LHCb-PUB-2018-005
May 7, 2018

Upgrade trigger & reconstruction strategy: 2017 milestone

J. Albrecht[1], P. Billoir[8], D. Campora[2,9], M. Cattaneo[2], M. Chefdeville[14] M. Clemencic[2], B. Couturier[2], A. Dziurda[2], C. Fitzpatrick[3], M. Fontana[2,4], L. Grillo[10], C. Hasse[1,2], D. Hill[5], C. Jones[6], F. Lemaitre[2], O. Lupton[2], R. Matev[2], A. Pearce[2], F.Polci[8], L.Promberger[2], S. Ponce[2], R. Quagliani[8], G. Raven[11], B. Sciascia[7], M. Schiller[12], S. Stahl[2], M. Szymanski[13]

[1] *TU Dortmund,* [2] *CERN,* [3] *EPFL Lausanne,* [4] *Cagliari,* [5] *Oxford,* [6] *Cambridge,* [7] *LNF Frascati,* [8] *LPNHE,* [9] *Universidad de Sevilla,* [10] *University of Manchester,* [11] *NIKHEF,* [12] *University of Glasgow,* [13] *University of Chinese Academy of Sciences,* [14] *LAPP Annecy*

Abstract

The LHCb collaboration is currently preparing an update of the experiment to take data in Run 3 of the LHC. The dominant feature of this upgrade is a trigger-less readout of the full detector followed by a full software trigger. To make optimal use of the collected data, the events are reconstructed at the inelastic collision rate of 30 MHz. This document presents the baseline trigger and reconstruction strategy as of the end of 2017.

Contents

1 Introduction

This document presents the baseline trigger and reconstruction strategy as of the end of 2017. The reconstruction as implemented in Run 2 is presented first to set a baseline, followed by the developments that yield towards the Run 3 reconstruction sequence. Differences between Run 2 and Run 3 are emphasised and the current algorithmic implementation is gives. The document is structured as follows:

Chapter 2 gives an overview of the Run 2 reconstruction. It begins witha discussion of the tracking sequence, followed by the particle identification sequence. Chapter 3 then introduces the *functional framework* in which the upgrade reconstruction algorithms are implemented. Chapter 4 gives a short overview of the requirements on the evnet model. Chapter 5 then discusses the Run 3 sequence, structured in tracking and PID parts. Chapter 6 then discusses the trigger strategy for Run 3, including the selection framework and persistency.

2 Reconstruction overview: Run II

2.1 Tracking

The LHCb tracking system consists of three detectors: the VELO, the TT and the T stations. Information from these detectors is used to reconstruct the following types of tracks:

- **VELO tracks**,
  are made using the VELO detector and are used in finding primary vertices.
- **Upstream tracks**,
  are reconstructed from VELO tracks and TT hits. These tend to be low momentum particles which are then swept out of the LHCb acceptance by the magnet.
- **T tracks**,
  are determined from hits in the T stations, sometimes referred to as a **Seed tracks**.
- **Downstream tracks**,
  have both TT and T station hits, but do not use the VELO detector. This type of track is important in the reconstruction of daughters from long-lived particles such as K^0_S or $\mathit{\Lambda}$, which often decay after leaving the VELO detector.
- **Long (Forward, Match) tracks**,
  have signatures in at least the VELO and T stations. Since these tracks pass through the magnetic field, they have the most accurately measured momentum. Depending on the reconstruction algorithm they are called either **Forward** or **Match** tracks.

Due to their properties downstream and long tracks are the most useful for physics analyses. The different types of tracks are presented in Fig. 1.

The track reconstruction consists of three main parts. First, the signatures produced by charged particles in the detector, tracks, are found by pattern recognition algorithms. Second, all of the found tracks fit using a Kalman filter which obtains the best possible estimate of the true trajectory with corrections due to energy loss and multiple scattering.

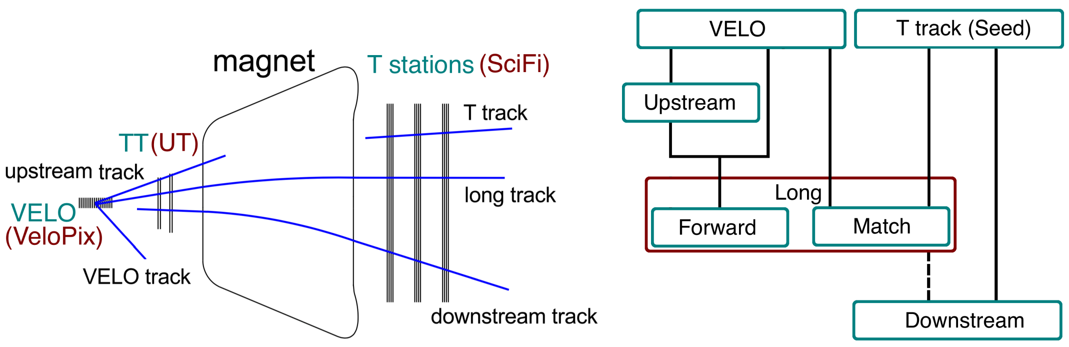


Figure 1: Left: different types of tracks in the LHCb experiment. The tracking system subdetectors are indicated in a cyan (red) for Run II (Upgrade), while the track types are shown as blue lines. Right: dependencies among the different types of tracks.

Finally, any particles that are reconstructed twice by different algorithms are removed and a container with the best unique tracks is created.

The track reconstruction currently consists of several independent algorithms. The source code of which can be found in the `LHCb/Rec` gitlab repository [1]. The dependencies among different type of tracks are shown in the right part of Fig. 1. Two basic algorithms are `FastVeloTracking` [2] and `PatSeeding` [3] which reconstruct the VELO and T-track candidates, respectively. Those tracks are then used as an input to other algorithms. Upstream track candidates are reconstructed from the VELO tracks and hits in the TT using dedicated `PatVeloTTHybrid` [4] algorithm. We specify two types of long tracks: forward and match. The first, reconstructed using (`PatForward{HLT1|HLT2}` [5]), starts with upstream (HLT1) or VELO (HLT2) tracks and searches for corresponding hits in the T stations. The latter, called the track matching (`PatMatch` [6, 7]), uses both VELO and T tracks as input and matches them in the magnet region. Finally, the downstream algorithm (`PatLongLivedTracking` [8]) uses T-track candidates as seeds and searches for corresponding clusters in the TT. The outputs of all algorithms are merged, eliminating candidates that were found twice. After cutting on the χ^2/ndof of the track approximately 22% of all reconstructed long tracks are still estimated to be fake. To further reduce this rate, a requirement on the ghost probability [9], is used which reduces about 1/3 of fake tracks. In addition to the track finding, the tracking sequence determines primary vertices (PVs). The PV finding algorithm (`PatPV3D` [10, 11]) uses the Kalman Filtered VELO tracks to determine the position of proton-proton interactions in the event.

In Run II (2015-2018) the LHCb experiment introduced real-time reconstruction which unifies the online and offline processing [12]. All of the tracking algorithms have to fit in the trigger timing budget, resulting in a two step procedure. In HLT1, all VELO tracks are reconstructed and fit with a simplified Kalman filter. Then the primary vertex finding is performed. These VELO tracks serve as an input to the Upstream tracking, where an initial momentum estimate is made. Upstream tracks are subject to a transverse momentum cut, which is presently > 400 MeV/c prior to the forward tracking. After the forward tracking this is further tightened to > 500 MeV/c, and a Kalman fit is applied resulting in long tracks. These long tracks are the necessary input to the tracker alignment task. The second stage, HLT2, runs with looser requirements. Remaining VELO tracks, which were not extended into long tracks, are propagated to the T stations, this time without a transverse momentum threshold and without requiring clusters in the TT. At this point, he output of both forward tracking algorithms is merged. A standalone T track reconstruction is performed, followed by the matching and downstream algorithms. All track candidates are fit using a Kalman filter and tracks that are made more than once by the different algorithms are removed. The tracking sequence runs in both the trigger (Moore) and offline (Brunel). The tracking sequence as constructed in `LHCb/Brunel` [13] is as follows:

```
#HLT1
RecoVELOSeq
 FastVeloTracking             #VELO tracks finding
RecoTrHLT1Seq
 TrackHLT1
   TrackHLT1VeloTTPatSeq
    PatVeloTTHybrid           #Upstream tracks finding
   TrackHLT1ForwardPatHLT1S
    PatForwardHLT1            #Long (Forward) tracks finding
   TrackHLT1FitHLT1Seq
    CopyVeloTracks
    VeloOnlyInitAlg
    ForwardHLT1FitterAlg      #Kalman Filter Forward tracks
RecoVertexSeq
 PatPV3D                      #Primary Vertex reconstruction
 PVVeloTracksCleaner

#HLT2
RecoTrHLT2Seq
 TrackHLT2
   TrackHLT2ForwardPatHLT2S
    PatForwardHLT2            #Long (Forward) tracks finding
    MergeForwardHLT1HLT2      #Merge both Forward findings
   TrackHLT2SeedPatSeq
    PatSeeding                #T-tracks finding
   TrackHLT2MatchPatSeq
    PatMatch                  #Long (Match) tracks finding
   TrackHLT2DownstreamPatSe
    PatLongLivedTracking      #Downstream tracks finding
 FitHLT2
   FitHLT2BestSeq
    TrackBestTrackCreator     #Kalman Filter and clone killing
```

2.2 Particle identification

2.2.1 RICH PID

RICH reconstruction involves a number of steps, that can broadly be broadly classified as:

1. Pixel Processing. This sequence deals with the RICH raw data. It is first decoded into unique channel identifiers. Optionally, clustering can then be performed if

required (not done in Run II for HPDs, but might be required with MaPMTs in Run III). Finally the position of these pixels (or clusters) are computed in both global and local RICH coordinate systems.

2. Track Processing. This sequence takes the Track objects as provided by the tracking system, and computes a number of quantities from them. First, the intersections with the RICH1 and RICH2 radiator volumes need to be determined. For a track to be considered for processing in the RICH, it must have an intersection with at least one radiator. This produces 'segments' which describe the trajectory through the radiator volume. The segments are then ray traced (as if it were a photon) through the RICH mirror system to the detector plane, and the hit positions are computed. The photon yields, expected Cherenkov angles and an estimate of the per track Cherenkov resolution are also computed under each of the deuteron, proton, kaon, pion, muon and electron mass hypotheses. An important algorithm in the sequence is the one which forms the Cherenkov mass cones for each segment. This involves taking each segment, and for each mass hypothesis (that is above threshold) ray traces a fixed number (say 100) of Cherenkov photons at the expected Cherenkov angle to the detector plane. This provides critical information on the detector acceptance for each segment. Due to the large number of ray tracings this involved, this was the second most CPU expensive step in the whole sequence.

3. Photon Reconstruction. This sequence takes the pixel and segment information formed in the previous sequences, and forms candidate Cherenkov photons from them. This step is the most CPU intensive due in part to the large number of segment and pixel combinations that need to be considered, and in part due to the computation intensity of the calculations. This is the single most CPU intensive step in the processing.

4. PID. The final sequence takes the information produced in the previous ones, and forms a description of the event based on a set of assumed mass hypotheses for each track, and derived from this the expected signal and background distributions in the detector. This is compared to the observed data to provide an overall event likelihood for the set of mass hypotheses that where assumed. These mass assignments are then changed, in an iterative minimisation of the likelihood to provide the final likelihood information for each track and mass hypothesis. Combined, the algorithms running in this sequence are the third most CPU expensive.

The following output shows how the above description of the steps in the RICH processing map onto the algorithms currently running as part of the RICH sequence. Note that only the sequence for Long tracks is shown. Each of the three track types (Long, Downstream and Upstream) have similar, but separate, sequences for steps 2, 3 and 4 above.

```
RichRecoSeq
  RichPixels                                  Step 1. Pixel processing
    RichPixClustering
    RichPixGlobalPoints
    RichPixLocalPoints
```

```
      RichLongReco                                  Step 2. Track processing
        RichTracksLong
          RichTrackSegmentsLong            cd
          RichTrackGloPointsLong
          RichTrackLocPointsLong
          RichEmittedYieldsLong
          RichEmittedCKAnglesLong
          RichMassConesLong
          RichDetectableYieldsLong
          RichGeomEffLong
          RichTkSegmentSelLong
          RichSignalYieldsLong
          RichSignalCKAnglesLong
          RichCKResolutionsLong
        RichPhotonsLong                          Step 3. Photon Reconstruction
          RichPhotonRecoLong
          RichPredPixelSignalLong
        RichRecSummaryLong
        RichPIDLong                            Step 4. Likelihood Minimisation
          RichGPIDInitLong
          RichPixBackgroundsIt0Long
          RichGPIDLikelihoodIt0Long
          RichPixBackgroundsIt1Long
          RichGPIDLikelihoodIt1Long
          RichGPIDWriteRichPIDsLong
```

Each algorithm in the above list is implemented using the developing Run 3 framework.

2.2.2 Muon PID

The identification of muons in LHCb is mostly based on the Muon detector [14], which is composed in Run II of five detecting stations interleaved by the calorimeters (M1, M2) and filtering iron walls (M2→ M5). The readout of the muon detector is given by the OR of horizontal and vertical *physical pads*, and the crossing of the two defines a *logical pad* whose dimensions give the *x, y* pad size associated to the hit. If there is no simultaneous readout, the *x, y* pad size is given by the whole physical dimensions of the *physical pad*. In the following we will refer to the single hits given by the *physical pads* as *uncrossed hits*, and to the *logical pads* as *crossed hits*.

The muon identification algorithm is a two-step procedure (the main code is `MuonIDAlgLite`). The first step identifies the incoming particle as a muon if the binary variable IsMuon is set to true. The evaluation of IsMuon relies on the number of hits found around the tracks extrapolated through the muon stations. The size of the hit search windows, named FoI (Field of Interest), are parametrised accounting for the particle momentum and the muon detector regions crossed (see [15] for the details). IsMuon is set to true, when the algorithm finds a coincidence of muon stations as a function of the momentum. Similar to IsMuon, other two boolean variables are constructed: IsMuonLoose that requires a fewer amount of hits with respect to IsMuon and IsMuonTight than requires the same amount of hits as IsMuon, but only *crossed hits*. The algorithms used to classify the muon candidates as just described, are in `CommonMuonTool` and run both at the HLT1 that at the offline reconstruction levels.

The second step builds a muon likelihood, DLL, using the average squared distance in units of pad size, between the closest hit in FoI to the track extrapolation points on each station. The DLL variable is used in the global particle identification procedure to be combined with the information from the other PID detectors to evaluate a combined likelihood variable. In each of the bins (samples are binned in momentum and position), two tests are performed that yield $P(\mu)$, the probability of the candidate being a muon, and $P(\text{ not } \mu)$, the probability of the candidate not being a muon. From those quantities the delta log likelihood, DLL, is calculated [15].

It should be noted here, that due to the two-dimensional binning, many calibration constants are needed. D^2 and the DLL are saved in the muon track object. In the offline reconstruction, the quantities $\log(P(\mu))$ and $\log(P(\text{ not } \mu))$ are stored in the muon PID object. Additionally, a track fit is performed on the extrapolation using only the closest hits. The resulting χ^2/ndof is also stored in the muon track object. The DLL is used in the global particle identification procedure to be combined with the information from the other detectors to evaluate the combined DLL variable. Furthermore, $\log(P(\mu))$ and $\log(P(\text{ not } \mu))$ are used as input to a Neural Network based particle identification called `ProbNN`. Details on the combined particle identification can be found in Ref. [16].

For each muon candidate the identification algorithm evaluate another variable, `NShared`, that helps to distinguish between actual and ghost tracks. The `NShared` variable has a discriminating power against background, and can contribute to the reduction of particle misidentification. The `DLL` and the `NShared` variables are evaluated for each muon candidate classified as `IsMuonLoose`, by the tools `DLLMuonTool` and `NShared` respectively. The last step of the muon identification algorithm is to classify as a muon track each incident track that has be found to be at least `IsMuonLoose`: this is done by the tool `MakeMuonTool`.

The `CommonMuonTool` is used since the beginning of Run II both at the HLT2 and offline reconstruction level to calculate the aforementioned variables, and is used at the HLT1 level to calculate `IsMuon`. Additionally, there are two tools called `CommonMuonHitManger` and `DeMuonDetector`, which extract the hit information from the muon raw detector data. The `CommonMuonTool` offers a dedicated method for each logical step in the `IsMuon` algorithm. In addition, functions offering functionality to calculate `IsMuonLoose` and `IsMuonTight` are implemented. What follows is an overview of the methods which are used both in the offline reconstruction by the `MuonIDAlgLite` code and in the HLT1 trigger by the `IsMuonTool` code. More details on the algorithm are given below.

- The `initialize` method sets up the tool. It loads additional tools and fetches the constants from the database.

- `preSelection` takes a track object and returns whether the track passes the pre-selection criteria. In this case it just checks if the track momentum is larger than the cut value ($p > 3$ GeV/c).

- `extrapolateTrack` takes a track object and extrapolates it through the muon stations. It returns a point (x, y) for each station (at a fixed z) except M1 which is not used.

- `inAcceptance` uses the output of `extrapolateTrack` in order to check whether the coordinates of the extrapolated hits are within the acceptance of the muon stations.
- `hitsAndOccupancies` takes both a track and a `MuonTrackExtrapolation` container as input and returns two containers: the first holds the hits that are found in the muon stations within the FoI around the extrapolations, the second container holds the total number of hits in each station, which is called the occupancy of the station. The latter is also used to check whether a station has hits within a FoI.
- `extractCrossed` takes the hits in the muon stations as input and selects only those that are crossed. Additionally, it also calculates the new occupancies considering only the crossed hits.
- `isMuon` uses occupancies and the track momentum to classify it. If it obtains a container of occupancies from crossed hits, it calculates `IsMuonTight` by definition.
- `isMuonLoose` also takes a container of occupancies and a track momentum and calculates `IsMuonLoose`
- `foi` for a given station, region, and momentum returns the edge of the field of interest.

In HLT1, muon lines make use of the `IsMuonTool`, which has been adjusted in order to use the functionalities offered by the `CommonMuonTool`. Like every tool that is used by a trigger line, it exploits a method, `tracksFromTrack`, which takes the current HLT1 reconstructed track (trigger track in the following) as input and writes to an output container if the `IsMuon` criterion is met. It uses the functions `preSelection`, `extrapolateTrack`, `inAcceptance`, `hitsAndOccupancies`, and `isMuon` in sequence. At the HLT2 the algorithm receives a collection of reconstructed tracks as input, and the calculation of `IsMuon` is embedded inside a loop over the tracks, using methods in `MuonIDAlgLite` algorithm (for backwards compatibility `MuonIDAlg` still exists). The algorithm accepts both long and downstream tracks.

Two tools are introduced in order to provide additional information that is not used in HLT1. Those are called `DLLMuonTool` and `MakeMuonTool`. The `DLLMuonTool` is responsible for calculating the delta log likelihood (DLL) of the muon hypothesis. It loads all the parameters for the hypothesis tests in different bins in momentum and region as described in Ref. [15] in order to calculate $P(\mu)$ and $P(\text{not } \mu)$. Two different implementations can be used via a flag: `calcMuonLL_tanhist` and `calcMuonLL_tanhist_landau`. In the first case the probabilities are extracted using the reference histograms for signal and background, without analytical description. In the second case $P(\mu)$ is computed as in the `calcMuonLL_tanhist` implementation, while a Landau description is used for $P(\text{not } \mu)$. The default method is the second one.

Both return the likelihood for the muon hypothesis $P(\mu)$ as well as for the background hypothesis $P(\text{not } \mu)$. It also contains the squared distance of the muon track D^2. The DLL is then calculated. Additionally, the tool allows to calculate the `NShared` variable via the `calcNShared` method. This variable relies on relationships between the tracks in an event. The `MakeMuonTool` is intended to create a muon track once all the necessary information is there, through a function called `makeMuonTrack`. If a corresponding flag is set, the tool also performs a track fit in order to obtain the χ^2 of the track.

The MuonPID event model has been expanded to include five more quantities that are not currently used in the muon ID selection. The first quantities are already defined and available for use:

- `chi2Corr`: a χ^2_{best} accounting for correlations among the hits on different stations induced by multiple scattering. It is produced by the `MuonChi2MatchTool`, called for each `IsMuonLoose` candidate.

- `muonMVA1`: a Boosted Decision Tree or μBDT obtained training space residuals, multiple scattering errors, *crossed/uncrossed* hit information, `NShared`, and - for the first time - also the hit times. It is produced by the `MVATool` called for each `IsMuon` candidate.

The remaining three quantities `muonMVA2`, `muonMVA3`, and `muonMVA4` are left free for future developments.

2.2.3 Calo PID

The calorimeter algorithms can be grouped into two main sequences: reconstruction and PID which are respectively configured by CaloRecoConf and CaloPIDConf classes. The reconstruction sequence takes as input SPD/PS, ECAL and HCAL raw banks and produces calorimeter hypothesis, according to the following steps:

1. Digits preparation for the reconstruction: `CaloDigits`. This is done by processing of the raw data (`SpdFromRaw, PrsFromRaw, EcalZsup, HCALZsup`). For the ECAL and HCAL, the data is zero-suppressed; this was done at an earlier stage for the PS while SPD data is binary. Cell energies are then calculated using stored calibration constants.

2. Reconstruction of ECAL clusters: `ClusterReco`. This uses the cellular automaton algorithm which groups digits around local energy maxima called seeds (`EcalClust`). The defult cut on the transverse energy E_{T} of the cluster if 50 MeV. In a second step, the cell energies are corrected for energy leakage from neighbouring showers (`EcalShare`). The formed clusters are then cropped to 3×3 cell clusters centred around the seeds. This cluster shape is default in Run1 and Run2, however, different shapes were implemented in view of Run3 to reduce the aforementioned leakage effects which will be more pronounced at higher luminosity. Finally, the covariance of the cluster is calculated (`EcalCovar`).

3. Reconstruction of photons: `PhotonReco`. Clusters are classified into charged and neutral based on the extrapolation of tracks to the calorimeters. The track-cluster matching quality is quantified by means of a χ^2_γ which takes into account the uncertainties both on the cluster position measurement and on the track extrapolation (`CaloTrackMatch`). Photons are formed from neutral clusters with $\chi^2_\gamma \geq 4$ and $E_{\mathrm{T}} \geq 200$ MeV, to which SPD/PS digits are added. The photon energy is then calculated from the digit energies using Monte Carlo coefficients to correct for lateral and longitudinal leakage (so-called E-corrections). The three-dimensional shower barycentre is calculated from the energy-weighted 2D cluster barycentre

(corrected for a cell-size-dependent bias, S-corrections) and a Monte Carlo parameterisation of the shower penetration depth with energy (L-corrections). These cuts and calculations are performed by the `SinglePhotonRec` algorithm.

4. Reconstruction of electrons: `ElectronReco`. This proceeds in a similar way as for single photons but with $\chi^2_\gamma < 25$ and specific E-S-L-corrections (`SingleElectronRec`).

5. Reconstruction of merged π^0: `MergedPi0Reco`. This is based on so-called split-clusters which are also produced by `ClusterReco`. The algorithm consists in splitting each of the Cellular Automaton single clusters into two interleaved 3×3 subclusters built around the two main cells of the original cluster. The energy of the common cells is then shared among the two virtual subclusters. Each of the two subclusters is then reconstructed as a single photon hypothesis cluster. In particular, *E,S,L*-corrections are applied to the merged π^0 reconstruction (`MergedPi0Rec`).

The calorimeter PID splits into a charged sequence (electrons and muons PID) and a neutral sequence (photon and merged π^0 PID). The charged sequence calculates for tracks matched to a cluster, the values of $DLLe$ and $DLLmu$ as a product of the DLL from ECAL, HCAL and PS. This is done in the following steps.

1. `InCaloAcceptance`. This sequence checks that the tracks are in the acceptance of the calorimeter so next sequences can be run.

2. `CaloMatch`. Calculates the χ^2 value of the track-cluster matching. This is done using the extrapolation of the track from its direction after the magnet (`ElectronMatch`) or before the magnet (`BremMatch`). The clusters matched by the second method are brem photon candidates.

3. `CaloEnergy`. Gets the energy along the track line as measured in the SPD, PS, ECAL and HCAL (`EcalE, HcalE, SpdE, PrsE`). This information will be also used for PID. In particular, PS and HCAL energy will improve the separation of electrons from other charged particles (h, μ) which leave a small energy in this detector.

4. `CaloChi2`. This is where the basic ECAL estimator is constructed as a χ^2_{e} of a global matching procedure. The later includes the balance between track momentum and cluster energy, and between track extrapolation and cluster position. It is run for electron candidates (`EcalChi22ID`), brem candidates (`BremChi22ID`) and clusters (`CluChi22ID`). It makes use of the quantities calculated in previous sequences.

5. `CaloDLLe`. Calculates the $DLLe$ values for each sub-detector. This uses quantities calculated in the previous sequences and template histograms stored in a database: $DLLe^{\mathrm{ECAL}}$ uses χ^2_{e} (`EcalPIDe`), $DLLe^{\mathrm{brem}}$ uses χ^2_{brem} (`BremPIDe`), $DLLe^{\mathrm{HCAL}}$ uses E_{HCAL} (`HcalPIDe`) and $DLLe^{\mathrm{PS}}$ uses E_{PS} (`PrsPIDe`).

6. `CaloDLLmu`. Similar to the previous sequence, using as input only the energy in the ECAL (`EcalPIDmu`) and the energy in the HCAL (`HcalPIDmu`).

The neutral PID sequence assigns a confidence level (DLL) to the neutral calorimeter hypothesis using the following inputs: χ^2_γ, E_{seed} and E_{PS}. Reference histograms of these variables are available for signal and background in each ECAL section (inner, middle, outer) and for converted and non-converted photons (*i.e.* with and without hits in the SPD). The sequence is ran for the three neutral hypothesis: `PhotonID`, `MergedID` and `PhotonFromMergedID` for single photons, merged π^0 and split-photons.

2.2.4 Global PID

The PID information obtained separately from the muon, RICH, and calorimeter systems is combined to provide a single set of more powerful variables. Two different approaches are used. In the first method the likelihood information produced by each sub-system is simply added linearly, to form a set of combined likelihoods, $\Delta log\mathcal{L}_{comb}(X - \pi)$, where X represents either the electron, muon, kaon, proton or deuteron mass hypothesis. These variables give a measure of how likely the mass hypothesis under consideration is, for any given track, relative to the pion hypothesis. Along Run I a second approach has been subsequently developed to improve upon the simple log likelihood variables both by taking into account correlations between the detector systems and also by including additional information. This is carried out using multivariate techniques, combining PID information from each sub-system into a single probability value for each particle hypothesis. These variables are known as `ProbNNx`, with `x` standing for electron, muon, pion, kaon, proton, or ghost. Notice that since the beginning of Run II, the `ProbNN` variables were available also in the trigger.

Detailed information about the `ProbNN` approach is available only in presentations given at meetings and in the code itself. Best information can be found here [1] for Run I and here [2] for Run II. For what concerns code, the main repository is the `ChargedProtoANNPID` [3]. The list of variables used as input for the ProbNN can be found here `ChargedProtoANNPID`/data for each tuning and specie. This list is used at runtime to list what variables are used (looking at any specific file: the first 5 lines are other settings, the inputs start on line 6; lines with a # at the beginning are commented out, so not used). Technically, to know how the data are extracted, the names in these files can be matched by looking at the mapping between the name and a helper class `ChargedProtoANNPID/src/ChargedProtoANNPIDCommonBase.icpp` and then here `ChargedProtoANNPID/src/ChargedProtoANNPIDCommonBase.h` to see exactly what each helper does.

The training of ProbNN variables is done using inclusive B Monte Carlo events. Actual performance depends on the tuning *i.e.* the blending of MC samples used. A large collection of information on the various tunes can be found by looking in this folder [4]. Several different tunings are available for both Run I and Run II. For the Run II samples, only the `MC15TuneV1` and `MC12TuneV4` `ProbNN` PID variables should be used. For all Run II analyses, it is recommended to use the `MC15TuneV1` `ProbNN` variables - those of `MC12TuneV4` are optional. For the Run I samples, only `MC12TuneV2` and `MC12TuneV3` are

[1]https://indico.cern.ch/event/226062/contributions/475644/attachments/371741/517276/ANNPIDRetuning-Reco14-06052013.pdf

[2]https://indico.cern.ch/event/508832/contributions/2030857/attachments/1249785/1842643/ANNPID-2015TuneV1-30032016.pdf

[3]https://svnweb.cern.ch/trac/lhcb/browser/Rec/trunk/Rec/ChargedProtoANNPID/

[4]http://www.hep.phy.cam.ac.uk/ jonesc/lhcb/PID/ANN/

accessible. V3 is not an exact “upgrade” of V2, but it adds more kinematic regions and removed ghosts from the training samples. For many decays it will be better, but there might be some specific cases where it is not. Different versions of `ProbNN` variables are accessible e.g. through `TupleToolANNPID` for ntuples. (Default one in other code, like LoKi, is `MC12TuneV2` for Run I and `MC15TuneV1` for Run II.)

3 Functional framework

To better exploit multi- and many-core architectures, the *functional framework* was introduced. Its aim is to give developers general building blocks that are well defined, multithreading friendly and handle the dataflow between algorithms.

Every algorithm has to define its inputs and outputs. That means that at initialisation time of the application a static data dependency graph can be generated to, first, prevent configuration errors due to wrongly defined locations and, second, to schedule algorithms in the right order according to the data dependencies. Multiple or no inputs and outputs are possible. To guarantee thread safety, inputs to an algorithm are declared as constant and the main execution is not allowed to change the state of an algorithm[5].

The RICH reconstruction was the first big part of the reconstruction to fully embrace the functional framework and modern coding ideas. It was completely redesigned, removing a number of design choices incompatible with the functional framework, and put into production in 2017, see Section 2.2.1 for a description.

4 Requirements: Event model

The event model is not described in detail here, for further information see Ref [17]. Some details have been mentioned in the previous sections. In summary, the event model comprises two aspects, transient data and persistent data. The event model should allow to pass transient objects between algorithms with little overhead and allow to persist the necessary information to perform analyses. Little overhead often means to create smaller objects with only minimal information for the next step. Having all the necessary information available gravitates towards bigger objects. Additionally, the event model has to take into account the need of data structures that better match the requirements placed by modern hardware to exploit parallelism (SIMD vectorisation).

As an example, in HLT1 where the data have to be processed at 30 MHz any overhead from memory allocations quickly contributes a significant fraction to the runtime, while in HLT2 the individual reconstruction algorithms are considerably slower and overhead from the event model might be negligible. Ref. [17] lays out several approaches to reduce the overhead from memory allocations. One example is removing the extensive use of `KeyedContainers`. Very simplified, a `KeyedContainer` assigns each member a unique key which can be different to the index in the container. The `KeyedContainer` is implemented as a map. Adding objects to a map is considerably slower than adding objects to a vector. Replacing `KeyedContainers` in between algorithms seems obvious in many cases, but

[5]Nevertheless, the use of tools inside an algorithm and having *mutable* class members can introduce data races.

further simplifications are likely needed. However, the current persistency heavily relies on the use of keyed objects, *e.g.* see Section 6.2.2, and needs to be adapted.

5 Reconstruction overview: Run III

5.1 Tracking

The upgrade tracking sequence is designed to take an advantage of the successful Run II strategy. Two separated stages are constructed: `fast` and `best`. The first fast stage `RecoTrFastSeq` provides the necessary input to the run-by-run calibration and alignment, while the second best stage `RecoTrBestSeq` performs the remaining part of tracking reconstruction. The source code of algorithms and configuration of sequence can be found in Rec [1]. Since the tracking system is fully replaced by the new detectors, it requires a new reconstruction sequence with new dedicated algorithms:

```
RecoTrFastSeq
  PrPixelTrackingFast          #VELO tracks finding
  PatPV3D                      #Primary Vertex reconstruction
  PrVeloUTFast                 #Upstream tracks finding
  PrForwardTrackingFast        #Long (Forward) tracks finding
  ForwardFitterAlgFast         #Kalman Filter Forward tracks
RecoTrBestSeq
  PrForwardTrackingBest        #Long (Forward) tracks finding
  PrHybridSeedingBest          #T-tracks finding
  PrMatchNNBest                #Long (Match) tracks finding
  PrLongLivedTrackingBest      #Downstream tracks finding
  BestTrackCreatorSeq
    TrackBestTrackCreator      #Kalman Filter and clone killing
  TrackAddExtraInfoSeq
    TrackAddNNGhostId          #Ghost Probability
```

The logic of the reconstruction is similar to already used for Run II. The reconstruction begins with the VELO tracking (`PrPixelTrackingFast`), where an internal simplified Kalman Filter is used. Those tracks are used to either reconstruct primary vertices (`PatPV`) or serve as seeds for the upstream tracking (`PrVeloUTFast`). In the recent implementation, at this stage the transverse momentum requirement is set to be greater than 300 MeV/c. The upstream candidates are then extended to SciFi detector, and the forward tracking is performed (`PrForwardTrackingFast`) with the transverse momentum threshold increased to 400 MeV/c. The last part of the fast `RecoTrFastSeq` stage is a Kalman Filter based track fit of all Forward candidates. The best stage mimics the HLT2 Run II sequence. First, the Forward tracks are found based on the VELO input (`PrForwardTrackingBest`). In contrast to the Run II sequence, already extended VELO tracks are reconsidered, since the flexibility written for Run II is not yet ported. A standalone T-track seeding is then performed (`PrHybridSeedingBest` [18]), which together with the VELO tracks provide the input the matching algorithm (`PrMatchNNBest` [19]). Finally, the downstream tracks are created using `PrLongLivedTrackingBest`. All dependencies among tracks are kept as shown in Fig. 1. The sequence finishes with the Kalman Filtering, where all duplicated tracks are removed.

In Run II, the optimised reconstruction algorithms allowed to loosen the p_T threshold in the forward tracking from 1.2 GeV/c to 500 MeV/c in HLT1, significantly improving the reconstruction efficiency of low momentum particles. The recent transverse momentum thresholds for the upgrade, 300 MeV/c for (`PrVeloUTFast`), and 400 MeV/c (`PrForwardTrackingFast`) have been chosen due to improvements in the forward tracking. These thresholds will however need to be optimised based on the timing and performance. For the timing studies, two alternative transverse momentum threshold settings are considered: a) intermediate cut: $p_T > 600$ MeV/c (`PrVeloUTFast`), $p_T > 800$ MeV/c (`PrForwardTrackingFast`) b) hard cut: $p_T > 1.2$ GeV/c (`PrVeloUTFast`), $p_T > 1.4$ GeV/c (`PrForwardTrackingFast`). Nevertheless, those thresholds are not yet fixed and need to be carefully optimised based on both, timing and performance.

5.1.1 Developments since LHCb-PUB-2017-005 [20]

The performance reported in Ref. [20] has been obtained using the Run II framework. Since then, a new framework has been developed which allows the efficient usage of multithreading and parallelism paradigms. The two tracking stages, fast and best, have been successfully ported to the new, functional framework.

In addition to this several changes has been made in the algorithms themselves. The matching algorithm `PrMatchNNBest` has been reoptimised, where the main attention has been paid to the Neural Net optimisation with a new set of variables. The new tuning results with the speed up about 20% and reduction of the fake tracks from 26% to 20% with minimal efficiency loss at the level of 0.5%.

In 2017 a new tuning for the downstream tracking has been performed where two different multivariate techniques were employed. Firstly, the algorithm filters *T tracks* using a bonsai Boosted Decision Tree with 11 dimensional discretised space resulting in a rejection of fake seed tracks. Then the remaining *T-track* candidates are matched with TT hits. Finally, the good track candidates are selected based on a neural network decision. Overall signal efficiency improves by about $O(3-5\%)$, together with $O(3-5\%)$ improvement in fake track reduction. The improvements from `PatLongLivedTracking`) can be ported to the upgrade tracking downstream algorithm `PrLongLivedTrackingBest`.

In addition, several changes have been made to the individual algorithms:

- `StoreVPClusters` and the nominal Velo Pixel tracking algorithm have been merged into one algorithm, which was then been modified to be thread safe. This resulted in a slight increase in the execution time. Following this, the memory usage of the this algorithm, `PrPixelTrackingFast`, has been optimised by removing `KeyedObject` and `KeyedContainer` for `VPClusters`. The structure used to store hits has been changed to Structures of Arrays (SOA). These modifications result in 75% less memory allocation and a timing reduction at the level of about 30% (with respect to merged version).

- The matching algorithm `PrMatchNNBest` has been re-optimised, where the majority of the attention has been paid to the Neural Net classifier optimisation with a new set of variables. The new tuning gives a speed up at the level of about 20% and reduction of the fake tracks from 26% to 20% with minimal efficiency loss at the level of 0.5%. Further improvements are expected, including optimisation of track fit model and further timing reduction.

- In 2017 a new tuning for the downstream tracking has been developed where two different multivariate techniques were employed. T-track candidates are selected using a bonsai Boosted Decision Tree (add Ref) which results in a significant rejection of fake seeds. The remaining T-track candidates are matched with TT hits and the good track candidates are selected based on a neural network decision. Overall signal efficiency improved by about $O(3-5\%)$, together with $O(3-5\%)$ improvement in fake track reduction. This development shows the potential direction of the future improvements for this particular type of tracking.

5.1.2 Ongoing developments

Preliminary throughput studies indicate that at least a factor of 6 speedup is still required to implement a sequence similar to that used in Run 2, however significant reductions to this factor can be made at some cost to the physics [21]. A critical part is the VELO tracking, which currently takes about 30% of the timing budget, and is mandatory to perform any physics measurement. Despite the merge of `StoreVPClusters` and the nominal VELO tracking algorithm, several additional improvements are under investigation:

- implementation of the `VPFilter`,
  which distinguishes tracks pointing from the primary and secondary vertices. It is another implementation of the IPχ^2 requirement commonly used in the LHCb experiment and used for finding a well displaced tracks from the primary interactions. The `VPFilter` could serve as an alternative filter for selecting hits corresponding to only secondary particles in the detector, therefore reducing the time spent in the nominal VELO tracking.

- removing backwards tracks.
  The backwards tracks are needed for unbiased primary vertex reconstruction, however, physics analyses need only tracks associated to the particles passing through the detector in the forward direction. Removing the backward tracks reduces by O(48%) the CPU requirements of the VELO tracking. The current studies show that this results in 20% poorer primary vertex resolution and 4% lower primary vertex efficiency.

- splitting and early breaking a pair creation for forward/backward tracks in the VELO tracking.

- using cellular automata in the VELO tracking.

Another part of the fast stage is the primary vertex reconstruction. The work focuses on speeding up the code without significant efficiency and resolution degradation. The improvements consider changes in the seeding procedure, where the default three dimensional approach `PVSeed3DTool` has been changed to simplified two dimensional version `PVSeedTool`, with reoptimised settings. In addition to this, the default fitter `LSAdaptPV3DFitter` has been replaced by `AdaptivePV3DFitter`, with the corrected computation of primary vertex χ^2 as well as code speed up. The overall time improvement order of O(65-70)% is found, with negligible impact to the physics performance.

The main time consumer of both the fast and best reconstruction stages is a Kalman Filter, a linear quadratic estimator, which produces the final fitted tracks and their associated covariance matrices. To achieve this goal several physics aspects have to be considered such as multiple scattering or energy loss. There are many ongoing and independent activities which have a potential of the significant time reduction without performance loss. We briefly describe them:

- Cross Kalman [22],
  which allows tracks to be processed in parallel. The goal is to perform calculations using SIMD instructions while avoiding empty vector units. From a technical point of view it is another implementation of `ITrackFitter`, developed as the `TrackVectorFitter`.

- Parametrised Kalman,
  the Kalman Filtering requires many track extrapolations from one detector layer to the next, where the material and magnet field maps are needed. Possible time reductions can be made by replacing the maps by parametrised layer to layer predictions, which take into account the magnetic field intensity. Several different parameterisations are used inside (VELO, UT, SciFi) and between (VELO-UT, UT-SciFi) subdetectors. Preliminary studies show a factor of five speedup. This algorithm has a potential to be used in the fast stage for fitting the Forward tracks.

- Simplified geometry [23],
  the Kalman Filtering relies on the geometry used for the detector description, where the number of volumes reduces from $O(10^6)$ to about 20 with the effective material. Preliminary tests already indicate the speed by the factor $\sim$10-13 with respect to the full geometry. The Simplified geometry has been used in Ref. [20], however work is still ongoing and requires validation.

5.1.3 Future developments

The preliminary throughput studies show that the tight selection requirements are not enough for running the recent tracking sequence at the 30 MHz bunch crossing rate. All tracking algorithms need to be further revisited looking for the time improvements. The main effort is focused on the speeding up the fast stage without significant performance lost. As a well defined bottleneck, the possible improvements and/or compromises in the VELO tracking are under extensive investigation. In addition, other part of the fast stage are widely studies paying attention to the primary vertex finding and forward reconstruction.

The effective memory usage requires a consistent use of SOA/AOS paradigms. The data structures are optimally chosen for the specific algorithmic problem. Currently, the data flow among tracking algorithms uses the AOS, the preliminary studies indicate no visible profits from using the SOA. Therefore, changes in the data structure might require reimplementation of particular algorithms. It is an important open topic for the future discussions.

Another crucial topic is the clusters decoding in the Event Filter Farm. It has to be understood whether the new detector's Readout Boards could perform the preliminary or partial clusters sorting, which would result in the faster decoding.

5.2 Particle identification

5.2.1 RICH PID

The RICH reconstruction, as described in Section 2.2.1, has already been ported to the upgrade framework, and was very successfully used during 2017 data taking. As such the processing sequence already fully utilises the Function Framework, and the algorithms have been modified to make them re-entrant and thus thread safe.

Another aim during the modernisation process was to update the internal data structures used in the RICH, to pass information between the various sub-algorithms, to be more suitable for modern computing standards and in a format that promotes the use of techniques such as SIMD vectorisation. Utilisation of SIMD instructions is a critical aspect of the upgrade, as utilising these (increasingly) powerful instruction sets is the only way to fully exploit the full compute power of modern hardware. The developments in place for the 2017 Run II processing are only the first steps in this direction. Some explicit use of SIMD instructions were used, in the photon ray tracing, that lead to modest CPU improvements. Further work for Run III will focus on both extending this to the full reconstruction sequence, but also to more thoroughly using the SIMD instructions, and thus fully realising the expected gains from this area.

Finally, one aspect of the RICH reconstruction that has not been addressed is the final (persistent) event model, that saves the PID information for each track. This event model, will need to be heavily adapted for the upgrade. This work will follow in close harmony with the associated modernisation of the track objects.

5.2.2 Muon PID

The increase of the incident rate on the muon system will be tolerable up to the upgrade luminosity of $2{\times}10^{33}\text{cm}^{-2}\text{s}^{-1}$, in all stations apart from M1, which will be removed during LS2. Also, because of the particle flux expected on the innermost regions of M2 will be very high, an additional shielding will be installed around the beam-pipe behind the HCAL to reduce the occupancy in these regions. These will be the main hardware interventions foreseen at the upgrade, together with the installation of a new off-detector readout electronics compliant with full 40 MHz readout.

In this high luminosity scenario the muon identification algorithm must guarantee a high muon identification efficiency, while keeping the misidentification probability as low as possible. Due to the high hit occupancy expected at the upgrade running conditions, the `IsMuon` criterion plus a soft DLL cut produces an unacceptable increase of the misidentification probability of about a factor of two [24]. The two new variables `chi2Corr` and `muonMVA1` already available for the last year of Run II data taking allow to preserve the present identification versus misidentification performance.

Another muon identification algorithm has been developed in the context of the $K_S \to \mu^+\mu^-$ physics analysis [25] to both improve the background rejection and increase the identification efficiency mainly for muons of low momenta. The tools developed for this algorithm are under study to be included in one or more new muon identification algorithms, `nIsMuon`, that could replace the actual `IsMuon` in Run III.

5.2.3 Calo PID

SPD/PS removal In Run III, the scintillating pad detector (SPD) and the preshower (PS) of the current detector will be removed. The principal purpose of these components in the current experiment is in the L0 trigger. The removal of the SPD/PS simplifies the calorimeter system, with benefits for energy calibration and project costs. Nevertheless, there are some consequences for the offline performance such as photon and electron PID because the current estimators are using SPD/PS information. For instance, studies reported in the LHCb PID Upgrade TDR show an absolute reduction of 10–15 % in photon efficiency while electron PID is almost unchanged at high-p_{T} (above 10 GeV/c).

Higher luminosity The higher instantaneous luminosity in Run III and resulting increased pile-up will impact on the energy resolution of the ECAL due to overlap of neighbouring showers. To mitigate this effect, the current size of the clusters can be reduced and two new shapes were investigated: 2×2 square and swiss-cross shapes. The energy reconstruction with these new shapes mitigate to a large degree the effect of the pile-up with respect to the present reconstruction, without significantly degrading the energy resolution.

Portability of current sequences Reconstruction and PID sequences can already be configured to ignore the SPD/PS information and use the alternative cluster shapes. The CaloRecoConf and CaloPIDConf classes both use a boolean (`NoSpdPrs`) to choose between current/upgrade geometry. The new shapes were implemented in the cluster reconstruction through the options `ClusterEnergyMasks` and `ClusterPositionMasks` in CaloRecoConf and will be set by default from the database. All PID algorithms, however, needs to be adapted to these new shapes.

5.2.4 Global PID

For Run III a global PID sequence has not yet been defined. Different choices are possible depending on the event model used. As far as the PID reconstruction sequences populate the objects in the actual event model, the same set of combined likelihoods $\Delta log\mathcal{L}_{comb}(X-\pi)$ as Run I and Run II is available. An improved version of the $\Delta log\mathcal{L}_{comb}(X-\pi)$ variables can be obtained using the improved performance of each single PID sub-detector as soon as they are ready. The performance of such variables should inherit all those inbuilt in the PID information of the muon, RICH, and calorimeter systems separately for Run III. Another technically available choice is the use of the present definition of the `ProbNN` variables which may make sense or not depending on the existence of the various input variables. Finally the longer term solution of implementing new approaches equivalent to the `ProbNN` variables, profiting of correlations among variables and of information from non-PID systems, needs the definition of the new the event model.

6 Trigger

At its most general, the aim of a trigger is to reduce the amount of data recorded by a detector to only that of interest for physics analysis. This can be achieved in two ways: *reduction of rate* by separating events that contain interesting signals from those that do not, and *reduction of size* by selecting the subset of the event that is useful for further analysis. The present Run II trigger does both of these things, and the Run III trigger is expected to do the same.

Reduction of rate is achieved through *selections*, where analysts choose events based on a very broad range of discriminating criteria such as the invariant mass of a combination of particles, the particle ID of daughter particles, vertex quality, track quality, *etc.* Two types of selection are possible: *inclusive* selections save more than one decay type based on general criteria, *e.g.*: 'the decay of a B meson to any number of tracks' and *exclusive* selections where the full decay is completely specified, for example 'the decay of a B meson to two kaons and two muons'. In Run II the majority of the B signal rate is selected inclusively using the topological triggers, while the majority of the D signal rate is selected exclusively. In Run III both types of selection should be catered for.

Selections are defined by the analysts and as such we should endeavour to make building these selections straightforward. In the present Run II trigger this is achieved using LoKi functors.

Reduction of size is achieved through the *Turbo* paradigm, where analysts choose how much of the event information they wish to persist. Potential objects to be persisted are: the raw or derived event information from one or more subdetectors, the reconstructed objects explicitly requested for an exclusive decay, partial selections of reconstructed objects based on some criteria, *e.g.* 'save all of the tracks in a cone surrounding this decay', *etc.*

This section describes the principal layout of the HLT1 and HLT2 sequences, meaning the algorithms which bind together the different reconstruction stages and the algorithms which are used to make a trigger decision and persist the trigger decision for analysis usage.

Very few of these algorithms have been ported to the functional framework and no work has been done on a potentially new event model from the Hlt side. In the following sections we will layout the current design and later discuss the short comings of the current design. This chapter should act as a guideline to prioritise and focus the software development in the coming year.

6.1 Selection framework

The main task of the software trigger is to select signal events and candidates based on the reconstructed objects. The online reconstruction is based on the algorithms described in Section 2.

A requirement of HLT1 and HLT2 lines is to ensure that algorithms which produce input to the decision making are run in the right order and are executed only once. The layout of the decision sequences is different between HLT1 and HLT2. HLT1 selections are based on partial event information, *e.g.* one or two track combinations only. HLT2 performs a full event reconstruction and full decay chains with many objects can be reconstructed. Both are described in the following.

6.1.1 HLT1 selections

HLT1 lines are based on the streamer framework[6]. The streamer framework allows reconstruction steps to be interleaved with selection cuts to only perform the necessary operations. Every step of a trigger line is defined by a LoKi-functor which is passed if at least one candidate passes the selection or reconstruction step.

As an example the code of the `Hlt1TrackMuon streamer` is given:

```
TrackCandidates
>>  ((TrPT > %(PreFitPT)s * MeV) & (TrP  > %(PreFitP)s * MeV))
>>  FitTrack
>>  ((TrPT > %(PT)s * MeV ) & (TrP  > %(P)s * MeV ))
>>  ((TrCHI2PDOF < %(TrChi2)s) & (Tr_HLTMIPCHI2('PV3D') > %(IPChi2)s ))
>>  IsMuon
>>  SINK('Hlt1%(name)sDecision')
>>  ~TC_EMPTY
```

The input to the line are long tracks coming from the `TrackCandidates` functor. To reduce the number of tracks to fit, a preselection on the momentum is performed. Track quality and impact parameter requirements are placed after the track fit. Finally, the trigger candidate has to be identified as a muon. The `SINK` functor saves the decision of the line.

The streamer framework itself does not guarantee that the reconstruction algorithms which provide the input to selections are run. If a functor needs external input, this has to be specified separately, *e.g.* the `Tr_HLTMIPCHI2('PV3D')` functor requires that the primary vertex reconstruction is executed.

Therefore the the layout of an `Hlt1Line` looks like this:

```
Hlt1Line('TrackMuon',
         prescale  = 0..1,
         postscale = 0..1,
         priority  = 0..,
         L0DU = 'L0Muon|L0DiMuon',
         algos = [ Hlt1GECUnit('Loose'),
                   PV3D('Hlt1'),
                   trackingAlgos,
                   streamer  ]
)
```

Additional selections which can be set are prescales, postscales, ODIN, L0 requirements and global event cuts. The latter is defined in the algorithm sequence.

6.1.2 HLT2 selections

The information available to HLT2 lines changed drastically between Run I and Run II. While in Run I particle identification could only be run for selected lines after a further reduction of events, in Run II basically the full offline reconstruction is run in HLT2. This means that in Run II every line specifies the full offline reconstruction as input while in

[6]A more detailed description of the HLT1 streamer framework is given in Reference [26].

Run I different lines could have different levels of reconstruction, *e.g.* with or without particle identification to fulfil the timing requirements.

Most HLT2 selections combine basic particle candidates to composite particle candidates, referred to as *combinatorics* in the next section.[7] The input to most HLT2 lines are basic `Particle` s. A basic `Particle` is built out of a `LHCb::ProtoParticle`, in form of a pointer, which holds the full reconstructed information and an assigned particle hypotheses, *e.g.* the kaon, pion, electron, muon or proton hypotheses. Different `Particle`s can be built out of the same `ProtoParticle`. The `ProtoParticle` holds pointers to different reconstructed objects like in the case of a charged particle a track and the associated particle identification objects.

As the HLT1 lines, HLT2 lines have to declare manually their inputs to guarantee that the reconstruction sequence provides all necessary information; *e.g.* a common pitfall in HLT2 lines is that impact parameter cuts which require the existence of the primary vertex reconstruction do not declare the primary vertex reconstruction as an input, eventually leading to a loss of these events or depending on the luck that another line provided the PV reconstruction. Another pitfall mainly in Run I was that different paths of reconstruction existed. So a line which used PID cuts had to specify different dependencies than a line which did not use PID cuts.

6.1.3 Combinatorics

Finding "good" multibody combinations of particles is generically referred to as "combinatorics". In the current trigger then "good" is rather analysis-dependent, but typical criteria are that the charged particle tracks form a vertex with small χ^2, and that this vertex is displaced from the PVs.

The relevant algorithms in the old framework are `CombineParticles` and `DaVinci::N{3,4,5,6,7,8}BodyDecays`. These take several containers of `LHCb::Particle` objects (*e.g.* kaons, muons, ...) as input, and produce a new container of `LHCb::Particle` as output, applying cuts at several stages internally. The vertex fit is performed by a tool, typically `LoKi::VertexFitter`. There are three types of selection applied by these algorithms:

- `DaughtersCuts` that remove particles in the input containers from consideration,
- `CombinationCut` that act on n-body tuples of particles (before the vertex fit), and
- `MotherCut` that act on n-body composite particles (after the vertex fit).

The `DaVinci::N{3,4,5,6,7,8}BodyDecays` algorithms add extra versions of the `CombinationCut` called `CombinationCut12`, `CombinationCut123` *etc.* that are applied to 2-body, 3-body *etc.* tuples of particles and allow bad combinations to be rejected more efficiently. Typical examples of cuts that are applied at each stage are:

- `DaughtersCuts` displacement from PVs (χ^2_{IP}– `BPVIPCHI2`), PID information (`PIDK`), p_{T} thresholds (`PT`)
- `CombinationCut{,12,123,...}` pairwise distance between tracks (χ^2_{DOCA}– `ADOCACHI2`), vector and/or scalar sum of child transverse momenta (`APT`, `ASUMPT`), pre-vertex-fit parent mass (`AM`)

[7]Examples of HLT2 lines can be found in `gitlab Hlt/Hlt2Lines`.

- `MotherCut` vertex fit quality (χ^2_{vtx}– `CHI2VX`), post-vertex-fit parent mass (`M`)

Note that the first of these, the `DaughtersCuts`, could be implemented by placing appropriate filtering algorithms in front of `CombineParticles`, and the `MotherCut` could be implemented by attaching a filter to the output, but the `CombinationCut` is tightly integrated into the algorithm.

In Run II there is no protection against running the vertex fit multiple times on the same set of n input particles in different selections. Additionally, because the inputs of these algorithms are `LHCb::Particle` not `LHCb::Track`, the same set of particles may be fitted several times under different mass hypotheses (pion, kaon, *etc.*), even though the assigned mass has no, or negligible, impact on the fit. This is mitigated by the use of PID information in `DaughtersCuts` in the Run II trigger, but it may not be possible to run the PID reconstruction before every trigger line.

6.1.4 Looking forward

Running HLT1 at 30 MHz will be challenging. Assuming the current budget of 1000 farm nodes, this means that every farm node has to process 30k events per second. An HLT1 framework which adapts the functional framework and runs together with a multi-threaded scheduler needs to be developed. The concepts of the HLT1 streamer model map well onto the functional framework, albeit the implementation will need a lot of changes, *e.g.* classes like `HltBaseAlg` and `HltSelection` can be seen as a direct ancestor of `Gaudi::Functional`. To profit from newer architectures it should act on containers and not single objects, *e.g.* the streamer framework heavily relies on the `tracksFromTrack(const LHCb& Track track, std::vector<LHCb::Track> output)` interface. This interface inherently makes horizontal vectorisation more complicated[8].

For the upgrade we have to maintain the possibility that different HLT2 lines can specify different reconstructions. It will likely be the case that the same reconstruction will be the basis for many trigger lines. The functional framework which focuses on properly defining inputs and outputs removes the boilerplate in the python configuration. The input and output matching is moved to the scheduler of Gaudi and will guarantee the existence of the right inputs and outputs. This will help to simplify the writing of HLT2 and potentially HLT1 lines and will make them more closely resemble what is being done in *Stripping* and user-job selections.

It is an open question whether or not in Run III it will be more computationally efficient to keep the Run II model, or to replace it with an "up front" combinatorics engine that finds all 2-track, 3-track, ... combinations that form a good vertex and provides these as inputs to the trigger selections. It is clear that with the higher luminosities and higher track multiplicities of Run III the combinatoric timing will be more difficult to control than in Run II. Additionally, in Run III it might be more efficient to seed HLT2 selections from HLT1 candidates, rather than starting from scratch and matching afterwards.

Another idea for multi-body combinations is the following: if multi-body particle combinations are formed up-front without reference to specific mass hypotheses, more efficient clustering algorithms can be used than in the current trigger. For a simple illustration of this, assume that the selection requirements are expressed as a "seed" requirement that one particle in the combination must satisfy, and a second requirement

[8]Nevertheless, benchmarking the throughput should drive the design process

that every pair of particles must pass. In this case, the search for the 3rd and subsequent particles in the combination must only cover a small subset of particles in the event that are known to satisfy the pairwise requirement.

6.2 Persistency

The meaning of persistency is two fold in the Run II trigger, the first is the persistency of trigger decisions and trigger objects used in offline reconstructed data, the second is the persistency of trigger objects in the Turbo stream where no further offline reconstruction is performed.

6.2.1 Persistency of trigger decisions

It is important to know which trigger lines selected an event and which trigger objects lead to a positive trigger decision in offline analysis. Both types of information are persisted at the end of every event. The trigger decisions are saved in the `HltDecReports` and the `HltSelReports`.

For every trigger line present a `HltDecReport` is persisted. A `HltDecReport` are two 16-bits mask where the first is the identifier and the second contains amongst other the information if the given trigger line selected the event, how many trigger candidates the line created and also basic information in which selection step the trigger line failed. The `HltDecReport` are written by the `HltDecReportsWriter` and can be decoded from the raw event by the `HltDecReportsDecoder`.

A `HltSelReport` contains the necessary information to match a particle candidate used in the trigger to a particle candidate created in the offline processing. The particle candidate can be a composite object or basic particle, like e.g. a pion or a muon candidate. The matching itself is based on `LHCbIDs` of the primitive reconstructed objects, like tracks in the tracking system, calorimeter cluster or muon tracks. For that the `HltSelReportsMaker` and the `HltReportConvertTool` extract the `LHCbIDs` from the basic objects and save it in a structure which corresponds to the structure of the trigger candidate, i.e. the `HltSelReport` allows to decode later of which particle candidates a composite object was made of. In addition to the `LHCbIDs` some basic information like the momentum or the fit quality of a reconstructed track are persisted. The reports are then written to the raw event by the `HltSelReportsWriter`.

6.2.2 Turbo persistency

Events in which at least one Turbo line fired are sent to the `TURBO` stream, where most of the raw information is discarded. In 2015 and 2016, candidates firing Turbo lines were persisted in the previously described `HltSelReports`, which were extended to accommodate the additional information required to ‘resurrect’ the full `Particle` objects, from the reports, for use offline. The conversion from information in the `HltSelReports` to analysis objects, like `Track`, `CaloCluster`, and `Particle`, was performed by the TESLA application.

In 2016, the `PersistReco` flag was made available on a per-line basis. For events with firing Turbo lines that have the `PersistReco` flag enabled, the entire HLT2 reconstruction is persisted, along with the trigger candidate as before. Rather than attempting to store the entire reconstruction in the `HltSelReports`, the packed reconstruction is serialised in to the as-of-then unused `DstData` raw bank. This raw bank is propagated through Tesla, and is de-serialised and unpacked in user analysis jobs. The packing is performed by the standard packing algorithms (*e.g.* `PackTrack` and `PackParticlesAndVertices`), and the (de-)serialisation is done by the `HltPackedDataWriter` and `HltPackedDataDecoder` algorithms.

The introduction of `PersistReco` meant the existence of two persistence strategies. In an attempt to unify these, the persistence of Turbo candidates was migrated to the `PersistReco` model in 2017: candidates that fire any Turbo HLT2 line are packed and persisted in the `DstData` raw bank. This has two advantages:

1. The `HltSelReports` are no longer extended beyond their original purpose, and the complex (de-)serialisation code can be removed. It is then no longer possible for Turbo candidates to interfere with the TIS/TOS mechanism, which uses the `HltSelReports` (such interference caused a bug which required a re-stripping of 2015 and 2016 data), and the number of Turbo-specific algorithms is reduced.

2. Turbo candidates are treated in the same way as the rest of the HLT2 reconstruction. This allows for fine-grained per-line control as to what parts of the reconstruction are saved (*e.g.* only tracks which form a good vertex with the trigger candidate), and prevents a class of problems caused by Turbo candidate tracks not being pointer-equivalent to `PersistReco` tracks.

In principle, Tesla then only needs to convert from the online `.mdf` format to a DST, but in practice Tesla implements streaming in a similar manner to the Stripping. This loosely groups together lines by physics category and saves each group in separate output files, with the intention of reducing the number of files individual analysts have to process.

Both the online implementation of the Turbo persistence model and the offline implementation of Tesla make extensive use of the μDST cloning framework (which gives in various `MicroDST*` packages under the `MicroDST` hat in `Phys`). At the highest level, the framework consists of algorithms which can each clone some `KeyedContainer`, from `/Event/Some/Location` to `/Event/<prefix>/Some/Location`, using a cloner tool. The same algorithm may also use other cloner tools to clone the 'dependencies' of each `KeyedObject` in a similar manner. For example, `Particle` containers can be cloned with the `ParticleCloner` algorithm, which will also clone associated `Vertex` and `ProtoParticle` objects with implementations of the `ICloneVertex` and `ICloneProtoParticle` tools.

In HLT2, the cloner framework is used to copy the set of `Particle` objects created by firing Turbo lines, as well as subsets of the reconstruction requested by lines, from their original locations to those under the `/Event/Turbo` prefix. All objects under this prefix are then packed and persisted to the `DstData` bank. The introduction of the cloner framework into HLT2 in 2017 allowed for the persistence of only the parts of the reconstruction relevant to offline analysis, as opposed to 2016 when only the whole, original containers could be persisted.

In Tesla, the cloners are used for streaming, as in the Stripping. Each HLT2 line is assigned to a stream, and then all of the locations requested by that line are copied to the respective stream for each event in which the line fired. The list of locations requested by a line, which includes the location of the `Particle` objects used to the make the trigger decision, is defined by the TCK. The mapping from line name to output stream is defined in the Tesla configuration.

6.2.3 Tesla

The Tesla application converts from the `.mdf` format output by HLT2 into something which is easily analysable in DaVinci. Ignoring streaming, in 2017 this means:

1. Decoding the `DstData` bank and persisting the resulting packed containers (so that user jobs don't need to decode the bank themselves);
2. Decoding the HLT1 and HLT2 `HltSelReports`, removing Turbo reports from the latter, and persisting both. The `RecSummary` is encoded in the `HltSelReports` in HLT2, so this is decoded in Tesla and the resulting `Rec/Summary` location is also persisted.
3. Juggling the raw event and persisting the resulting locations (most raw banks are removed by the `TURBO` stream writer in HLT2).

In DaVinci user jobs, the packed containers from Tesla are unpacked, and symbolic links are created from the HLT2 reconstruction output locations to those of Brunel, such that most standard analysis tools do not need to be configured specifically to analyse Turbo data (as, for example, they expect `Track` objects to be in `Rec/Track/Best` rather than `Track/Best/Long`).

For simulated events, Tesla also creates relations tables for `ProtoParticle` $\leftrightarrow$ `MCParticle` and `CaloCluster` $\leftrightarrow$ `MCParticle` matching. For μDST output, the list of `MCParticle` and `MCVertex` objects are filtered, based on the simulated signal process and the MC objects that can be associated to the HLT2 reconstruction.

Streaming in Tesla is implemented using the μDST cloner framework, whereby HLT2 lines are grouped into streams, and each stream is an output file containing only the information required by the particular HLT2 lines.

6.2.4 Looking forward

The size of the `HltDecReports` could be reduced if one would save the `HltDecReport` only for trigger lines which fired in an event. It would imply the removal of the information in which selection step a trigger line failed. However, this information is hardly used offline and the information is anyhow biased as it is only available in events which got selected by another trigger line. Given that we expect more than a thousand trigger lines in the upgrade this might be a sizeable reduction. The information is useful in the online monitoring of the Hlt and can be persisted in monitoring histograms.

The `HltSelReports` store reconstructed information in a specialised format that is incompatible with standard analysis tools: one cannot simply retrieve particle momentum from the `HltSelReports`, for example. Instead, it seems desirable to save the information about trigger candidates in the default event format to benefit from developments there. A first step towards this was made in 2017 and is described in the the next section about Turbo persistency.

Packing, used both in HLT2 and Tesla, is expected to be play an important role in the upgrade, as it can compress the data using knowledge about the detector noise and its dynamic range. For example, a `double` may not be required for storage if a measurement is not very precise, but it is useful to use when performing computations in memory. The cloner framework, on the other hand, can be considered as an implementation detail, existing because Gaudi can only persist entire containers of objects. As we often require only certain objects within containers to be persisted, *e.g.* only the information requested by Turbo lines, we must first clone that information to new containers. This requires time, to perform the copy operations, and consumes additional memory, that required to hold the cloned objects. There is no current plan for replacing the cloner framework, although the possibility of alternatives should be investigated.

Given that the full raw event can only be saved in rare cases it has to be ensured that the possibility to perform detector calibrations is available, *e.g.* one could imagine to refit tracks with a different alignment or apply a better calibration of the calorimeter offline. Especially, the calorimeter calibration needs more data than can be collected in one fill, making it difficult to provide the best calibration before HLT2 runs.

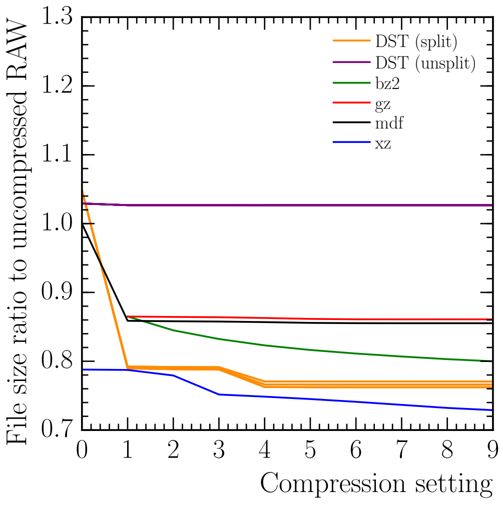
A more invasive idea would be to only save the `LHCbIDs` and associated detector response for selected candidates and discard all reconstructed information at the end of HLT2. The information then could be recreated offline rerunning the same algorithms as done online. This will be an optimisation between offline storage and CPU resources and it needs to be ensured that enough information is saved to recreate the full information.

Currently, data from the pit is not compressed before being sent offline. The `.mdf` format supports compression of raw information within events, and the `DstData` bank used for writing Turbo information is compressed by the `HltPackedDataWriter`, but further gains may be possible simply by compressing whole `.mdf` files. This is illustrated in Fig. 2. Using the `xz` compression algorithm, it has been seen that savings of 10–15 % can be made with respect to the current strategy in 2017.

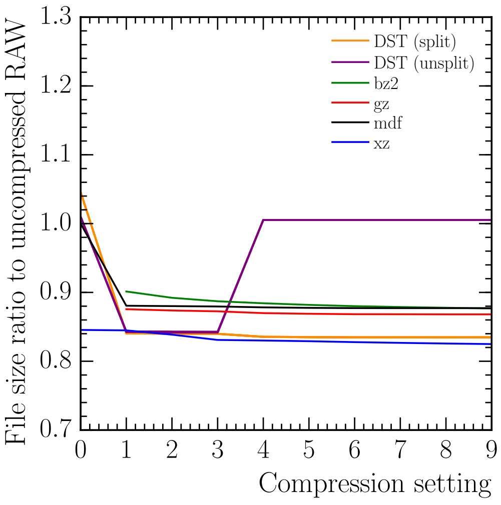
It is a matter of discussion how much of the work done in Tesla could be moved into Moore. For example, Moore already streams events into *e.g.* the `FULL` and `TURBO` streams, and preserves only certain information in certain streams. Streaming into `TURBO` could become more granular to prevent the need for streaming in Tesla. In addition, it would be nice to avoid the encoding of packed containers in Moore, which is followed by decoding in Tesla, and rather just write and read packed containers directly. The inability to do this today is a feature of the `.mdf` format.

7 Conclusion

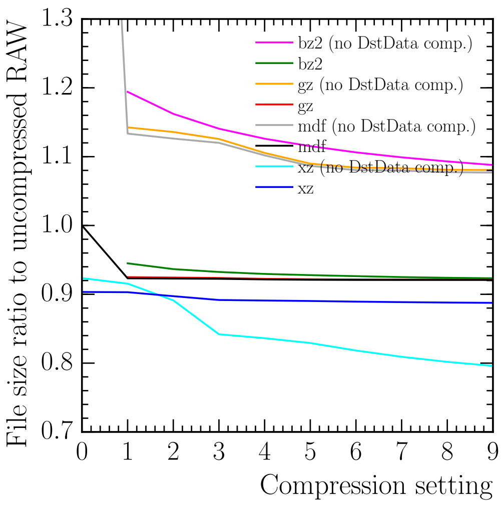
This note has outlined the current state of the baseline trigger and reconstruction strategy as of the end of 2017. The next steps, to be implemented in future documents, is a performance benchmark of the implemented algorithms, and studies of the HLT selections.



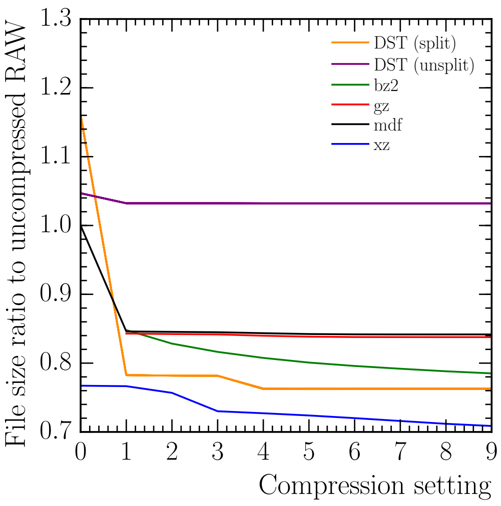
(a) Full stream.



(b) Turbo stream.



(c) Turbo++ stream.



(d) TurboCalib stream.

Figure 2: Comparison of compression ratios obtained by a variety of compression algorithms on the output of the Run II HLT2. Figure 2a shows the performance on the full stream, which contains the full raw event information and Run-I-like `HltSelReports`, Fig. 2b is evaluated on Turbo data that includes Turbo `HltSelReports` and Fig. 2c takes as input the subset of Turbo data that have the full reconstructed event information persisted in the `DstData` raw bank (Turbo++). Figure 2d is evaluated on the TurboCalib stream, which includes both the full raw event information and Turbo `HltSelReports`. In Fig. 2c the curves labelled “no DstData comp.” show the results when the internal compression of the `DstData` raw bank is disabled. The DST configuration uses ROOT's LZMA compression, and multiple DST curves correspond to different setting for the ROOT basket and buffer sizes.

References

[1] LHCb collaboration, *Rec gitlab repository*, https://gitlab.cern.ch/lhcb/Rec.

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

[3] O. Callot and M. Schiller, *PatSeeding: A Standalone Track Reconstruction Algorithm*, Tech. Rep. LHCb-2008-042. CERN-LHCb-2008-042, CERN, Geneva, Aug, 2008.

[4] E. E. Bowen, B. Storaci, and M. Tresch, *VeloTT tracking for LHCb Run II*, Tech. Rep. LHCb-PUB-2015-024. CERN-LHCb-PUB-2015-024. LHCb-INT-2014-040, CERN, Geneva, Apr, 2016.

[5] O. Callot and S. Hansmann-Menzemer, *The Forward Tracking: Algorithm and Performance Studies*, Tech. Rep. LHCb-2007-015. CERN-LHCb-2007-015, CERN, Geneva, May, 2007.

[6] M. Needham and J. Van Tilburg, *Performance of the track matching*, Tech. Rep. LHCb-2007-020. CERN-LHCb-2007-020, CERN, Geneva, Mar, 2007.

[7] M. Needham, *Performance of the Track Matching*, Tech. Rep. LHCb-2007-129. CERN-LHCb-2007-129, CERN, Geneva, Oct, 2007.

[8] A. Davis, M. De Cian, A. M. Dendek, and T. Szumlak, *PatLongLivedTracking: A tracking algorithm for the reconstruction of the daughters of long-lived particles in LHCb*, Tech. Rep. LHCb-PUB-2017-001. CERN-LHCb-PUB-2017-001, CERN, Geneva, Jan, 2017.

[9] M. De Cian, S. Farry, P. Seyfert, and S. Stahl, *Fast neural-net based fake track rejection*, Tech. Rep. LHCb-PUB-2017-011. CERN-LHCb-PUB-2017-011, CERN, Geneva, Mar, 2017.

[10] M. Kucharczyk, P. Morawski, and M. Witek, *Primary Vertex Reconstruction at LHCb*, Tech. Rep. LHCb-PUB-2014-044. CERN-LHCb-PUB-2014-044, CERN, Geneva, Sep, 2014.

[11] A. Dziurda, V. V. Gligorov, and M. Witek, *Primary Vertex Reconstruction for Run II of data taking*, Tech. Rep. LHCb-INT-2016-034. CERN-LHCb-INT-2016-034, CERN, Geneva, Jul, 2016.

[12] B. Storaci, *Optimization of the LHCb track reconstruction*, J. Phys. : Conf. Ser. **664** (2015) 072047. 6 p.

[13] LHCb collaboration, *Brunel gitlab repository*, https://gitlab.cern.ch/lhcb/Brunel.

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

[15] G. Lanfranchi *et al.*, *The Muon Identification Procedure of the LHCb Experiment for the First Data*, Tech. Rep. LHCb-PUB-2009-013. CERN-LHCb-PUB-2009-013, CERN, Geneva, Aug, 2009.

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

[17] LHCb collaboration, *LHCb Upgrade Software and Computing TDR*, CERN-LHCC-2018. in preparation.

[18] R. Quagliani, Y. S. Amhis, P. Billoir, and F. Polci, *The Hybrid Seeding algorithm for a scintillating fibre tracker at LHCb upgrade: description and performance*, Tech. Rep. LHCb-PUB-2017-018. CERN-LHCb-PUB-2017-018, CERN, Geneva, May, 2017.

[19] S. Esen and M. De Cian, *A Track Matching Algorithm for the LHCb upgrade*, Tech. Rep. LHCb-PUB-2016-027. CERN-LHCb-PUB-2016-027, CERN, Geneva, Dec, 2016.

[20] R. Aaij *et al.*, *Upgrade trigger: Biannual performance update*, Tech. Rep. LHCb-PUB-2017-005. CERN-LHCb-PUB-2017-005, CERN, Geneva, Feb, 2017.

[21] M. De Cian *et al.*, *Status of HLT1 sequence and path towards 30 MHz*, Tech. Rep. LHCb-PUB-2018-003. CERN-LHCb-PUB-2018-003, CERN, Geneva, Mar, 2018.

[22] D. H. Cámpora Pérez and O. Awile, *An efficient low-rank Kalman filter for modern SIMD architectures*, submitted to Concurrency and Computation: Practice and Experience.

[23] B. Couturier and T. Szumlak, *Simplified Detector Description for LHCb Upgrade*, Tech. Rep. LHCb-PUB-2017-003. CERN-LHCb-PUB-2017-003, CERN, Geneva, Jan, 2017.

[24] LHCb collaboration, *LHCb PID Upgrade Technical Design Report*, CERN-LHCC-2013-022. LHCb-TDR-014.

[25] LHCb collaboration, R. Aaij *et al.*, *Improved limit on the branching fraction of the rare decay $K^0_\mathrm{S} \to \mu\mu$*, Eur. Phys. J. **C77** (2017) 678, `arXiv:1706.00758`.

[26] R. Aaij, G. Raven, and M. Merk, *Triggering on CP Violation: Real-Time Selection and Reconstruction of $B_s \to J/\psi\,\phi$ decays*, Jan, 2015. Presented 07 May 2015.