beauty and charm hadrons by more than a factor of 6 with respect to the Run 1 and 2 datasets. For muonic channels like $B^0 \to K^*\mu^+\mu^-$ and $B^0 \to D^{*-}\mu^+\nu$ which exhibit high efficiencies in both datasets, this corresponds to a factor of at least $\sqrt{6}$ improvement in statistical sensitivity.

For channels involving electrons, like $B^0 \to K^*e^+e^-$, the sensitivity is further improved due to the increased efficiency that the hardware trigger removal yields. In this case, the efficiency with and without the hardware trigger differs by about a factor of two. The efficiency increase is most pronounced in channels with purely hadronic final states like $B_s^0 \to \phi\phi$, $B^0 \to D^+D^-$ and $B^0 \to D^{*-}(\tau^+ \to 3\pi\overline{\nu}_\tau)\nu_\tau$, where improvements of up to factors of 4 due to the removal of the hardware trigger are expected. This translates to another factor of $\sqrt{2} - 2$ in sensitivity.

The optimization of the Topo further increases the signal efficiency by itself, even more so if the bandwidth reduction due to selective persistence is utilized to retain a higher rate of events. This leads to further efficiency improvements of up to 50 %.

To complete the Upgrade trigger, several open tasks remain. Most importantly, the HLT2 reconstruction has to be sped up by about a factor of two to be able to process the HLT1 output rate. Secondly, the new selection framework has to be completed and validated carefully for different decay scenarios to ensure correctness. While there are several other details to be solved in both HLT1 and HLT2, I am confident that the LHCb collaboration will successfully operate the fully software based upgrade trigger as soon as the LHC starts again.

A Appendix

A.1 Description of HLT1 trigger lines

This Section describes the HLT1 trigger lines used as preselection to the topological trigger presented in Chapter 8.

Hlt1(Two)TrackMVAVLoose share a preselection of particles with these requirements:

- before track fit:
  - PT $>$ 350 MeV & P $>$ 4750 MeV
- after track fit:
  - PT $>$ 400 MeV & P $>$ 5 GeV &
    χ^2 (Track) $<$ 2.5 & GhostProb $<$ 999

Hlt1TrackMVAVLoose continues with a single track based selection from the shared preselection:

- GhostProb $<$ 0.2 &
- ((PT $>$ 26 GeV & χ^2 (IP)$>$ 7.4) |
  ((2 GeV $<$ PT $<$ 26 GeV) &
  (log(χ^2 (IP)) $> \frac{1}{\mathrm{PT/GeV}-2}^2 + \frac{1.248}{26\,\mathrm{GeV}} \cdot (26\,\mathrm{GeV} - \mathrm{PT}) + \log(7.4)$))))

Hlt1TwoTrackMVAVLoose continues by vertexing two tracks pairwise:

- χ^2 (IP)$>$ 4
- vertex $\chi^2 < 10$ & $\eta \in [2, 5]$ & CORRM $>$ 1000 & cos(DIRA) $>$ 0
- MVA ($>$0.96) based on
  - vertex χ^2
  - Flight distance χ^2

  - the sum of PT from basic children
  - the displacement of basic daughters

Hlt1TrackMuonMVA is a specialized line for reconstructing muons. It is similar to Hlt1TrackMVAVLoose but it has a much lower rate due to the muon requirement:

- PT $>$ 1 GeV & P $>$ 5 GeV &
- χ^2 (Track) $<$ 2.5 & GhostProb $<$ 0.2 &
- IsMuon
- ((PT $>$ 25 GeV & χ^2 (IP)$>$ 7.4) |
  ((PT $<$ 25 GeV) &
  $(\log(\chi^2\ (\mathrm{IP})) > \frac{1}{\mathrm{PT/GeV}-1}^2 + \frac{1.2}{25\,\mathrm{GeV}} \cdot (25\,\mathrm{GeV} - \mathrm{PT}) + \log(7.4))))$

A.2 Additional information on input samples for the Topo

Table A.1 shows the number of simulated events for each decay used in Chapter 8. They were produced with simulation version sim-09c-up02 and selected with the HLT1 configuration as defined in TCK Trig0x52000000. They were simulated with center of mass energy $\sqrt{s} = 14\,\text{TeV}$, average number of collisions per event $\nu = 7.6$ and 25 ns bunch spacing with the Pythia version 8.

Table A.1: Event counts for input samples

Decay	Number of events
$B^0 \to K^+\pi^-$	124404
$B^0 \to D^+D^-$	277834
$B_s^0 \to \phi\phi$	119393
$B^0 \to \pi^+\pi^-\pi^0$	119986
$B^0 \to K^*(892)e^+e^-$	121372
$B^0 \to D^{*-}\mu^+\nu_\mu$	89423
$B^0 \to D^{*-}(\tau^+ \to 3\pi\overline{\nu}_\tau)\nu_\tau$	109586
$B^0 \to D^{*-}(\tau^+ \to \mu^+\nu_\mu\overline{\nu}_\tau)\nu_\tau$	113543
$B_s^0 \to D_s^-K^+$	126719
$\Lambda_b^0 \to \Lambda_c^+D_s^-$	124458

Bibliography

[1] Lyndon Evans and Philip Bryant. "LHC Machine." In: *JINST* 3 (2008). DOI: 10.1088/1748-0221/3/08/S08001.

[2] CMS Collaboration. "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].

[3] ATLAS Collaboration. "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].

[4] S. L. Glashow. "Partial-symmetries of weak interactions." In: *Nuclear Physics* 22.4 (1961), pp. 579–588.

[5] A. Salam and J.C. Ward. "Electromagnetic and weak interactions." In: *Physics Letters* 13.2 (1964), pp. 168–171.

[6] S. Weinberg. "A Model of Leptons." In: *Phys. Rev. Lett.* 19 (21 1967), pp. 1264–1266.

[7] M. Gell-Mann. *The Eightfold Way: A Theory Of Strong Interaction Symmetry.* 1961.

[8] G. Bertone, D. Hooper, and J. Silk. "Particle dark matter: evidence, candidates and constraints." In: *Physics Reports* 405.5 (2005), pp. 279–390.

[9] S. Dimopoulos and L. Susskind. "Baryon asymmetry in the very early universe." In: *Physics Letters B* 81.3 (1979), pp. 416–418.

[10] A. D. Sakharov. "Violation of Cp-Invariance C-Asymmetry and Baryon Asymmetry of the Universe." In: ed. by Y. A. Trutnev. World Scientific Publishing Co, 1998, pp. 84–87.

[11] LHCb Collaboration. *LHCb Trigger and Online Upgrade Technical Design Report.* Tech. rep. CERN-LHCC-2014-016. 2014. URL: https://cds.cern.ch/record/1701361.

[12] R. Aaij et al. (LHCb collaboration). “Observation of the Resonant Character of the $Z(4430)^-$ State.” In: *Phys. Rev. Lett.* 112 (22 2014), p. 222002.

[13] LHCb Collaboration. “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]`.

[14] Q. R. et al Ahmad (SNO collaboration). “Measurement of the Rate of $\nu_e + d \to p + p + e^-$ Interactions Produced by ^8B Solar Neutrinos at the Sudbury Neutrino Observatory.” In: *Phys. Rev. Lett.* 87 (7 2001), p. 071301.

[15] Q. R. et al Ahmad (SNO collaboration). “Direct Evidence for Neutrino Flavor Transformation from Neutral-Current Interactions in the Sudbury Neutrino Observatory.” In: *Phys. Rev. Lett.* 89 (1 2002), p. 011301.

[16] Y. Fukuda et al. (Super-Kamiokande collaboration). “Evidence for Oscillation of Atmospheric Neutrinos.” In: *Phys. Rev. Lett.* 81 (8 1998), pp. 1562–1567.

[17] *Why is the Higgs discovery so significant?* visited on 13.05.2020. URL: `www.stfc.ac.uk/research/particle-physics-and-particle-astrophysics/peter-higgs-a-truly-british-scientist/why-is-the-higgs-discovery-so-significant`.

[18] R. Aaij et al. (LHCb collaboration). “Measurement of the b-quark production cross-section in 7 and 13 TeV pp collisions.” In: *Phys. Rev. Lett.* 118.5 (2017), p. 052002. arXiv: `1612.05140 [hep-ex]`.

[19] R Aaij et al. “Prompt charm production in pp collisions at sqrt(s)=7 TeV.” In: *Nucl. Phys. B* 871 (2013), pp. 1–20. DOI: `10.1016/j.nuclphysb.2013.02.010`. arXiv: `1302.2864 [hep-ex]`.

[20] R. Aaij et al. “Erratum to: Measurements of prompt charm production cross-sections in pp collisions at $\sqrt{s} = 13$ TeV.” In: *Journal of High Energy Physics* 2016 (Sept. 2016). DOI: `10.1007/JHEP09(2016)013`.

[21] Michael Dine and Alexander Kusenko. “The Origin of the matter - antimatter asymmetry.” In: *Rev. Mod. Phys.* 76 (2003), p. 1. DOI: `10.1103/RevModPhys.76.1`. arXiv: `hep-ph/0303065`.

[22] Roel Aaij et al. “Measurement of CP violation in the $B_s^0 \to \phi\phi$ decay and search for the $B^0 \to \phi\phi$ decay.” In: *JHEP* 12 (2019), p. 155. DOI: `10.1007/JHEP12(2019)155`. arXiv: `1907.10003 [hep-ex]`.

[23] Roel Aaij et al. "Measurement of CP violation in $B^0 \to D^+D^-$ decays." In: *Phys. Rev. Lett.* 117.26 (2016), p. 261801. DOI: `10.1103/PhysRevLett.117.261801`. arXiv: `1608.06620 [hep-ex]`.

[24] R. Aaij et al. "Search for Lepton-Universality Violation in $B^+ \to K^+l^+l^- Decays$." In: *Physical Review Letters* 122.19 (May 2019). ISSN: 1079-7114. DOI: `10.1103/physrevlett.122.191801`. URL: `http://dx.doi.org/10.1103/PhysRevLett.122.191801`.

[25] R. Aaij et al. (LHCb collaboration). "Test of lepton universality with $B^0 \to K^{*0}\ell^+\ell^-$ decays." In: *JHEP* 08 (2017), p. 055. arXiv: `1705.05802 [hep-ex]`.

[26] R. Aaij et al. (LHCb collaboration). "Measurement of the ratio of branching fractions $\mathcal{B}(\bar{B}^0 \to D^{*+}\tau^-\bar{\nu}_\tau)/\mathcal{B}(\bar{B}^0 \to D^{*+}\mu^-\bar{\nu}_\mu)$." In: *Phys. Rev. Lett.* 115.11 (2015), p. 111803. arXiv: `1506.08614 [hep-ex]`.

[27] R. Aaij et al. (LHCb collaboration). "Measurement of the ratio of the $B^0 \to D^{*-}\tau^+\nu_\tau$ and $B^0 \to D^{*-}\mu^+\nu_\mu$ branching fractions using three-prong τ-lepton decays." In: (2017). arXiv: `1708.08856 [hep-ex]`.

[28] C Lefevre. "LHC: the guide (English version). Guide du LHC (version anglaise)." 2009. URL: `https://cds.cern.ch/record/1165534`.

[29] ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider." In: *JINST* 3.08 (2008), S08003–S08003. DOI: `10.1088/1748-0221/3/08/s08003`.

[30] CMS Collaboration. "The CMS experiment at the CERN LHC." In: *JINST* 3.08 (2008), S08004–S08004. DOI: `10.1088/1748-0221/3/08/s08004`.

[31] LHCb Collaboration. "The LHCb Detector at the LHC." In: *JINST* 3 (2008). DOI: `10.1088/1748-0221/3/08/S08005`.

[32] *bb production angle plots.* URL: `https://lhcb.web.cern.ch/lhcb/speakersbureau/html/bb_ProductionAngles.html` (visited on 08/08/2016).

[33] LHCb Collaboration. *LHCb reoptimized detector design and performance: Technical Design Report.* Tech. rep. CERN-LHCC-2003-030. 2003. URL: `https://cds.cern.ch/record/630827`.

[34] LHCb Collaboration. *LHCb VELO Upgrade Technical Design Report.* Tech. rep. CERN-LHCC-2013-021. 2013. URL: `https://cds.cern.ch/record/1624070`.

[35] LHCb Collaboration. *LHCb Tracker Upgrade Technical Design Report.* Tech. rep. CERN-LHCC-2014-001. 2014. URL: `https://cds.cern.ch/record/1647400`.

[36] LHCb Collaboration. *LHCb PID Upgrade Technical Design Report.* Tech. rep. CERN-LHCC-2013-022. 2013. URL: `https://cds.cern.ch/record/1624074`.

[37] CMS & LHCb Collaboration. "Observation of the rare $B_s^0 \to \mu^+\mu^-$ decay from the combined analysis of CMS and LHCb data." In: *Nature* 522 (2015), pp. 68–72. DOI: `10.1038/nature14474`. arXiv: `1411.4413 [hep-ex]`.

[38] N. Scharmberg. *Search for the rare decay B to e+ e- at the LHCb detector.* M.Sc. Thesis, Technische Universität Dortmund.

[39] ISO. *ISO/IEC 14882:2017 Information technology — Programming languages — C++.* Fifth. 2017, p. 1605. URL: `https://www.iso.org/standard/68564.html`.

[40] Guido Van Rossum and Fred L Drake Jr. *Python tutorial.* Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[41] *The Gaudi project.* URL: `https://gitlab.cern.ch/gaudi/Gaudi`.

[42] *The LHCb project.* URL: `https://gitlab.cern.ch/lhcb/LHCb`.

[43] *The Online project.* URL: `https://gitlab.cern.ch/lhcb/Online`.

[44] *The Rec project.* URL: `https://gitlab.cern.ch/lhcb/Rec`.

[45] *The Brunel project.* URL: `https://gitlab.cern.ch/lhcb/Brunel`.

[46] *The Phys project.* URL: `https://gitlab.cern.ch/lhcb/Phys`.

[47] *The DaVinci project.* URL: `https://gitlab.cern.ch/lhcb/DaVinci`.

[48] *The Gauss project.* URL: `https://gitlab.cern.ch/lhcb/Gauss`.

[49] Torbjörn Sjöstrand, Stephen Mrenna, and Peter Skands. "PYTHIA 6.4 physics and manual." In: *Journal of High Energy Physics* 2006.05 (2006), pp. 026–026. DOI: `10.1088/1126-6708/2006/05/026`. URL: `https://doi.org/10.1088%2F1126-6708%2F2006%2F05%2F026`.

[50] David J. Lange. "The EvtGen particle decay simulation package." In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 462.1 (2001). BEAUTY2000, Proceedings of the 7th Int. Conf. on B-Physics at Hadron Machines, pp. 152–155. ISSN: 0168-9002. DOI: `https://doi.org/10.1016/S0168-9002(01)00089-4`. URL: `http://www.sciencedirect.com/science/article/pii/S0168900201000894`.

[51] *The Boole project.* URL: `https://gitlab.cern.ch/lhcb/Boole`.

[52] *The Allen project.* URL: `https://gitlab.cern.ch/lhcb/Allen`.

[53] J Borel et al. *The Bs -> Ds pi and Bs -> Ds K selections.* Tech. rep. LHCb-2007-017. CERN-LHCb-2007-017. LPHE Note 2007-03. Geneva: CERN, 2007. URL: `http://cds.cern.ch/record/1027381`.

[54] LHCb Collaboration. *Framework TDR for the LHCb Upgrade: Technical Design Report.* Tech. rep. CERN-LHCC-2012-007. 2012. URL: `https://cds.cern.ch/record/1443882`.

[55] J Albrecht, V V Gligorov, and G Raven. *Review Document: Full Software Trigger.* Tech. rep. LHCb-PUB-2014-036. CERN-LHCb-PUB-2014-036. On behalf of the the HLT software group. Geneva: CERN, May 2014. URL: `http://cds.cern.ch/record/1700271`.

[56] LHCb Collaboration. *Letter of Intent for the LHCb Upgrade.* Tech. rep. CERN-LHCC-2011-001. 2011. URL: `https://cds.cern.ch/record/1333091`.

[57] M Williams et al. *Trigger selections for the LHCb upgrade.* Tech. rep. LHCb-PUB-2014-031. CERN-LHCb-PUB-2014-031. Geneva: CERN, Mar. 2014. URL: `http://cds.cern.ch/record/1670992`.

[58] Christoph Hasse. "Alternative approaches in the event reconstruction of LHCb." Presented 12 Dec 2019. 2019. URL: `https://cds.cern.ch/record/2706588`.

[59] LHCb Collaboration. "Search for Dark Photons Produced in 13 TeV *pp* Collisions." In: *Phys. Rev. Lett.* 120.6 (2018), p. 061801. DOI: `10.1103/PhysRevLett.120.061801`. arXiv: `1710.02867 [hep-ex]`.

[60] LHCb Collaboration. "Design and performance of the LHCb trigger and full real-time reconstruction in Run 2 of the LHC." In: *JINST* 14.04 (2019), P04013. DOI: `10.1088/1748-0221/14/04/P04013`. arXiv: `1812.10790 [hep-ex]`.

[61] A. Apollonio et al. “Reliability and Availability of Particle Accelerators: Concepts, Lessons, Strategy.” In: *Proc. 9th International Particle Accelerator Conference (IPAC'18), Vancouver, BC, Canada, April 29-May 4, 2018* (Vancouver, BC, Canada). International Particle Accelerator Conference 9. https://doi.org/10.18429/JACoW-IPAC2018-FRXGBD1. Geneva, Switzerland: JACoW Publishing, 2018, pp. 5014–5018. ISBN: 978-3-95450-184-7. DOI: `doi:10.18429/JACoW-IPAC2018-FRXGBD1`. URL: `http://jacow.org/ipac2018/papers/frxgbd1.pdf`.

[62] *LHCb trigger conference diagrams.* visited on 25.05.2020. URL: `https://twiki.cern.ch/twiki/bin/view/LHCb/LHCbTriggerConferenceDiagramsPlots`.

[63] CERN (Meyrin) LHCb Collaboration. *Computing Model of the Upgrade LHCb experiment.* Tech. rep. CERN-LHCC-2018-014. LHCB-TDR-018. Geneva: CERN, 2018. URL: `http://cds.cern.ch/record/2319756`.

[64] Roel Aaij et al. *Upgrade trigger: Biannual performance update.* Tech. rep. CERN-LHCb-PUB-2017-005. 2017. URL: `https://cds.cern.ch/record/2244312`.

[65] *What is vectorization?* URL: `https://lappweb.in2p3.fr/~paubert/ASTERICS_HPC/6-6-1-985.html`.

[66] Karl Rupp. *42 Years of Microprocessor Trend Data.* URL: `https://github.com/karlrupp/microprocessor-trend-data/tree/master/42yrs` (visited on 06/13/2019).

[67] LHCb Collaboration. *Upgrade Software and Computing.* Tech. rep. CERN-LHCC-2018-007. 2018. URL: `https://cds.cern.ch/record/2310827`.

[68] M Clemencic, B Hegner, and C Leggett. “Gaudi evolution for future challenges.” In: *J. Phys. : Conf. Ser.* 898.4 (2017), 042044. 3 p. DOI: `10.1088/1742-6596/898/4/042044`. URL: `https://cds.cern.ch/record/2297285`.

[69] I. Shapoval et al. “Graph-based decision making for task scheduling in concurrent Gaudi.” In: *2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC).* 2015, pp. 1–3.

[70] Niklas Nolte et al. “A new scheduling algorithm for the LHCb upgrade trigger application.” In: *Journal of Physics: Conference Series* 1525 (Apr. 2020), p. 012052. DOI: `10.1088/1742-6596/1525/1/012052`. URL: `https://doi.org/10.1088%2F1742-6596%2F1525%2F1%2F012052`.

[71] James E. Kelley. “Critical-Path Planning and Scheduling: Mathematical Basis.” In: *Oper. Res.* 9.3 (1961), pp. 296–320. ISSN: 0030-364X. DOI: 10.1287/opre.9.3.296. URL: https://doi.org/10.1287/opre.9.3.296.

[72] M Clemencic et al. “Preparing HEP software for concurrency.” In: *Journal of Physics: Conference Series* 513 (June 2014), p. 052028. DOI: 10.1088/1742-6596/513/5/052028.

[73] *Intel VTune Amplifier.* URL: https://software.intel.com/en-us/vtune.

[74] *LoKi.* URL: https://lhcb-comp.web.cern.ch/Analysis/Loki/doc.html.

[75] *CPPYY.* URL: https://cppyy.readthedocs.io/en/latest/.

[76] *Cling - The interactive C++ Interpreter.* URL: https://github.com/root-project/cling.

[77] “Throughput and resource usage of the LHCb upgrade HLT.” In: (Apr. 2020). URL: https://cds.cern.ch/record/2715210.

[78] *Intel Intrinsics Guide.* URL: https://software.intel.com/sites/landingpage/IntrinsicsGuide.

[79] Vladislav Belavin. “TurboSP and the Topological Trigger.” In: (Aug. 2016). URL: https://cds.cern.ch/record/2281011.

[80] Particle Data Group et al. “Review of Particle Physics.” In: *Progress of Theoretical and Experimental Physics* 2020.8 (2020). 083C01. ISSN: 2050-3911. DOI: 10.1093/ptep/ptaa104. eprint: https://academic.oup.com/ptep/article-pdf/2020/8/083C01/33653179/ptaa104.pdf. URL: https://doi.org/10.1093/ptep/ptaa104.

[81] *Kaggle.* URL: https://www.kaggle.com/.

[82] A Höcker et al. “TMVA - Toolkit for Multivariate Data Analysis with ROOT:Users guide.” In: (2010).

[83] Alejandro Alfonso Albero et al. *Upgrade trigger selection studies.* Tech. rep. LHCb-PUB-2019-013. CERN-LHCb-PUB-2019-013. Geneva: CERN, Sept. 2019. URL: https://cds.cern.ch/record/2688423.

[84] Anna Dorogush, Vasily Ershov, and Andrey Gulin. “CatBoost: gradient boosting with categorical features support.” In: (2018).

[85] Guolin Ke et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree.” In: *NIPS.* 2017.

[86] R. Aaij et al. “A comprehensive real-time analysis model at the LHCb experiment.” In: *Journal of Instrumentation* 14.04 (Apr. 2019), P04006–P04006. DOI: `10.1088/1748-0221/14/04/p04006`. URL: `https://doi.org/10.1088%2F1748-0221%2F14%2F04%2Fp04006`.

Acknowledgements

This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 05E15CHA).

I would like to thank Prof. Johannes Albrecht for making this thesis and going to CERN possible. Thank you for the fruitful supervision and the time you spent helping me! I would also like to thank Dr. Johannes Erdmann for taking the time to be the second assessor.

My biggest gratitude goes to Dr. Sascha Stahl. You always made the time to help, discuss, guide and assist me in every other way until the very end of the doctoral period. I cannot have hoped for a better supervisor. You rock!

I also express my thanks to all the awesome workmates that made the time at CERN truly pleasurable, with whom it was a enjoyable and instructive time to work with, and who helped me with all my annoying inquiries. In no particular order: Thank you, Alex, Rosen, Olli, Dominik, Laurent, Claire, Daniel and specifically my office buddy/girlfriend Christoph!

Next up, a big shout out to my hometown friends. It is a great comfort to know that I can count on every one of you! Thank you Baran, Faber, Kevin and Stefan!

Lastly, of course, I want to express my deepest gratitude to my family. Thank you, Vic, Mama, Papa and Mormor! To you I owe everything.