Tracking in the High Level Trigger for CMS Phase 2

Liv Helen Våge
Supervisor: Dr. Alex Tapper

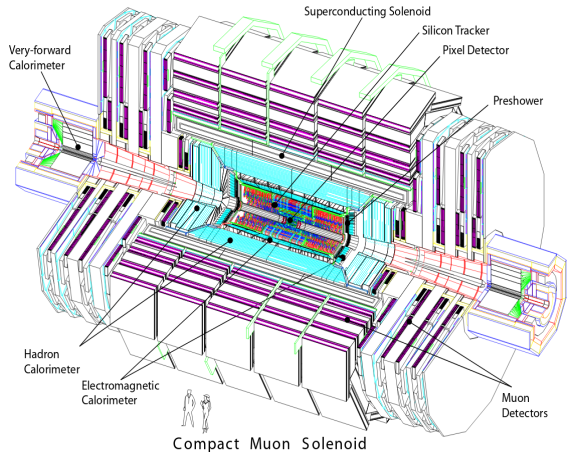
June 2020

Abstract

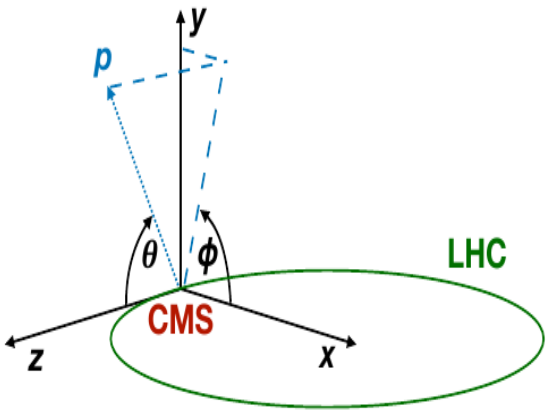
The LHC will be upgraded starting 2025 to begin a new run in 2028. This will increase the number of simultaneous proton-proton collisions to a maximum of 200. Reconstructing charged particle tracks in this environment is very challenging. To reduce the data written to storage, CMS has two triggers. Both of these need to be updated for the high luminosty LHC. This document focuses on the second stage trigger, the High Level Trigger (HLT). The algorithms used in the HLT are outlined, and alternatives for the update are discussed. Performance modelling of the HLT is presented, and four high level functions that together take up about 70% of the online CPU time are identified. It is suggested that the major reason for the increase in CPU time with more simultaneous collisions is the number of times the functions are called, instead of an overall increased computational complexity in the functions. Parallelising the tracking on FPGAs is suggested as a good method for accelerating the HLT tracking.

1 Introduction

The LHC [1] collides particles at an instantaneous luminosity of $10^{34}\text{cm}^{-2}\text{s}^{-1}$. The Compact Muon Solenoid (CMS) [2] produces about 40 TBytes of data per second, but only has storage capacity of 10 TBytes per day [3]. To reduce the amount of data CMS has a two-level trigger system, filtering events from 40 MHz to 100 kHz [4]. These triggers will be upgraded. The LHC discovered the Higgs boson eight years ago [5], and is now planned to undergo large changes, in order to start a new run in 2028, called Phase 2 [6]. This will allow searches for heavier mass particles and smaller cross sections, extending Higgs physics, searches for super symmetry and beyond the standard model physics. The instantaneous luminosity will be upgraded to $7.5\text{x}10^{34}\text{cm}^{-2}\text{s}^{-1}$. In the most recent run, there has been an average of 40 simultaneous proton-proton collisions. Most of these are not hard scatter events, and the events that are not of interest are known as pileup events. In Phase 2 the pileup will reach 200 [1, 6, 7]. Reconstructing events in this high-occupancy environment is an exponentially harder challenge than at the current pileup. The short latency of the triggers also pose several constraints on the reconstruction algorithm. CMS is currently implementing the first level of the trigger, L1T, in Field Programmable Gate Arrays (FPGAs). The second level, the High Level Trigger (HLT), is implemented on a farm of 30 000 CPUs [8]. One of the main trigger challenges in a high pileup environment is the charged particle tracking implemented in the HLT. Relying on current techniques is not enough; "Assuming only technology improvements and maintaining existing techniques, the offline software and computing areas would fall short by a factor of 4(12) of the resources needed for the challenging conditions expected in Phase-II at 140(200) pileup" [9]. This document explores how the tracking in CMS HLT works. Some approaches for acceleration are outlined, and performance studies of the CMS tracking software are given. "Performance" refers here to CPU time - for efficiency of track reconstruction, please refer to e.g. Ref. [10]. Based on the performance modelling, ideas for further exploration are suggested, with a focus on FPGAs.



(a) The CMS detector



(b) Coordinate system at CMS

Figure 1: Fig. (a) illustrates the structure of the CMS detector, the largest silicon tracker in the world. Figure from [1]. Fig. (b) shows the global coordinate system of CMS. Particle collisions occur at x = y = 0. Figure from [11].

2 The CMS detector

The CMS detector is illustrated in Fig. 1. A silicon pixel tracker is closest to the beam line. The high track multiplicity at this point requires fine spatial granularity, so the pixel tracker has 66 million 2D pixels, distributed in three layers at a distance of 4.3,7.2, and 11 cm from the beam line. This is responsible for track seeding, described in section 4.2. It is followed by ten layers of silicon strips, reaching a radius of approximately 130 cm. These are responsible for track building in HLT. Further out are the calorimeters. The ECAL consists of $PbWO_4$ crystals, arranged as a barrel of 61200 crystals and two endcaps of 10764 crystals each. The HCAL consists of a barrel, outer layer, forward layer and endcaps. These are all enclosed in a superconducting solenoid, responsible for bending the charged particle trajectories. The muon detector is the outer layer of CMS, and is designed to detect muons and measure their transverse momentum, p_T. It is divided into chambers consisting of resistive-plates, cathode strips and drift tubes, so a muon will ionise several chambers, and provide redundancy [1, 4]. The calorimeters and muon detectors are used for both L1T and HLT. CMS uses a right-hand coordinate system, where z is in the direction of the beam line, x goes radially inwards, ϑ is the polar angle and φ is the azimuthal angle. This is illustrated if Fig. 1. It is also convenient to define pseudorapidity, $\eta = -\ln\left[\tan\left(\frac{\vartheta}{2}\right)\right]$.

3 L1T

The first step of data reduction is the Level 1 Trigger [4, 12], which receives data at the full collision rate, 40 MHz, and reduces this by a factor of 400. The L1T consists of FPGAs placed underground, close to the detector. Collision data is stored in buffers in ASICS (integrated circuit chips designed for a specific use) until a trigger signal is received. The depth of the buffers limits the latency to 4 μs. The FPGAs are ideal for such short latency tasks, due to the high degree of parallelisation, described further in section 6.3. A trigger menu, consisting of up to 500 different parameters, can be set to accommodate the interests of the experiment. L1T has a calorimeter trigger and a muon trigger, which are combined into a global trigger. The main task of the calorimeter trigger is to sort raw data into lists of electron/photon, hadronic τ, and jet candidates. Data is transferred from ECAL off detector in high-speed optical fibres. The signals from windows of 5x5 crystals are summed into trigger towers, for input to the trigger. Similarly for the HCAL, Cherenkov light produced in the steel absorber of HCAL forward and the barrel is transported in optical fibres and summed over 5x5 cells. The trigger passes on e/γ candidates and their energy to a

global calorimeter trigger, which sorts the candidates further and identifies jets. The muon trigger uses all three muon chambers. Information from these three stages is combined in the global muon trigger, which rejects candidates that pass only one chamber. A quality score is also assigned to the candidates. The global calorimeter and global muon triggers are then combined to a final, global trigger. This is where the selection menu is implemented, which will typically specify the type of candidate, and put restrictions on the total energy and transverse momentum of the event. If accepted, the data is passed on to the HLT [4].

4 HLT

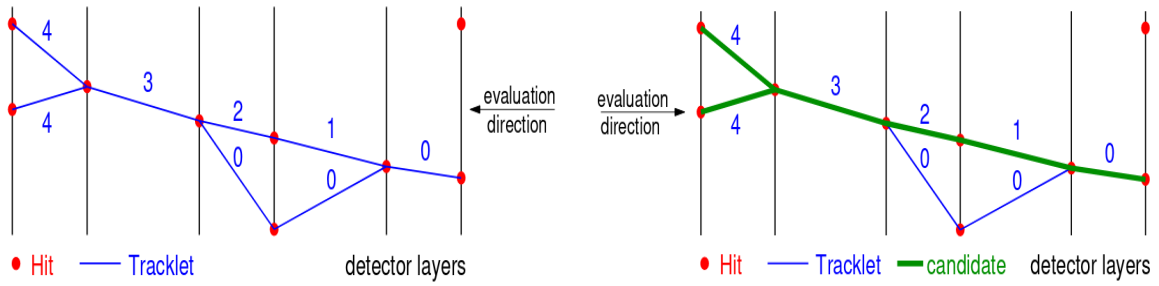
The HLT [4] is based on the same algorithms as the offline reconstruction, but is limited to a latency of 50 ms [13]. Events are processed on single, multi-core CPUs. One event is processed on one CPU, but is typically multithreaded into four parts, so track building happens in parallel. The HLT uses the full granularity of the detectors, and is centered around the idea of paths. Different physics objects need to be processed differently by the HLT, and this is defined in several sequences of modules, known as a path. Each module performs a single task, and they are combined in sequences which carry out an overall task. The modules and module sequence are defined in configuration files, given as cmsRun executables. The configuration may be complex. For example, the 2012 data taking for the HLT used a configuration of 400 different paths, with a total average of 2200 modules. It is therefore beyond the scope of this document to describe any of these in detail. The paths are, however, based on four common main steps; seeding, track building, track fitting and track selection. The paths increase in complexity, so as to reduce the amount of data that reaches the time consuming steps. For instance, the calorimeters and muon detector are used to reduce the amount of data before the tracking begins. After an event is accepted, the event is sent to a storage manager to be archived [4, 14, 15].

4.1 Pixel hit reconstruction

The starting point and initial uncertainty of a track is defined by a seed, found from the pixel detector. Before seeding begins, 3-D pixel hits must be reconstructed. Signal sizes over an adjustable threshold are read out. If only one pixel has been hit, the centre of the pixel is read as the hit position. Because of the angle of incidence and Lorentz drift, a track may also deposit charge along several adjacent pixels. The position of the hit is determined by considering the width of the cluster of pixels in two dimensions, while adjusting for Lorentz drift. The pixel hits are then used to define the initial parameters of the tracks, known as track seeds [16].

4.2 Seeding

The seeds are created in an iterative manner, starting with the tracks that are easiest to reconstruct, i.e. prompt tracks with high p_T, that originated close to the beam line. Phase 2 will use six tracking iterations dubbed HighPtQuadrupletpixel, HighPtTripletpixel, DetachedQuadruplet, LowPtQuadruplet-pixel LowPtTripletpixel, and Muon inside-out. This document will focus on the first two, since they are responsible for the vast majority of the tracks [10]. First, the layers used for seeding are specified, which for Phase 2 are the first four layers of the pixel detector. A global tracking region is defined based on what is compatible with the beam spot. Hit pairs and triplets are found by extrapolating the z and φ values of one or two hits to a subsequent layer. Quadruplets are found using cellular automata [17], explained in Fig. 2. Using the inner tracker for seeding ensures a high reconstruction efficiency because particles lose minimal energy at the start of their trajectory. Although there is a higher track occupancy in the pixel detector than the strip detector, the high granularity also ensures a lower channel occupancy. A Kalman filter [18] is then used to find the track parameters, as explained in section 4.3.1.



(a) Evolution of cellular automata states

(b) Track seeds

Figure 2: A track triplet is considered a cell. Its neighbouring cells are defined by the cells that share any same two hits and have similar momentum estimates. All cells start with a trivial state 0. If a cell has a neighbouring cell to the left with the same state, 1 is added to the state. This is repeated iteratively until there are no neighbouring cells with an equal state. Starting by the highest count, hits with decreasing counts are read out. This may lead to multiple hits in a layer, as shown in b). Seed candidates are chosen by ranking these tracks by a quality measure. Figure from [19].

4.3 Track building

Tracks are built in a loop over all the seeds. The trajectory of a seed is extrapolated to a subsequent layer, assuming a perfect helix, and accounting for material effects like multiple scattering. First, layers with possible compatible hits are identified. Second, silicon modules compatible with this are identified. Modules often slightly overlap, so groups of mutually exclusive modules are identified, and overlapping modules are assigned to different groups. Hits are assigned to a track based on their χ^2 compatibility with the track. Ghost hits can also be added to account for possible missed hits due to detector inefficiencies. A track is discarded if it contains too many missing hits, the p_T drops under a set threshold, or the χ^2 for the track is too high. The main challenge of track building is the combinatorics introduced by having several hits compatible with a track. All of these hits are branching points that need to be explored. This is shown in Fig. 4. The exponential explosion of possible combinations help explain why the current algorithm running on the current hardware will be insufficient for Phase 2. In a high pileup environment, there may be many compatible hits, causing many branching points, and possible tracks that must be built. After a track has been built, the hits are removed from the pool of hits. The general architecture is shown Fig. 4. The track is subsequently fit using a Kalman filter.

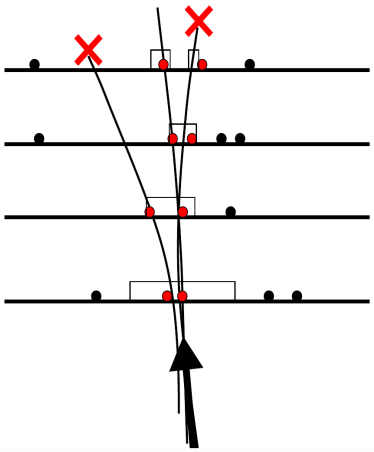
4.3.1 Kalman filter

Kalman filtering is a linear estimation technique from the 1960s, commonly used in vehicle navigation. It is also the de facto method of track building, both in HLT and offline. It is sequential, so hits can be removed as tracks are reconstructed. It is robust to uncertainties, and allows for simple matrix calculations.

Since it is the essence of track building, we will consider it in some detail. Three dimensional trajectories follow helical paths, and are described using five parameters; $x = (\frac{q}{p}, \vartheta, \varphi, x_\perp, y_\perp)$, as illustrated in Fig. 1. This is divided into two parts, a state vector x, and a measurement vector m.

$$x = \left((2R)^{-1}, \varphi_0, \tan\vartheta, z_0\right) \qquad m = (\varphi, z) \tag{1}$$

R is the radius of curvature of the track, φ the azimuthal angle, z the longitudinal position of the trajectory, and ϑ is the dip angle. The values with a subscript of zero refer to initial values. Initial estimates of these parameters are given by the centre of the beam and the beam spot uncertainty.



(a) Assigning hits to a track

```
while candidate pool is not empty do:
  initialise empty temporary candidate pool;
  for all candidates in candidate pool do:
    find hits compatible with candidate;
    for all compatible hits do:
      update candidate state with hit;
      if updated state is finished then:
        add candidate to result tracks;
      else:
        add candidate to temporary candidate pool;
  sort temporary candidate pool by quality;
  discard all but best N candidates;
  replace candidate pool with temporary candidate pool;
return result tracks;
```

(b) Pseudocode of the track building algorithm

Figure 3: Fig. (a) shows a track seed illustrated by a black arrow. Regions with hits that would be compatible with the track, as calculated by previous layers, are shown by the boxes. Compatible hits are shown in red. The tracks have to be extrapolated to all possibilities. Tracks are then rejected, e.g. the track to the left has too many missing hits, and the track to the right has a too high χ^2-value. Figure from Ref. [20]. Fig. (b) contains pseudocode of the track building. Figure from Ref. [21].

The next step is to propagate the state to a new layer of the detector using track helix equations. Further information on these can be found in [22]. Each k represents a layer, and the helix equations are represented by the function f.

$$x_k^{k-1} = f_{k-1}(x_{k-1}) \tag{2}$$

This can also be written in terms of the change of the state vector

$$x_k^{k-1} = \mathbf{F}_{k-1} x_{k-1}, \mathbf{F} = \frac{\partial f}{\partial t}, \tag{3}$$

from which it follows that the covariance matrix C is

$$\mathbf{C}_k^{k-1} = \mathbf{F}_{k-1} \mathbf{C}_{k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1}, \tag{4}$$

where Q accounts for multiple scattering. The residuals of the projected state are found by comparing it to the measured state. The measurements may be in different units and need to be translated by a matrix $\mathbf{H}$.

$$r_{k-1}^k = m_k - \mathbf{H}_k x_k^{k-1} \tag{5}$$

The covariance of the residuals is given by the covariance matrix $\mathbf{V}$ of the measurement m plus the covariance of the parameters in the same units.

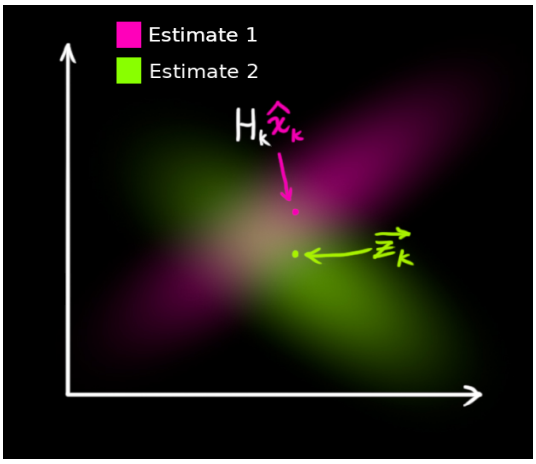
$$\mathbf{R}_k^{k-1} = \mathbf{V} + \mathbf{H}_k \mathbf{C}_k^{k-1} \mathbf{H}_k^T \tag{6}$$

Both the projected state and measured state have uncertainties, defining regions in space where the new measurement should be. Analogous to two probabilities both being true, we find the overlap of the regions by multiplying them together. This gives what is called the Kalman Gain $\mathbf{K}$, which then allows one to propagate the state, allowing for uncertainties both in the measurement and the prediction.

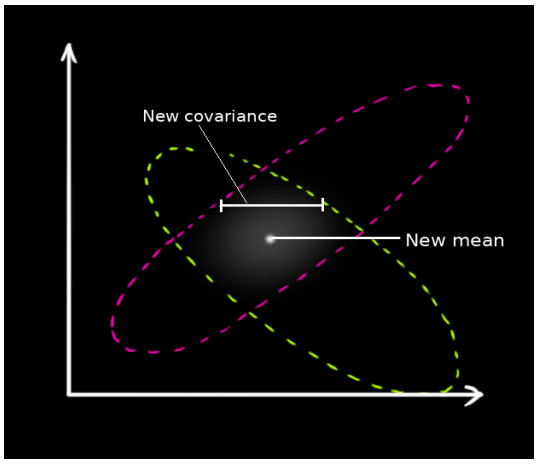
$$\mathbf{K}_k = \mathbf{C}_k^{k-1} \mathbf{H}_k^T \left(\mathbf{R}_k^{k-1} \right)^{-1} \tag{7}$$

The new state is then:

$$x_k = x_k^{k-1} + \mathbf{K_k} r_k^{k-1}. \tag{8}$$



(a) Assigning hits to a track



(b) Pseudocode of the track building algorithm

Figure 4: Fig. (a) shows a track seed illustrated by a black arrow. Regions with hits that would be compatible with the track, as calculated by previous layers, are shown by the boxes. Compatible hits are shown in red. The tracks have to be extrapolated to all possibilities. Tracks are then rejected, e.g. the track to the left has too many missing hits, and the track to the right has a too high χ^2-value. Figure from Ref. [20]. Fig. (b) contains pseudocode of the track building. Figure from Ref. [21].

The covariance of this state is given by

$$\mathbf{C_k} = (\mathbf{I} - \mathbf{K_k H_k})\mathbf{C_k^{k-1}} \tag{9}$$

The χ^2 of a track is given by

$$\chi^2_+ = \mathrm{r}_\mathrm{k}^{\mathrm{k-1T}} \left(\mathbf{R}_\mathrm{k}^{\mathrm{k-1}}\right)^{-1} \mathrm{r}_\mathrm{k}^{\mathrm{k-1}} \qquad \chi^2_\mathrm{k} = \chi^2_{\mathrm{k-1}} + \chi^2_+ \tag{10}$$

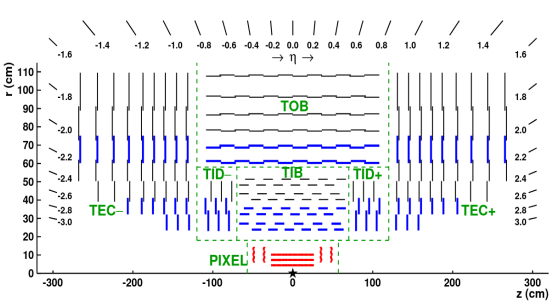
Since there are five parameters, these matrices will take dimensions of 5x5, 5x2 etc. The Kalman filter is called many times in HLT. It is called when finding the initial track parameters of the seed, when updating the state of a track while building, and when finding the parameters of a built track, known as fitting. It is also called by the primary vertex fitter. As will be discussed in section 6, other ways to fit lines have been considered, but the Kalman filter has proven robust [18, 21].

4.4 Track fitting

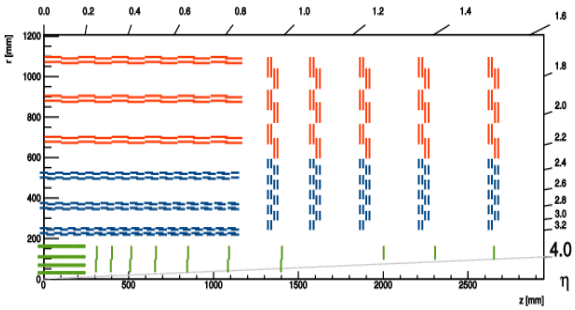
After the track building, a list of hits compatible with a track has been identified. A Kalman filter is again used to fit a track to the hits. Starting from the innermost hit, the filter is propagated through the known seeds, just as described in section 4.3.1. This is supplemented by a smoother - another Kalman filter is initialised with the result from the first Kalman filter, and runs from the outermost hit in. The parameters of the two filters are combined.

4.5 Track selection

Not all of the fitted tracks are real tracks, and a set of selections are made based on the χ^2 of the track, and its compatibility with the beam spot and the point of the track closest to the primary vertex. Hence, primary vertices must be identified. Online, this is done with deterministic annealing [23, 24]. Associating tracks to a vertex can be seen as a clustering problem. Inspired by statistical mechanics, the problem is analogous to a system in contact with a heat bath at a temperature T, with a particle reservoir of a given chemical potential. The χ^2 of the position between the z-coordinate of the closest



(a) Tracker for Phase 1



(b) Tracker for Phase 2

Figure 5: Fig. a) shows the tracker as it is now, with pixel layers in red and silicon strips in blue. Figure from [16]. Fig. b) shows the pixel tracker in green and the outer tracker in red and blue. The blue correspond to pixel-silicon modules, and the red to silicon-silicon modules. The silicon strips r ≈ 200 to 600 are the tilted barrel layer, above is the barrel layer, and the strips from z ≈ 1400 to the end is the endcap. Figure from [7]

point of approach of the track and the beam spot centre is analogous to energy. One can then minimise the free energy of the system with respect to the z-position of the vertices. The initial state has a very high temperature, which corresponds to all of the tracks originating from one vertex. From statistical mechanics, a system undergoes a phase transition if a perturbation to the system brings the free energy to another local minimum. The perturbation is here achieved by decreasing the temperature incrementally. When the temperature falls beneath a certain threshold, a vertex splits into two vertices, separated by a small distance. Eventually, many vertices will converge on the same location, and the primary vertex parameters can be found. The tracks will then be clustered within these vertices by updating their weights. Tracks with low weights are discarded [16, 24]. Track selections are applied, and the track is written to offline storage.

5 Phase 2

CMS will be upgraded in steps starting in 2025, expected to reach a luminosity of $7.5\text{x}10^3 4\text{cm}^{-2}\text{s}^{-1}$ and pileup up to 200. According to simulations, as they are now, the pixel detector would significantly degrade its resolution and the strip detector would suffer thermal runaway [7]. The tracker will therefore be completely replaced, as will the calorimeter endcaps. The tracker will be designed to withstand high doses of radiation. The pixel tracker will have smaller pixels, 25x100 or 50x50 μm^2, to maintain the same occupancy in the high pileup environment. It will extend from $\eta = 2.4$ to $\eta = 4$. The outer tracker will see the number of silicon strips increase by a factor of around 5. The layout of the tracker is shown in Fig. 5. A major change is that the outer tracker will also be the basis of tracking as a part of L1T. Reading out all hits for tracking at 40 MHz is unfeasible, so new modules have been designed that can reject low momentum particles. Two closely spaced sensors read hits out to front-end ASICS that correlate hits in the two. Such pairs are called stubs, and the modules are known as p_T modules. By the curvature of the track between the two sensors, it can reject tracks with momentum below a certain threshold, likely to be set at 2 GeV. This is a data reduction of an order of magnitude, and sufficient to pass the hits on to L1T. The outer tracker is therefore populated with p_T modules in the first three layers from the centre, where resolution for this is sufficient. The L1T latency will increase by a factor of four. Conceptually, L1T will stay quite similar, with information from the calorimeters and muon detectors, and now also tracker, being combined into a global trigger. The way of combining these will be similar to that of HLT. Primary vertex reconstruction and pileup subtraction will also take place [7, 25]. To cope with the challenge of increased pileup, the latency for the HLT will be increased from 50 ms to 200 ms. How to change HLT so that it can cope with the high pileup has not been determined, though many different methods have been explored [21].

6 Alternatives for HLT Phase 2

Up until around 2005, increases in computing power generally came from an increase in processor frequency rate. As transistor density increased, power consumption and thus heat started to become a big obstacle. Since then, the industry focus has been to increase performance by parallelising processes [26]. Parallelising the tracking in HLT is a natural step to tackle the high luminosity environment. One could achieve a better tracking performance by parallelising CPUs and CPU threads. However, this would require a massive increase in the number of CPUs used, which is unfeasible. Therefore, one CMS tracker update document describes the possible steps for HLT research as using multiple specialised algorithms, us GPUs, target physics objects in the tracking, or using the L1 tracking as seeds FIXME ref. This section presents some methods that have been explored within these approaches.

6.1 Software methods

To mitigate the combinatorial explosion of matching hits to tracks, one group implemented a Convolutional Neural Net (CNN) for finding hit doublets in the seeding stage. They represented a pixel as a three dimensional matrix of x and y positions, and the signal size of the pixel. They achieved an accuracy of 99.997 on their test set, showing that CNN's may be a promising candidate. The latency of the method was not discussed, which may be a potential challenge [27]. This is one of many ways that HLT can be accelerated in software, but porting it to other devices than CPUs is widely discussed.

6.2 GPU methods

GPUs are graphics processing units, initially created for gaming. CPUs typically consist of a few cores that can handle a few software threads at a time, but GPUs can have hundreds of cores that can handle thousands of threads each. This means GPUs are ideal for highly parallelised, repetitive processes. A CPU typically has a high clock frequency, where a GPU has a high throughput. CPUs are more general, and when GPUs are implemented, CPUs usually orchestrate computations which are fed onto the GPUs. Using GPUs therefore usually means implementing a heterogenous environment, with I/O between CPUs and GPUs. Typically, one must have a certain number of minimum events for this expense to pay off [26].

Several GPU based approaches for HLT have been explored. The cellular automata explained in section 4.2 is a method that lends itself well to GPUs, since it's a small, repetitive computation. One collaboration explored building all the tracks, not just the seed, with this method. To avoid computationally expensive I/O, the seed generation as well as the track builder was ported to GPUs. With events with more than 500 tracks, a speedup of a factor of 64 was achieved compared to the CMS software CMSSW. They achieved a reconstruction efficiency of 80%, but steps were suggested to mitigate this. [17].

The collaboration Patatrack [28] is a group sponsored by NVIDA that looks at porting pixel tracking to GPUs. They redesigned the CMSSW data format for the GPU, and demonstrated that it may be an efficient alternative to CPUs. Their next steps will be to integrate it into CMSSW and study which other algorithms may benefit from being ported to GPUs.

Another project, which has been going on for about six years, is mkfit [29]. It is a collaboration between data scientists at several universities that aims to speed up the Kalman filter tracking. Though e.g. cellular automata approaches have been explored, the Kalman filter is well liked since it deals with material effects well and results in a high tracking efficiency. The mkfit collaboration use multithreaded, vectorized tracking to achieve 6 times the speed of the current algorithm. Vectorisation means applying operations to arrays instead of individual elements. Multithreading can involve multiple instructions in parallel. Parallelising tracking can be challenging since the number of compatible hits in each track may vary, and each possible branching point must be explored. Both of these could lead to parallel elements becoming unsynchronised. The authors behind mkfit solve this by keeping the branching points in non-vectorized functions, and Kalman Filter logic in vectorised functions. Thread scheduling is implemented at several levels. Detector geometry is represented in parametric descriptions rather than lookup tables,

and they try to minimise memory movement. As outlined in section 4.3.1, the Kalman filter involves several matrix multiplications. These are usually vectorisable if they have the same dimensions as the vector registers of a CPU, typically 16 floating point numbers. The Kalman matrices are 5x5 or smaller, so independent tracks from an event are grouped together and propagated in lockstep. There are three levels of parallelisation; events, detector regions and groups of seed tracks. A big draw back at the moment is that 25% of the mkfit processing time is due to data conversion between CMSSW and mkfit. They also implemented the algorithm on a GPU, which was 3 times faster than CPUs, including overhead. mkfit is already an integrated package in CMSSW, and seems like a promising approach [29].

6.3 FPGA methods

An alternative to GPUs are FPGAs. An FPGA is an integrated circuit device with re-configurable logic blocks. They are typically programmed in hardware description languages, such as VHDL. Unlike high-level programming languages, such as C++ or Python, hardware description languages can describe the data path to different parts of hardware, thereby parallelising the computation. This requires extensive understanding of the FPGA hardware, and is time consuming to code, sometimes requiring specifications down to a transistor level. FPGAs offer more predictable performance than a CPU or GPU, since they do not use branch or cache predictors, which may create variable performance. The also consume less power than CPUs and GPUs since they customize data paths. Suitable targets for FPGA computing will typically be when there are real time requirements and iterative process where data is reused [26]. FPGAs are already in use at L1T, so they may be a logical step for HLT.

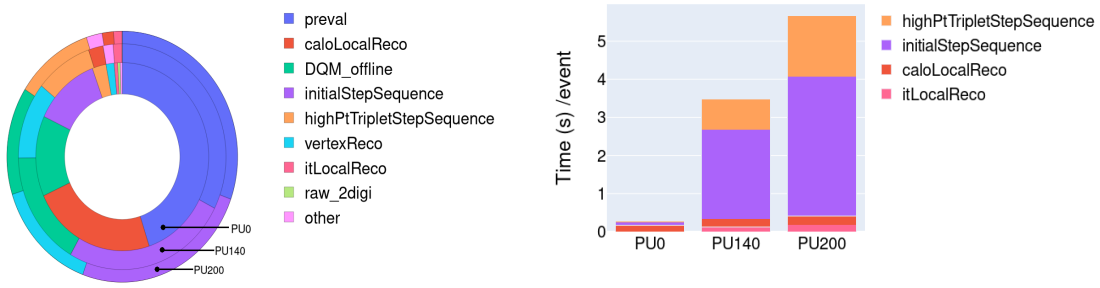
Dr. Sioni Summers implemented the Kalman Filter on Maxeler Technologies' FPGA Data Flow engine [30, 31]. London based Maxeler Techologies have developed a Java based FPGA interface. The high level language allows easy programming of the FPGA, while a data flow model is used to not compromise performance. The data flow approach means a program exists as nodes of computation. There are no storage or load instructions between the nodes, as all data is contained within the nodes. This was popularised by J.B. Dennis in the 1980s, but the high transistor density we see now has allowed this to become efficient. Instead of having a set of sequential instructions, a pipeline is created. A tool called MaxCompiler synthesises the pipeline automatically. This orchestrates a scheduler, which determines what parts of the computation can be parallelised. Dr. Summers ported the Kalman filter equations from section 4.3.1 to an FPGA, while a CPU was responsible for finding compatible hits. The host CPU was set up so it buffered a number of hits and states to minimise I/O between the devices. Due to the variability of the exponents, he used floating point throughout the computation. The update was 4.4 faster on the FPGA than on the CPU. A limiting factor was the I/O [21, 30].

7 Performance modelling

As outlined, porting code to an FPGA can be time consuming. One should first model the performance of the existing code on a CPU to decide what part of the code to port to an FPGA.

7.1 Performance modelling strategy

The results presented in this section are based on simulated Monte Carlo data from a Phase 2 configuration. Top-quark pair production events are used, with a minimum bias pileup. Minimum bias refers to physics events where both inelastic and elastic scattering can occur, as well as secondary scattering etc. All events are run in a single thread on an Intel(R) Core(TM) i7-8665U CPU at 1.90GHz. CMMSW is designed to run on the operating system Centos. Approaches with a virtual machine and an external SSD with this OS were tried, where the latter was used for the results shown here. These two had comparable overheads. Two different ways of timing the code was used. Module based events, as found in section 7.2 were collected with CMSSW's DQM Fast Timer service, where the time resolution was set to 0.5 ms. Timing related to specific source code functions, as found in section 7.3 were timed with Intel's software VTune. A hardware mode which sampled every ms was used. Both of the methods have been designed to



(a) Proportional timing by module type at three different pileups

(b) Timing of module types

Figure 6: We are interested in the sections that increase non-linearly with pileup. As seen from b), the intialStepSequence and highPtTripletStepSequence increase exponentially, going from 12.4 % at PU0 to 25.1 at PU 200 and from 2.6% to 11.0% respectively. The vertexReco also jumps from 1.5 to 14.6%, but this is offline, so not of interest here. The absolute timings of the most interesting module sequences are shown in b).

have minimal overhead. A sample of 10 000 events are usually seen as a good benchmark for performance modelling of CMSSW. 10 000 events at a pileup of 200 would mean a data sample around 400 GB, which was impossible with the setup used. The timings presented in the next section were performed with 1000 events in section 7.2 and 100 in 7.3. A few tests were done to check the difference in timing between 100 events and 1000 events. Some functions that are very quick are not captured with 100 events, and otherwise there was an average uncertainty of about 10-20%. The relative timings of the functions to the total stayed constant. What we are interested in here are not necessarily the absolute timings, but the comparative values between the different timings, so the uncertainty in the absolute timings is considered acceptable.

7.2 Timing by module

First, we want to identify the parts of the software that take up a lot of the CPU time. This can be done by using CMSSW's DQM Fast Timer Service to time each module. With the configuration used here, there were around 145 different modules. These are all part of larger sequences. The timings by sequence is shown in Fig. 6. The first step is raw2digi, which converts the raw detector information to a digital format. The inner tracker and outer tracker hits are reconstructed in itLlocalReco and otLocalReco. caloLocalReco then makes use of the calorimeter towers. The tracking considered here are the first two steps of the Phase 2 tracking setup; labelled initial and HighPtTriplet. These are responsible for finding the vast majority of the tracks. The initial step uses 4 pixel hits and looks for high p_T prompt tracks. HighPtTriplet uses pixel triplets to look for prompt tracks with high p_T. From Fig. 6, it is clear that the highPtTriplet and initialStep sequence take the longest time online. preval is a validtation service, and DQM is the data quality monitoring service. Most of these happen offline, but a subset of the data is also monitored online. This will not be considered here. The section 'other' includes reconstruction of the beam spot and creating the general tracks. The vertexReco timed here happens offline, though some vertex reconstruction occurs inside the initialStepSequence.

Knowing which sequences are relevant, one can go deeper, and consider modules take the longest in each sequence, and in total. This is illustrated in Fig. 7 and Fig. 8.

Fig. 7 illustrate that at a pileup of either 140 or 200, four modules account for almost 70% of the online modules (i.e. excluding DQM offline, vertexREco and preval). The first three modules are the same for both pileups, while the fourth is different. highPtTripletsStepHitTriplets takes about 7.2% at a pileup

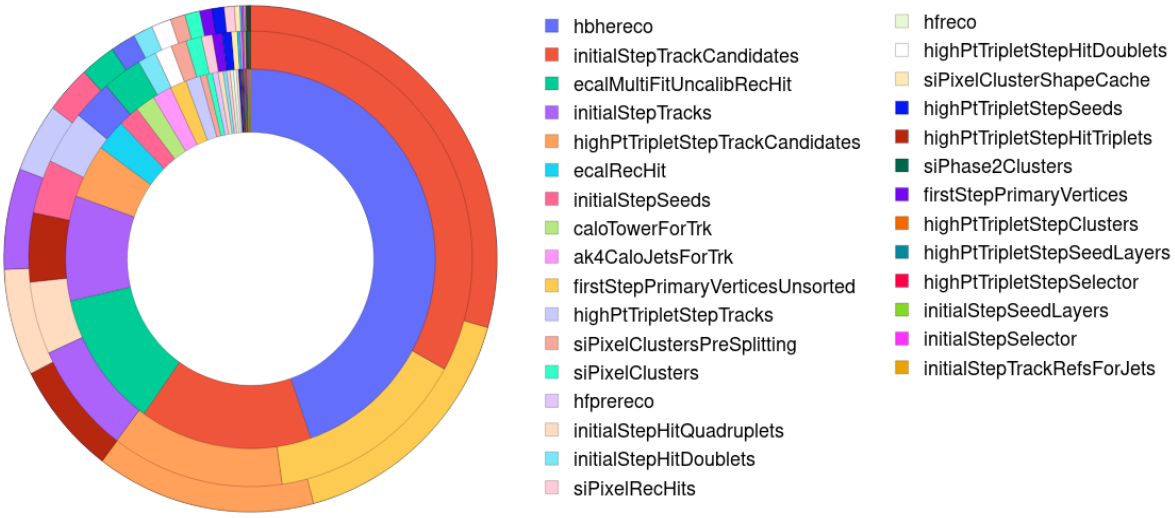


Figure 7: Timing of modules by pileup, where the inner doughnut is no pileup, the middle is PU140, and the outer is PU200. initialStepTrackCandidates, firstStepPrimaryVerticesUnsorted, highPtTripletStepTrackCandidates and highPtTripletsStepHitTriplets take 29.4%, 16.5%, 14.4% and 7.8% of the total CPU time at a pileup of 200.

of 200, compared to 4.9% at pileup 140. initialStepTracks is instead a larger proportion for pileup of 140. It is clear that these four or five modules are severely affected by an increased pileup. In order to understand them better, we can consider the tools they make use of. Each module makes use of an event data producer (EDPRoducer) or function that process the data according to some path. The module call sequence with their associated path is shown in Fig. 8.

Comparing Fig. 7 to Fig. 8, one can see that two of the most time consuming functions call the same tool, the CkfTrackCandidateMaker. This is the part of the algorithm that builds the track, as described in Section 4.3. The other two call the primary vertex producer and the CAHitTripletProducer. The PrimaryVertexProducer recreates the primary vertices using deterministic annealing, as described in section 4.5. The CAHitTripletProducer produces hit triplets using the cellular automata method described in 2. These modules need further exploration. It will be particularly useful to have a close look at the CkfTrackProducer, since this is used by two of the modules, together accounting for about 45% of the online HLT at pileup of 140 and 200. The CMSSW Fast Timer Service does not enable a higher granularity than module level, so one must turn to other tools. Intel's software VTune was chosen.

7.3 Timing with Intel's VTune Profiler

Intel's VTune Profiler is a tool created for performance modelling, and handles combinations of python and C++, such as is found in CMSSW. It is designed to have a minimised overhead, and has a wide range of capabilities. It is integrated with an API called ITT that enables one to start and stop the timing of the code. Being able to do this is quite important since the call stack for CMSSW is very deep and can be difficult to interpret. ITT is a scram tool, so VTune is a natural choice for CMSSW modelling. A slight caveat is that accurate timing relies on hardware counters, which is typically not possible through virtual machines or Docker-like interfaces. To see the callstack, CMSSW must also be recompiled with debug info. We will now use VTune to attempt to find explanations for why the modules identified in the previous section take a long time for high pileup. Again, since the callstacks in CMSSW are very deep, often containing hundreds of files and thousands of functions, only 100 events were used for each pileup sample.

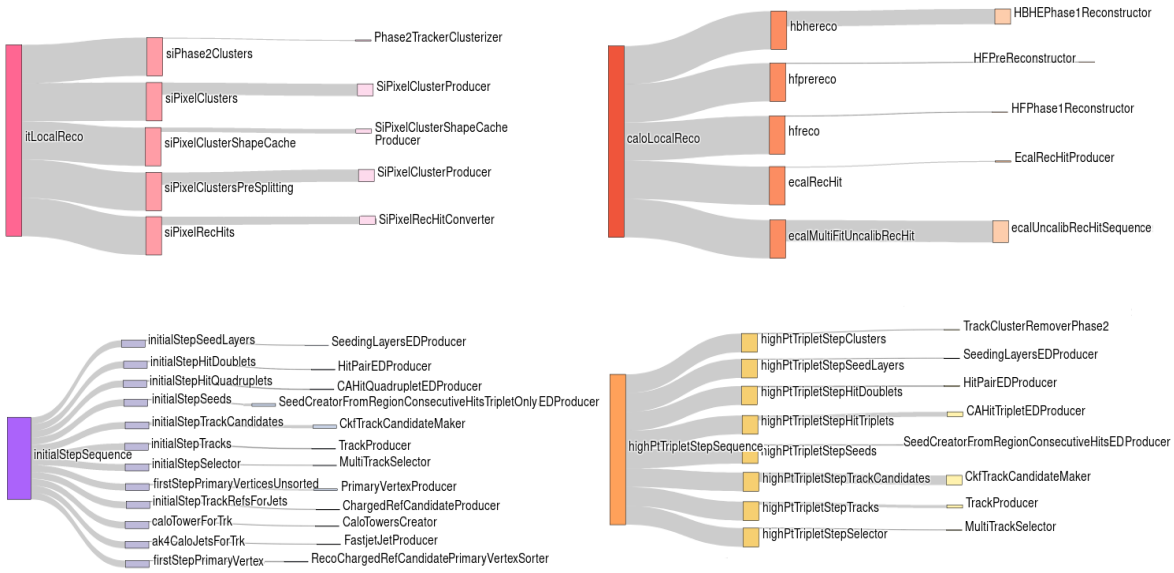


Figure 8: Module sequence of the four most important online sequences at a pileup of 200. The diagrams are called sequentially, starting from itLocalREco and ending at highPtTripletSetpSequence. Each diagram shows the order the modules are called in, starting from the top arm and ending at the bottom. The arms show which path or tool each of the modules are calling. The width of each arm shows the proportion of time that module use of that particular sequence.

7.3.1 CKFTrackCandidateMaker

The CKFTrackCandidateMaker follows seven steps. It sets the event, retrieves the seeds, creates an empty output collections, invokes the building algorithm, cleans the tracks to avoid duplicates, converts the result to track candidates, and writes the output to file. With VTune, this algorithm took 262.5 s for 100 events at at pileup of 200. With the Fast timing service, it took 2.6s per event. This is in good agreement, illustrating that 100 events is sufficient, and that the two services have comparable overheads. The first thing to note is that there is no low level function that took longer than a couple of seconds to run for 100 events. It is therefore unlikely that the track building can be accelerated satisfactorily by porting any one low level function to a GPU or FGPA, since the time for I/O would likely outweigh the small benefit.

The most time consuming part of the CKFTrackCandidateMaker is the track building. Building the trajectory happens in two ways, one is a 'regular' track builder, and the second builds the seed from the outside in, using the information from the first build. These two can then be combined. They both make use of a function called AdvanceOneLayer, which is the main part of the building. This function is illustrated in Fig. 9.

AdvanceOneLayer is responsible for advancing the track by one layer. First, the state of the track and the compatible layers are found in the function findStateandLayers, Then, the trajectory segment builder finds the compatible detectors and groups them. This is done in groupedMeasurements. The compatible detectors are found by propagating the state to the next layer and calculating the uncertainty. This uses computationally expensive elements, like calculating the full Jacobian matrix of the state. Details on the calculations can be found in [22]. The detectors are then grouped into mutually exclusive groups. Each part of the silicon strip tracker has a different geometry, so the exact calculations differ. The geometry of the trackers, as shown in Fig. 5, may help explain the relative timings. The endcap is large, and the titled barrel is close to the interaction region, and will therefore likely have a high occupancy. After the detectors have been grouped, the measurements from the detectors are fetched in a loop, labelled

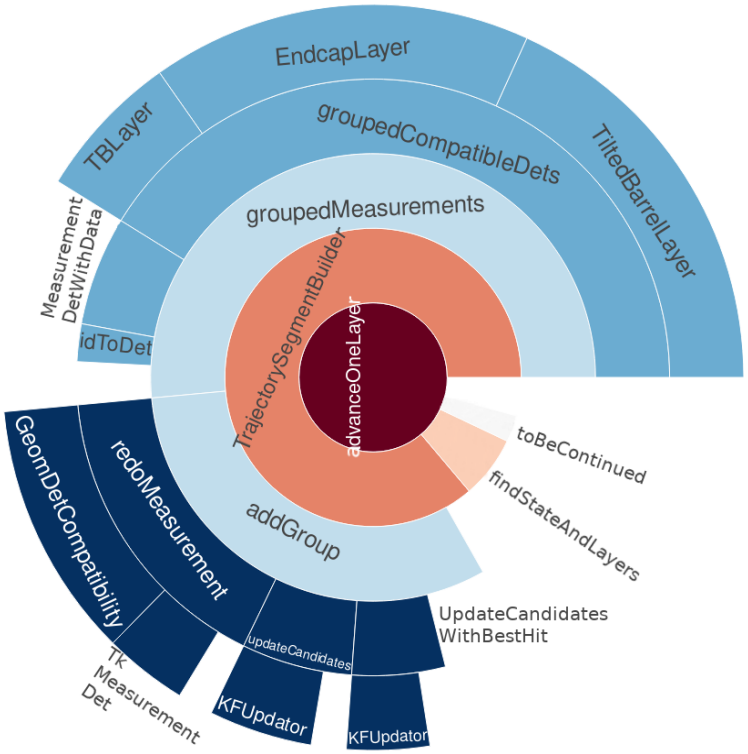
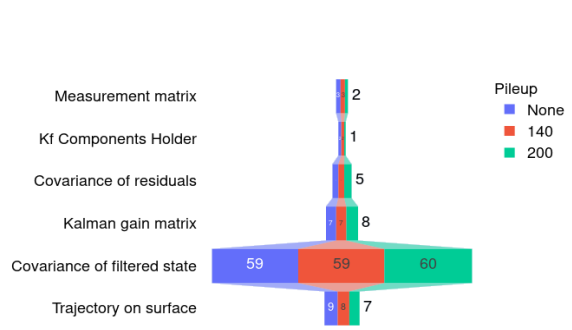
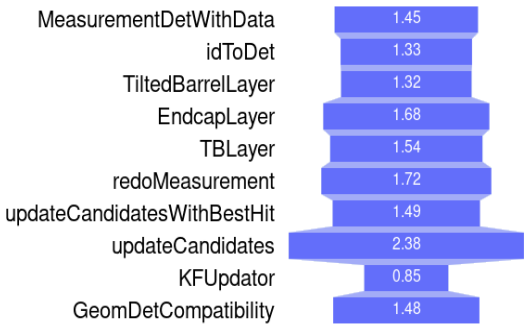


Figure 9: Call sequence for advancing one layer when building trajectories for a pileup of 200. Each slice represents a function, calling the functions that are adjacent, radially outwards from the centre. The size of each slice represents the fraction of the CPU time.



(a) Proportional time by module for the Kalman filter



(b) Ratio of number of times each function was called for a pileup of 200 divided by that of 140

Figure 10: Most functions from the CKFTrackCandidateMaker take up the same proportion of time of their total at higher pileups. However, the number of times each function is called changes.

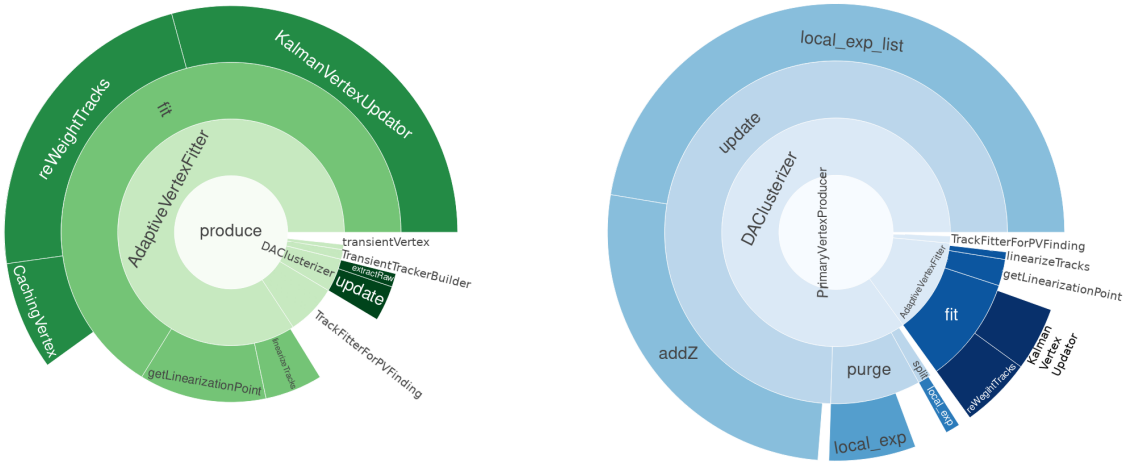
MeasurementDetWithData. One hit is fetched from each compatible group. idToDet contains a look-up table of detector ids and detectors. The function addGroup is then called. If it is not the first outer tracker hit considered, the function redoMeasurement is called. It loops over all the detectors and re-checks the compatibility for the hits in the detectors. The update is then done. A number of maximum hits to consider can be set. Here, it was set to 3. If there are more than three compatible hits, the hit with the lowest χ^2 is chosen. If there are between one and three, a track candidate is made for each possible hit. The update is done using the Kalman filter described in 4.3.1. The function toBeContinued is called to check that the track is good enough to continue building. The updated trajectory is added to the list of temporary trajectories.

Interestingly, making Fig. 9 for a pileup of 0 creates an almost identical chart - the total CPU time is much less, but the proportion of time each section takes up is very close to what is shown here. This is also true for the individual parts of the Kalman filter, as is illustrated in Fig. 10. Also note that the covariance matrix of the updated state is the most time-consuming part of the Kalman filter. If none of these functions take up a larger proportion of the CPU time at a higher pileup, where does the exponential explosion come from? The answer may lie in the loops. There are many more tracks at higher pileup, hence the functions are called many more times. VTune can provide an estimate for how many times it believes a function was called. This was done for the CkfTrackCandidateMaker. The result is shown in Fig. **??**. The number of estimated function calls for a pileup of 200 was divided by that of 140. As can be clearly seen, each function is called many more times for the higher pileup. The only exception is the KFUpdator, which takes less time for the 200 pileup. It is possible that there are more fake tracks that have been rejected early on, resulting in a lower final number of tracks to fit. However, this would need further testing.

The CKFTrackCandidateMaker is a very nested sequence of functions. As is also seen from 9, there are many elements that contribute to the CPU time. Every time a track is to advance one layer, detector geometry is calculated, the track is propagated using computationally expensive matrices, and the Kalman filter is called, where the covariance of the residuals take up the most time. If this function were to be ported to an FPGA, one would have to consider the I/O between all the loops. It is likely that the I/O will be a limiting factor, unless one makes something like mkfit, and put a large part of the HLT on a device. The function that took the longest after the track building was the building of the primary vertices.

7.4 PrimaryVerticesUnsorted

Building the primary vertices online takes a negligible amount of time for no pileup, and a significant amount of time for pileups of 140 and 200, as shown in 7. The performance profile for these scenarios is



(a) Primary vertex fitter for zero pileup

(b) Primary vertex fitter for a pileup of 200

Figure 11: Without any pileup, producing the primary vertices is almost trivial, taking about 1 second for 100 events. For a pileup of 200, the update to the clusterizer is being called many times, and the vertex reconstruction takes about 160 seconds for 100 events.

shown in Fig. 11.

When there is only one set of primary vertices, the deterministic annealing converges very quickly, since the starting point is just one set of vertices. After the clustering, which only uses the z-dimension, the track list is passed on to the AdaptiveVertexFitter. This takes up most of the CPU time for the no pileup scenario. It uses a Kalman filter to fit weights to the tracks according to their distance from the vertices. The tracks are then re-weighted. For higher pileup, the update of the deterministic annealing is being called many times. The update function is called each time the 'temperature' of the system is changed, as explained in 4.5. It updates the position of the vertices and the weights of the tracks. There are two time-consuming elements within the udpate; the $'\mathrm{local_{e}xp_{l}ist}'$ and the $'\mathrm{addZ}'$. The first of these is a function that iterates over a list and returns a list containing the exponential functions of the arguments. In deterministic annealing, the probability of a position of a certain vertex, given an input track x, is given by

$$\mathrm{p\,(y|x)} = \frac{\exp\left(\frac{-\mathrm{d(x,y)}}{\mathrm{T}}\right)}{\mathrm{Z_x}} \quad , \quad \mathrm{Z_x} = \sum_{\mathrm{y}} \exp\left(-\frac{\mathrm{d(x,y)}}{\mathrm{T}}\right) \tag{11}$$

where T is temperature and $\mathrm{d(x,y)}$ is the distance between the track and the vertex [24]. The exponentials shown here need to be calculated for all the tracks in each vertex cluster, repeated each time the temperature changes. The combinatorics of this may be time consuming.

The second part of the update is 'addZ'. This function calculates the position of the clusters. This is given by

$$\mathrm{y} = \frac{1}{\mathrm{N_x}} \sum_{\mathrm{x}} \mathrm{x} \tag{12}$$

, i.e. it puts the vertices at the centre of gravity of the x-distribution. Creating a figure like Fig.11 b) for a pileup of 140 gives almost exactly the same diagram, meaning the proportion of time each module takes of the total stays the same. An exception is the function 'purge', which takes about 30% longer for a pileup of 200 than that of 140. It eliminates vertices that have only one track associated with them.

It is possible this takes longer for the higher pileup since it could converge on a solution with a higher number of vertices. The function 'split' splits the vertices as the temperature drops below the threshold for a phase transition. Even though there are elements of combinatorics in equation 11, one does not see that either $'\text{local}_\text{e}\text{xp}_\text{l}\text{ist}'$ or $'\text{addZ}'$ change the proportion of the total CPU time. Either they both increase in exact proportion to each other, or the update function is being called many more times for a pileup of 200 compared to 140. This needs to be explored. It is likely that there is a similar situation for the third most costly function, CAHitTriplet. A method must be designed for analysing the loop flow of these functions.

8 Future work

The next step is to explore the loops further. Loop diagrams need to be made to clearly see how functions depend on each other and how many times they are called. VTune is again a good tool for this. A thorough exploration of the loops are particularly important since I/O can be a limiting factor on FPGAs. If a function is ported to an FPGA, but relies on frequent input from a CPU, the performance may be compromised. By understanding the loop structure, the memory flow may also be better optimised. It is deemed likely that GPUs or FPGAs will be implemented in the Phase 2 CMS HLT to increase parallelisation. FPGAs will be chosen for further study, in collaboration with Maxeler Technologies. The first step will be to replicate the work of Dr. Sioni Summers, explained in section 6.3. The Kalman filter will be ported to an FPGA. mkfit also showed some very promising results, and has not been explored on FPGAs yet. An mkfit approach on FPGA simulation will therefore be made. This may start with a small part of the code, likely containing one of the functions discussed in the previous section. If successful, the code may be ported to an FPGA.

9 Conclusion

The high level trigger at CMS must be updated to account for an increased number of simultaneous proton-proton collisions. Performance modelling of the CMS HLT showed that there were four high level functions that accounted for 70% of CPU time for a pileup of 140 and 200. The track building took up about 35% of the online runtime for pileups of 140 and 200. The primary vertex finder and cellular automaton hit accounted for the rest. These functions all contain functions that are called in loops. Further exploration of the loop structure should be undertaken. Guided by the loop flow structure the track building, or part of it, would be an interesting part to port to a GPU or FPGA. The collaboration mkfit has developed several ways to parallelise the tracking, and achieved a factor of 6 speedup. Dr. Sioni Summers has also ported the Kalman filter to an FPGA with promising results. An interesting next step would therefore be to either replicate the work of Dr. Summers, or simulate some of mkfit's approach on an FPGA.

A Appendix

The code for the plots in section 7 and the event generation can be found at: https://github.com/livcms/hlt_timing

The code for CMMSW can be found at: https://github.com/cms-sw/cmssw

References

[1] The CMS Collaboration et al. "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08004–S08004. DOI: 10.1088/1748-0221/3/08/s08004. URL: https://doi.org/10.1088%2F1748-0221%2F3%2F08%2Fs08004.

[2] Serguei Chatrchyan et al. "The CMS experiment at the CERN LHC". In: *Journal of instrumentation* 3.8 (Aug. 2008), S08004–S08004. DOI: 10.1088/1748-0221/3/08/s08004. URL: https://doi.org/10.1088%2F1748-0221%2F3%2F08%2Fs08004.

[3] Andrew William Rose. "The Level-1 Trigger of the CMS experiment at the LHC and the Super-LHC". PhD thesis. Imperial Coll., London, 2009. URL: https://pdfs.semanticscholar.org/ad25/af68ce5788a337948899e87f543df5f110c1.pdf.

[4] CMS Collaboration. "The CMS trigger system". In: (Sept. 2016). DOI: 10.1088/1748-0221/12/01/P01020. URL: http://arxiv.org/abs/1609.02366.

[5] Serguei Chatrchyan et al. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021.

[6] K Klein et al. *The Phase-2 Upgrade of the CMS Tracker Technical Design Report*. Tech. rep.

[7] Stefano Mersi. "Phase-2 Upgrade of the CMS Tracker". In: *Nuclear and Particle Physics Proceedings* 273-275 (Apr. 2016), pp. 1034–1041. ISSN: 24056014. DOI: 10.1016/j.nuclphysbps.2015.09.162.

[8] Mia Tosi. *The CMS trigger in Run 2*. Tech. rep. 2017. DOI: 10.22323/1.314.0523.

[9] Burkhard Schmidt. "The High-Luminosity upgrade of the LHC: Physics and Technology Challenges for the Accelerator and the Experiments". In: *Journal of Physics: Conference Series* 706 (Apr. 2016), p. 022002. DOI: 10.1088/1742-6596/706/2/022002. URL: http://stacks.iop.org/1742-6596/706/i=2/a=022002?key=crossref.f9d64a3e749adf2832fca8dba8a224b4.

[10] Erica Brondolin. "Expected Performance of Tracking in CMS at the HL-LHC". In: *EPJ Web of Conferences*. Vol. 150. EDP Sciences. 2017, p. 00001. DOI: 10.1051/epjconf/201715000001.

[11] Thomas Owen James. "A Hardware Track-Trigger for CMS at the High Luminosity LHC". 2018. DOI: 10.25560/60593.

[12] Albert M Sirunyan et al. "Performance of the CMS Level-1 trigger in proton-proton collisions at $\sqrt{s} = 13$ TeV". In: (June 2020). arXiv: 2006.10165 [hep-ex].

[13] Simon Spannagel. *CMS Pixel Detector Upgrade and Top Quark Pole Mass Determination*. Springer, 2017.

[14] Vincenzo Daponte and Andrea Bocci. "CMS-HLT Configuration Management System". In: *Journal of Physics: Conference Series*. Vol. 664. 8. IOP Publishing. 2015, p. 082008. DOI: 10.1088/1742-6596/664/8/082008.

[15] Daniele Trocino. "The CMS High Level Trigger". In: *Journal of Physics: Conference Series*. Vol. 513. 1. IOP Publishing. 2014, p. 012036.

[16] CMS Collaboration. "Description and performance of track and primary-vertex reconstruction with the CMS tracker". In: (May 2014). DOI: 10.1088/1748-0221/9/10/P10009. URL: http://arxiv.org/abs/1405.6569%20http://dx.doi.org/10.1088/1748-0221/9/10/P10009.

[17] Daniel Funke et al. "Parallel track reconstruction in CMS using the cellular automaton approach". In: *J. Phys. Conf. Ser.* Vol. 513. 2014, p. 052010. URL: https://iopscience.iop.org/article/10.1088/1742-6596/513/5/052010/pdf.

[18] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45. URL: https://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf.

[19] M Schiller. "Standalone track reconstruction for the Outer Tracker of the LHCb experiment using a cellular automaton". PhD thesis. University Heidelberg, 2007. URL: https://www.physi.uni-heidelberg.de/Publications/Schiller07.pdf.

[20] CMS Tracker Collaboration et al. "Stand-alone cosmic muon reconstruction before installation of the CMS silicon strip tracker". In: *Journal of Instrumentation* 4.05 (2009), P05004. URL: https://www.researchgate.net/publication/47684426_Stand-alone_cosmic_muon_reconstruction_before_installation_of_the_CMS_silicon_strip_tracker.

[21] Sioni Paris Summers. "Application of FPGAs to triggering in high energy physics". PhD thesis. Imperial Coll., London, 2018. URL: https://cds.cern.ch/record/2647951.

[22] Are Strandlie and W Wittek. *Propagation of Covariance Matrices of Track Parameters in Homogeneous Magnetic Fields in CMS*. Tech. rep. CMS-NOTE-2006-001. Geneva: CERN, Jan. 2006. URL: http://cds.cern.ch/record/927379.

[23] Kenneth Rose, Eitan Gurewitz, and Geoffrey Fox. "A deterministic annealing approach to clustering". In: *Pattern Recognition Letters* 11.9 (Sept. 1990), pp. 589–594. ISSN: 01678655. DOI: 10.1016/0167-8655(90)90010-Y.

[24] E Chabanat and N Estre. *Deterministic Annealing for Vertex Finding at CMS*. Tech. rep.

[25] CMS collaboration. *The Phase-2 Upgrade of the CMS Level-1 Trigger*. Tech. rep. CERN-LHCC-2020-004. CMS-TDR-021. Geneva: CERN, Apr. 2020. URL: http://cds.cern.ch/record/2714892.

[26] Murad Qasaimeh et al. "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels". In: *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE. 2019, pp. 1–8.

[27] Adriano Di Florio, Felice Pantaleo, and Antonio Carta. "IOP: Convolutional Neural Network for Track Seed Filtering at the CMS High-Level Trigger". In: *J. Phys.: Conf. Ser.* Vol. 1085. 2018, p. 042040. DOI: 10.1088/1742-6596/1085/4/042040.

[28] A. Bocci et al. "CMS Patatrack Project". In: (Mar. 2019). DOI: 10.2172/1570206.

[29] Giuseppe Cerati et al. "Reconstruction of Charged Particle Tracks in Realistic Detector Geometry Using a Vectorized and Parallelized Kalman Filter Algorithm". In: *arXiv preprint arXiv:2002.06295* (2020).

[30] Sioni Summers, A Rose, and P Sanders. "Using MaxCompiler for the high level synthesis of trigger algorithms". In: *Journal of Instrumentation* 12.02 (2017), p. C02015.

[31] Linker Compiler and Kernel Compiler. "Maxeler Acceleration Technology". In: (). URL: https://pdfs.semanticscholar.org/72c4/e33c3122026f6c9a3a848dab211a9256713e.pdf.