Evaluation of GPU-based track-triggering for the CMS detector at CERN’s HL-LHC

Hannes Mohr

October 17, 2016
Contents

1 Abstract 1

2 High energy particle physics and Collider Physics 2
   2.1 Collider Physics 2
   2.2 Interactions of particles with matter 4
   2.3 Motion of charged particles in magnetic fields 4

3 The Large Hadron Collider (LHC) 9

4 CMS 10
   4.1 The solenoid 10
   4.2 The silicon tracker 10
   4.3 The Muon Chambers 13
   4.4 The road to the High Luminosity Large Hadron Collider (HL-LHC) 14
   4.5 The upgraded CMS Tracker 18
   4.6 Stubs 18
      4.6.1 Uncertainties 20
      4.6.2 Bend information 20
   4.7 Discovery Potential 22

5 The CMS Trigger System 23
   5.1 The upgraded Trigger System 25
   5.2 Benefits of a Level 1 Track Trigger (L1TT) 25
   5.3 Proposals for L1 Track Triggers at the HL-LHC 28

6 GPUs 30
   6.1 Technological Advances 30
   6.2 Interconnects 33
   6.3 Comparison of Field Programmable Gate Arrays (FPGA) and Graphics Processing Units (GPUs) 34

7 The Software Trigger 37
   7.1 Data Compression 39
   7.2 Detector Layers 40

8 Validation System 41
   8.1 Tracking Particles 41
   8.2 Stubs 42
   8.3 Subsequent Fit 42
   8.4 Used Datasets 43
## 9 Hough Transform

9.0.1 Theory ........................................... 44
9.1 Parametrization ................................. 45
9.2 Localized Coordinates ......................... 45
9.3 Size of parameter space ....................... 46
9.4 Centralized Hough transform ................. 46
9.5 Choosing the amount of bins ................. 47
9.6 Threshold condition ............................ 48
9.7 Filling the Hough map ......................... 48
  9.7.1 Bend Filter ..................................... 48
9.8 Hough parameters .............................. 48
9.9 Resulting Track candidates .................... 48

## 10 Hexagonal Hough Transform

10.1 Motivation ....................................... 52
10.2 Properties ....................................... 53
  10.2.1 Choice of bins ................................. 54
  10.2.2 Allowed line gradients ..................... 54
  10.2.3 Choosing the relative amount of bins ...... 55
  10.2.4 Parameter space area ....................... 55
  10.2.5 Finding the correct bin .................... 55
10.3 Effects on valid parameter space ............. 57

## 11 Comparison of regular and hexagonal grid

11.1 Implementation details and specifics of the GPU architecture ............................. 63
  11.1.1 Workload ........................................ 65
  11.1.2 Conclusion and Outlook ..................... 65

## 12 GPU Benchmarks

12.1 RDMA setup ...................................... 67
12.2 RDMA Benchmark setup ....................... 67
12.3 Transfer Latency ............................... 71
12.4 Combined Latency .............................. 72

## 13 Efficiency and fake rates of the algorithms

14 Efficiency as a function of $p_t$ ................. 76

## 15 Discussion

15.0.1 Preferring high $p_t$ candidates ............. 79
15.1 Required Throughput ............................ 80
List of Figures

1. Stopping power for positively charged muons in copper. The solid lines represent the total stopping power. Vertical bands indicate boundaries between different approximations [21]...

2. Left: Sagita s and R as seen in the figure are convenient ways of parametrizing the helixoidal motion of a charged particle moving under the influence of a magnetic field. Right: Movement of a particle in a homogeneous electromagnetic field [13]...

3. Magnetic Field of the CMS detector, [8]...

4. Overview of the CMS detector...

5. The Electromagnetic calorimeter during its construction, from [30]...

6. Overview of the approved LHC programme up to 2035 [11, p. 6]...

7. Timeline of current and future CERN experiments...

8. The integrated luminosity at the LHC gathered so far since the beginning of RUN I...

9. Schematic view of the pixel strip and strip strip modules to be used in the phase-II tracker. They allow a momentum cut in \( p_t \), drastically reducing the data rates sent to the level 1 trigger.[1, p.4]...

10. Overview of the current Trigger system at Compact Muon Solenoid (CMS) [16]...

11. Vertex consistency requirement for hadronic triggers (\( |\Delta z_0| < 1\text{cm} \))...

12. Muon trigger rates for the barrel as a function of momentum...

13. Expected data rates for the upgraded L1TT system of the CMS detector...

14. The basic concept of RDMA connections. The additional latency penalties of virtual memory are cut out almost entirely by avoiding the CPU...

15. Comparison of key features of the NVIDIA Tesla K40c, and the XILINX VIRTEX-7 XC7VX1140T, both 28 nm manufacturing process...

16. Baseline geometry of the tracker used in this work, 3Pixel Strip (PS)×32 Strip (2S) 5 disks. The more sensitive PS modules provide a better resolution. The design allows a momentum cut and has good enough resolution to aid the triggering decision in the global trigger. The layout was designed by the collaboration, using the tklayout tool ([6], [5])...
Visualization of the concept of the Hough transformation. Each entry in the histogram corresponds to a possible curvature of the track.

Comparison of the regular and hexagonal parameter spaces.

left: Number of cells for the regular transformation (equals the area in parameter space). middle: number of cells, hexagon.
right: Area in parameter space hexagons.

Comparison of the regular and hexagonal binning behavior.

left: hexagonal transformation, 29 bins in \( \phi_0 \). Offers tighter binning and by means of the bigger slope decreases width of clusters in the parameter space. The cluster height increases. The negative impact of this is mitigated by the gap between hexagons. This is the momentum cut version, the binned representation of the line is without gaps.
right: regular transformation. Uses the standard rectangular grid. Is limited to slopes smaller than one, so as not to produce too many bins in \( \phi_0 \). The width of the clusters is bigger, not just in the index space, but also in the parameter space.

Implemented RDMA data transfer scheme, optimized for minimum latency, kernel idle while polling.

Implemented RDMA data transfer scheme, optimized for maximum throughput, has a higher latency.


Theoretical minimum latency for data transfer, given saturation of the bus.

Comparison of the quality of the seed for the production angle \( \phi_0 \). Top Left: [HT] Top Right: HHT\(_{23}\) Bottom Left: HHT conservative cut. Bottom Right: HHT\(_{aggressive}\) cut.

Comparison of the quality of the seed in \( \frac{4}{\mu} \). Top Left: [HT] Top Right: HHT\(_{23}\) Bottom Left: HHT conservative cut. Bottom Right: HHT\(_{aggressive}\) cut.
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Examples of Luminosities for different colliders [14]</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Bandwidths for different versions of the PCIe standard with respect to the</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>number of used lanes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Comparison of the properties of High Bandwidth Memory and GDDR5. HBM should</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>reduce memory access latency drastically [17]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Overview of the used compression scheme for the transmitted data words. The</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>detector resolution is given, as well as the resulting resolution for each</td>
<td></td>
</tr>
<tr>
<td></td>
<td>variable. The method is adapted from the one used by [4], but slightly</td>
<td></td>
</tr>
<tr>
<td></td>
<td>changed to suit the GPU.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Same as above, second data word. This package also includes information</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>needed by the fit, that is not yet implemented.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Includes information of the allowed bin range for the Hough</td>
<td></td>
</tr>
<tr>
<td></td>
<td>transformation. Also includes the layer number, using the numbering scheme</td>
<td></td>
</tr>
<tr>
<td></td>
<td>discussed in Section [9] for use in the minimum layer filtering condition.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Comparison of tracking efficiencies, total number of found track candidates</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>and reconstructed tracks without additional stubs, that do not come from a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tracking particle. TTbar PU140 Dataset. 3525 Events. Total tracks valid for</td>
<td></td>
</tr>
<tr>
<td></td>
<td>algorithmic tracking: 64593</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Same as above, but for a higher PU TTbar PU200 Dataset. 2400 Events. Total</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>tracks valid for algorithmic tracking: 44526</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Comparison of the hardware between two subsequent cards. The Tesla K40 used</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>in this work and its successor, the Tesla K80. The later has two GPU cores.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Processor names are given in the table. Data taken from [19]</td>
<td></td>
</tr>
</tbody>
</table>
1 Abstract

In this work we present an evaluation of GPUs as a possible L1 Track Trigger for the High Luminosity LHC, effective after Long Shutdown 3 around 2025.

The novelty lies in presenting an implementation based on calculations done entirely in software, in contrast to currently discussed solutions relying on specialized hardware, such as FPGAs and Application-specific integrated circuits (ASICs). Our solution relies on using GPUs for the calculation instead, offering floating point calculations as well as flexibility and adaptability and adaptability. Normally the involved data transfer latencies make GPUs unfeasible for use in low latency environments. To this end we use a data transfer scheme based on Remote Direct Memory Access (RDMA) technology. This mitigates the normally involved overheads. We based our efforts on previous work by the collaboration of the Karlsruher Institut für Technologie (KIT) and the English track trigger group [4] whose algorithm was implemented in FPGAs. In addition to the Hough transformation used regularly, we present our own version of the algorithm based on a hexagonal layout of the binned parameter space. With comparable computational latency and workload, the approach produces significantly less fake track candidates than the traditionally used method. This comes at a cost of efficiency of around 1 percent. This work focuses on the track finding part of the proposed L1TT and only looks at the result of a least squares fit to make an estimate of the performance of said seeding step. We furthermore present our results in terms of overall latency of this novel approach.

While not yet competitive, our implementation has surpassed initial expectations and are on the same order of magnitude as the FPGA approach in terms of latencies. Some caveats apply at the moment. Ultimately, more recent technology, not yet available to us in the current discussion will have to be tested and benchmarked to come to a more complete assessment of the feasibility of GPUs as a means of track triggering at the HL-LHC's CMS experiment.
2 High energy particle physics and Collider Physics

High Energy Physics (HEP) is concerned with studying the properties of subatomic particles and their interactions. The Standard Model (SM) of particle physics is the foundation of all modern HEP experiments. It describes the electromagnetic, strong and weak interactions and classifies all known subatomic particles. While the SM in its current form was mostly finalized in the 1970’s, building sufficiently large and powerful particle colliders to prove its predictions is an ongoing task up to this day. Discoveries ranged from the top quark to the $\tau$ neutrino and finally to the Higgs boson. This last missing piece predicted by theory was finally experimentally confirmed by the Large Hadron Collider (LHC) in 2012.

However, the SM provides neither an explanation for gravity nor for dark matter. While theories explaining these phenomena exist, none has yet been proven. To keep pushing the boundaries of our knowledge more sensitive accelerators are still necessary. Probing the small scales that make up matter requires high energies. To acquire statistically significant measurements of the rare processes that need to be studied, particle accelerators must provide high interaction rates.

2.1 Collider Physics

The intrinsic rate of an event is given by the differential cross section

$$d\sigma(\theta) = \frac{N(\theta)d\omega}{N_0},$$

where $\theta$ is the scattering angle, $N_0$ is the number of beam particles per square meter and second and $N(\theta)d\omega$ is the number of particles passing through the solid angle $d\omega$ per second. Integrating over all possible angles, the differential elastic cross section becomes the cross section

$$\sigma_{el} = \frac{1}{N_0} \oint_S N(\theta)d\omega.$$  

Collision can either be elastic or inelastic. Each possible outcome of an inelastic collision is called a channel. For the channel $i$, the differential cross section is given by

$$d\sigma_i(\theta) = \frac{N_i(\theta)d\omega}{N_0}.$$  

The total cross section is the sum of the elastic and the inelastic cross sections

$$d\sigma_{\text{tot}}(\theta) = \frac{N(\theta)d\omega}{N_0}.$$
Table 1: Examples of Luminosities for different colliders

<table>
<thead>
<tr>
<th>Collider</th>
<th>Interaction</th>
<th>Luminosity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPS</td>
<td>p + ¯p</td>
<td>$6 \times 10^{30}$</td>
</tr>
<tr>
<td>Tevatron</td>
<td>p + ¯p</td>
<td>$4.0 \times 10^{32}$</td>
</tr>
<tr>
<td>HERA</td>
<td>p + e⁺</td>
<td>$4.0 \times 10^{31}$</td>
</tr>
<tr>
<td>LHC</td>
<td>p + p</td>
<td>$1.0 \times 10^{34}$</td>
</tr>
<tr>
<td>LEP</td>
<td>e⁻ + e⁺</td>
<td>$1.0 \times 10^{32}$</td>
</tr>
<tr>
<td>PEP</td>
<td>e⁻ + e⁺</td>
<td>$3 \times 10^{33}$</td>
</tr>
<tr>
<td>KEKB</td>
<td>e⁻ + e⁺</td>
<td>$2.1 \times 10^{34}$</td>
</tr>
</tbody>
</table>

The rate $R$ of interactions in a collider experiment is given by

$$R = f \mathcal{L}$$

where $f$ is the collider frequency and $\mathcal{L}$ is the beam luminosity. It is given by

$$\mathcal{L} = \frac{N_1 N_2 f N_b}{4 \pi \sigma_x \sigma_y}.$$  

(6)

Shortly before the Interaction Point (IP), magnets push the beams towards each other to make them collide. The beams collide at an angle called crossing angle, which is about $\approx 300\mu$rad at the LHC. This leads to the introduction of the luminosity reduction factor, $S$. We thus get a corrected expression for the luminosity

$$\mathcal{L} = \frac{N_1 N_2 f N_b}{4 \pi \sigma_x \sigma_y} S.$$  

(7)

This result can be expressed as the corrected beam size, given by

$$\sigma_{\text{eff}} = \sigma \cdot \sqrt{1 + \frac{\sigma_s \phi}{\sigma_x^2}}.$$  

(8)

Another important characteristic of a particle accelerator is its integrated luminosity. Whereas the luminosity is a measure of the possible interactions that can occur within a given amount of time, the integrated luminosity measures the accumulated number of potential collisions.

$$\mathcal{L}_{\text{int}} = \int \mathcal{L} dt$$  

(9)
2.2 Interactions of particles with matter

Charged particles traveling through a material undergo interactions. These interactions can lead to ionization or excitation of the atoms in the material. The mean energy loss of the particle per distance traveled is described by the Bethe formula. This holds for protons, but not electrons. For a particle with speed $v$, charge $z$ (in multiples of the electron charge), and energy $E$, traveling a distance $x$ into a target of electron number density $n$ and mean excitation potential $I$, the relativistic version of the formula reads, in SI units:

$$\langle \frac{dE}{dx} \rangle = \frac{4\pi}{m_e c^2} \cdot \frac{n z^2}{\beta^2} \cdot \left(\frac{e^2}{4\pi\epsilon_0}\right)^2 \cdot \left[ \ln \left( \frac{2m_e c^2 \beta^2}{I \cdot (1 - \beta^2)} \right) - \beta^2 \right],$$

where $c$ is the speed of light, $\epsilon_0$ is the vacuum permittivity, $\beta = \frac{v}{c}$, and $e$ and $m_e$ are the electron charge and rest mass, respectively. The electron density of the material can be calculated as

$$\frac{N_A \cdot Z \cdot \rho}{A \cdot M_u},$$

where $\rho$ is the density of the material, $Z$ its atomic number, $A$ its relative atomic mass, $N_A$ the Avogadro number and $M_u$ the Molar mass constant.

Slower moving particles lose energy more quickly as they have more time to interact with the electrons of the material. The valley in figure 1 is at approximately the same energy for different particles. It is called the Minimum Ionizing Particle (MIP) point. The subsequent rise in energy loss after this point is due to relativistic effects. The spatial range $R$ of a particle inside a given material with respect to its energy, $E$, is $dE/dX$. In terms of the energy $E$:

$$R(E) = R(E_{\text{min}}) + \int_{E_{\text{min}}}^{E} \left( \frac{dE}{dx} \right)^{-1} dE.$$

2.3 Motion of charged particles in magnetic fields

The motion of a charged particle in a magnetic field is determined by the Lorentz force. The force only acts on the velocity component perpendicular to the magnetic field, $B$.

$$m_0 \gamma \frac{dv}{dt} = e \vec{v} \times \vec{B}$$

$$m_0 \gamma \frac{d^2 \vec{r}}{dt^2} = e \frac{d\vec{r}}{dt} \times \vec{B}.$$
Figure 1: Stopping power for positively charged muons in copper. The solid lines represent the total stopping power. Vertical bands indicate boundaries between different approximations [21].
describes the force acting on a relativistic particle of rest mass $m_0$, velocity $v$, and charge $e$, where $\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$. Interchanging the time $t$ with the path length

$$ds = vdt$$

we get

$$m_0 \gamma v \frac{d^2 \vec{r}}{ds^2} = e \frac{d\vec{r}}{ds} \times \vec{B}. \quad (15)$$

This can be rewritten as

$$\frac{d^2 \vec{r}}{ds^2} = \frac{e}{p} \frac{d\vec{r}}{ds} \times \vec{B}. \quad (16)$$

In the case where $B$ is homogeneous and does not vary along the trajectory of the particle, the trajectory is given by a helix. The sagita, $s$, is a measure of the track bend. For an inhomogeneous field, the solution of the equation becomes more complicated. The momentum of the particle can be conveniently decomposed into the two projections $p_\perp$ and $p_\parallel$. They are connected through the dip angle, $\lambda$, via

$$p_\perp = p\cos(\lambda)$$

$$= 0.3BR. \quad (17)$$

The sagita $s$ of the track is given by

$$s = 1 - \cos \alpha \quad (18)$$

and can be approximated as

$$s \approx \frac{R^2}{2} = \frac{L^2}{8R}. \quad (19)$$

The helix can be described in parametric form as:

$$x(s) = x_0 + R(\cos \phi_0 + \frac{hs \cos \lambda}{R} - \cos \phi_0)$$

$$y(s) = y_0 + R(\sin \phi_0 + \frac{hs \cos \lambda}{R} - \sin \phi_0)$$

$$z(s) = z_0 + s \sin \lambda \quad (20)$$

where $\lambda$ is again the dip angle and $h = \pm 1$ is the direction of rotation depending on the particles charge. The projection of the particle trajectory onto the $r-\phi$-plane is a circle, with the equation

$$(x - x_0 + R \cos \phi_0)^2 + (y - y_0 + R \sin \phi_0)^2 = R^2, \quad (21)$$
Figure 2: Left: Sagita s and R as seen in the figure are convenient ways of parametrizing the helixoidal motion of a charged particle moving under the influence of a magnetic field. Right: Movement of a particle in a homogeneous electromagnetic field [13].
where $x_0$ and $y_0$ are the initial position of the track and $\phi_0$ is the production angle. Assuming the magnetic field points in the $z$ direction, the helix can be described with the 5 parameters:

$q/p_t$ the charge of the particle, divided by the transverse momentum

$\phi_0$ the angle at the point of closest approach in the $xy$-plane

$d_0$ the transverse impact parameter, distance at the point of closest approach in the transverse plane

$\cot(\theta)$ the cotangent of the angle between momentum vector and $z$-axis, at the point $d_0$

$z_0$ the $z$-coordinate at $d_0$

These are the 5 parameters that need to be obtained when fitting a particle trajectory. In the case where $d_0 = 0$, we can give the relation of the distance, $R$, from the origin and the angle, $\phi$, of the trajectory as

$$\phi = \phi_0 - \arcsin\left(\frac{R}{2\rho}\right) \approx \phi_0 - \frac{R}{2\rho}.$$  (22)
3 The Large Hadron Collider (LHC)

The LHC at European Organization for Nuclear Research (CERN) in Geneva is the world’s largest and most powerful particle accelerator. Built between 1998 and 2008, it is situated in an tunnel 27 kilometres in circumference. Starting with an initial energy of $3.5\,\text{TeV}$ per beam, it has increased its energy to the current world record of $6.5\,\text{TeV}$ per beam.

Its aims include precision measurements of known physics, but also the probing of uncharted territories, such as the search for supersymmetry, as well as investigating the properties of dark matter and the origins of our universe. The LHC houses a total of seven experiments, one of which is the CMS detector, which we will discuss in some detail in the next chapter.

The particles are pre-accelerated before entering the colliders main ring. Inside the main ring beam pipes are kept at ultra-high-vacuum. Bunches of particles (mostly protons but also lead atoms) travel in opposite directions along the pipes close to the speed of light. The particle beams are guided by an electromagnetic field, produced by superconducting magnets. They are also in a very high vacuum and at a constant temperature of around $1.9\,\text{K}$ to shield them from thermal influences of the environment. This is necessary to keep the magnets operating at superconductivity, without which the machine would have to be drastically larger in size to be able to reach the same collision energies. Specialized lattice magnets, ensure the stability and precision of the particle beam. A total of 1232 dipole magnets keep the particles in their circular trajectory around the main ring. 392 additional quadrupole magnets are used to bunch the particles together. They are crucial to control the precise collisions of the bunches at the different IPs of the experiments.
4 CMS

The Compact Muon Solenoid (CMS) experiment is one of four major experiments at the LHC. Just like its counterpart, the ATLAS experiment, it is a general purpose, cylindrical symmetry particle detector. It consists of a high performance magnet, an electromagnetic calorimeter, a hadronic calorimeter as well as muon chambers. We will give a brief description of its parts in the following.

4.1 The solenoid

The heart of the CMS detector is its powerful solenoid. It has a very strong, superconducting magnet, that produces very homogeneous magnetic field of around 4 T inside the tracker. Outside of the tracker the field is less homogeneous as can be seen in figure 3.

4.2 The silicon tracker

The silicon tracker consists of six barrel and five endcap detector layers, surrounding the IP. It is the innermost part of the detector, closest to the interaction point. Each layer provides a high precision measurement of the position of a passing particle. Using the combined information of the hits of all layers, we can infer the particle momentum. The bending radius $R$ of a
particle depends on the magnetic field $B$ and its momentum $p_t$ as

$$R[m] = \frac{p_t}{0.3B[T]}.$$\hspace{20pt}(23)$$

Higher momentum particles, will have a bigger bending radius, hence they are closer to a straight line and their bend is smaller. In consequence, a strong magnetic field is needed to provide a good resolution of high momentum, low curvature tracks.

The magnetic field is very homogeneous with a field strength of around 4T.

Figure 4: Overview of the CMS detector.

**The Electromagnetic Calorimeter**

The Electromagnetic Calorimeter (ECAL) is the second major layer of the CMS detector. It provides a precise measurement of the energy of electrons and photons. The material used in the ECAL is lead tungstate. It emits photons by means of *scintillation*, the amount of light is proportional to the particle energy. The scintillation radiation is captured by dedicated photon...
detectors. The ECAL is made up of a barrel and endcap and has a total of about 80,000 scintillation crystals.

The Hadronic Calorimeter

The Hadronic Calorimeter (HCAL) measures the energy of hadronic particles, such as protons, neutrons, pions and kaons. It is designed to be hermetic, meaning the particles should deposit all their energy inside of the detection volume. If this goal is achieved, it becomes possible to detect otherwise undetectable particles, such as neutrinos, by means of the missing energy within the total energy distribution. The layers are staggered, leaving no room for particles to pass through undetected. The HCAL is a sampling calorimeter.

Hybrid Photodiodes (HPDs)

The signals from the detection chambers of the HCAL are transported to the HPDs fibre-optic waveguides. The surface of the HPDs is sensitive to light and converts it into electrons by means of the photoelectric effect. The electrons are accelerated towards the silicon diodes. They can detect
up to 19 different signals at once in this fashion and then send the signal to the trigger and Data Acquisition (DAQ) systems.

4.3 The Muon Chambers

The muon chambers are situated at the outskirts of the detector, because of the muons property to pass the rest of the detector without significant interactions. The basic detector principle in the CMS muon system is gas ionization. The chambers are placed between the iron return yoke plates of the detector. The return yoke has the additional benefit of stopping other particles that managed to pass the calorimeters undetected. The chambers, just like the tracker, take measurements at different radii to measure the muon’s momentum.
4.4 The road to the HL-LHC

Upgrading the machine was in planning even before it went online. Phases of ongoing measurements, referred to as runs, are alternated with Long Shutdowns (LSs), during which maintenance and upgrade work takes place. Figure 6 shows the current plans for the upgrades the machine will undergo.

We give a brief overview of the preparations and plans for phase 2 of the detector. We largely follow the discussion given in the technical design report. [11] p. 6-12] RUN I was the first data taking period between in 2011 to 2012. During LS1, the machine was upgraded to run at a beam energy of 13TeV. Starting with RUN II, the bunch spacing time was reduced from 50ns down to 25ns. In this current working mode, the CMS detector experiences about 25 inelastic collisions per event, they are referred to as event Pile Up (PU). These additional collisions are essentially background noise and need to be filtered out by the trigger system. Planned improvements to the injector chain will lead to brighter bunches and a higher luminosity of up to $2 \times 10^{34}$ cm$^{-2}$s$^{-1}$. This will create an integrated luminosity of around 300 $fb^{-1}$ by 2023. By then, the beam-focusing quadrupoles as well as the collision regions, are expected to be at the end of their life cycles due to radiation damage. With the end of LS3, the high luminosity period, referred to as HL-LHC or Phase-II, will start. The current proposal is planning for a 10 year operation at 250$fb^{-1}$ of integrated luminosity per year. The event PU will rise dramatically up to 140–200, leading to a major challenge for
The radiation and the degradation effects associated with it will increase dramatically. With a collision rate of $5 \times 10^9 s^{-1}$, the annual dose delivered to the detector at the HL-LHC in just one year will be of the same order of magnitude as the complete dose taken from the beginning of operations up to the beginning of LS 3 [11].

The bunch crossings, will still happen at a rate of 40MHz with collisions between bunches occurring at 25ns separation. The number of bunches can not be increased during the update, consequently the amount of PU will go up. Most of the collisions happening at the HL-LHC are soft collisions, those collisions do not play a role in the discovery and measurement of the HEP phenomena of interest at CMS. The amount of high $p_t$ particles is relatively small in comparison. They will however have a negative impact on the triggering system used, as well as possibly, if not dealt with online in the correct manner, have a negative impact on the offline construction of events.

We distinguish three different kinds of PU depending in the time of the energy deposition in the detector. In Time (IT) PU stems from some soft pp-collision within the current bunch crossing. It makes up the largest fraction. The other two types are Out Of Time (OOT) PU. Those are either left over...
Figure 8: The integrated luminosity at the LHC gathered so far since the beginning of RUN I.
from a previous bunch crossing, or actually comes already from the next bunch crossing, which may happen if the later parts of the current bunch crossing are still read-out. Because the current readout times of the tracker are relatively short, compared to the 25 ns bunch crossings, OOT PU does not affect the tracker as much. The calorimeters on the other hand are less robust because of the relatively large amount of time it costs to read out shower data. The high PU leads to higher data rates, possible shadowing of isolated events of interest, as well as being computationally more demanding for both offline and online tasks.
4.5 The upgraded CMS Tracker

Silicon detectors in the tracker come in two flavors, silicon $2S$ detectors and silicon $PS$ pixel detectors. Additionally, pure pixel detectors are used in the innermost part of the tracker, where spatial resolution is most critical to achieve sufficient vertex discrimination in the case of short-lived particles. The $2S$ modules are used in the outer regions of the barrel and endcap. They are simpler and have $2 \times 1024$ channels each, cover an area of $5 \times 10cm^2$ and have a strip pitch (the distance between two strips) of $90\mu m$ [7, p.2]. The use of the higher resolution $PS$ modules significantly increases the performance of the L1 reconstruction [1, p.8] They allow a precise measurement of the $z$ coordinate, in order to reconstruct the longitudinal impact parameter $z_0$ [7, p.3]. They have a pitch of $99\mu m$ [7, p.3]. The charge response of the silicon strip detectors is below $10\, ns$. This is fast enough to resolve the $25\, ns$ bunch crossings, but not fast enough to completely avoid dead time. The two module types provide a $30\%$ improvement in momentum resolution compared to the current CMS detector [1, p. 9].

4.6 Stubs

In order to deal with the large amount of data that will be produced by the silicon tracker after the proposed upgrade, low $p_t$ tracks must be filtered out on the detector itself. A selective readout on the front-end chip will be
(a) Two strips of a silicon detector are shown. Particle trajectories are indicated by a dashed line. The left trajectory shows the condition leading to a valid stub. The right is an example of detector hits being filtered out. The valid stub is passed on to the level 1 trigger.

necessary to reduce the data flow [7, p.2].

To achieve this goal, the concept of stubs is introduced. Stubs are pairs of hits in a stack. A stack has upper and lower sensors, with a spacer in the $r$ direction and some periodic pattern in $\phi$. A Read-Out Chip (ROC) is used to process this data in parallel and then solve the combinatoric problem of finding the stubs. If the two hits are not further away than 3 strips a stub is formed. This procedure corresponds to a filtering in $p_t$, as only stubs are passed on. Specifically, this means that the two clusters that make up the stub must satisfy [25, p.2]

$$|\phi_2 - \phi_1| < |r_2 - r_1| \frac{B}{2 \cdot 10^7 p_t,\text{thresh}}. \quad (24)$$

The momentum cut that gets applied, reduces the data by one order of magnitude or more [1, p.3]. It, furthermore, serves as a means of estimating the track bend. However, the bend resolution is not high enough to extrapolate from one stub to another stub in a different layer. Instead, it can be used to narrow down the number of possible stubs belonging to the same track in a certain region of the detector. One approach that uses this is the tracklet approach. Here, pairs of stubs are sorted into tracklets and are then further used to build roads belonging to track candidates.
4.6.1 Uncertainties

The measurement uncertainties of the detector cause uncertainties in the stub coordinates. These vary depending on the type of module the stub is coming from.

In the case of the $2S$ module, an uncertainty of half a strip length is introduced. In the case of a stub coming from the barrel, this affects the $r$-coordinate resolution of the stub, whereas in the case of a stub coming from one of the disks, it results in an error in the $z$-coordinate.

4.6.2 Bend information

The stub bend can be estimated from the information provided by the detector modules. The bend of a stub is given by the module’s strip pitch. The sensor spacing $d_{\text{sensor}}$ is given by the distance in $r$ as

$$\Delta r = r_1 - r_0$$

in case of a stub coming from the barrel, or by the distance in $z$ as

$$\Delta z = z_1 - z_0$$

in case of a stub in the disk of the tracker. The bend is given by the displacement of the two hits, or clusters, in units of the detectors strip pitch. This allows us to restrict the values of $\frac{\Delta}{p_t}$ in the Hough transform. The allowed
bins are given by

\[ \frac{q}{p_{t \text{ min, max}}} = \frac{d\phi \pm \Delta d\phi}{r \cdot B \cdot c} \cdot 10^{11}\text{[GeV}^{-1}], \]  

(25)

where \( B \) is the magnetic field in Tesla, \( c \) the speed of light and where \( d\phi \) is the stub bend. \( \Delta d\phi \), the resolution of \( d\phi \), is given by the uncertainty in the strip pitch.
4.7 Discovery Potential

The LHC will look for phenomena beyond the standard model, such as Super Symmetric (SUSY) models, which may still lead to signals of new particles in the TeV range. This could for example also mean heavier cousins of the Higgs boson. The newly discovered penta-quarks are also of great interest. Aside from the search for new physics, a big part of the program for the [HL-LHC] will consist of more precise measurements of known phenomena. We are interested in further measurements of the W-mass, WW, WWZ, WWγ couplings and the Higgs self-coupling. As of now, the Higgs boson couplings have been measured with a precision of about 20%. The upgraded [CMS] detector should be capable of improving upon those measurements down to a few percent precision during phase 2. The coupling of the Higgs to second-generation fermions will be probed for the first time, via the process

\[ H \to \mu\mu. \]  \hspace{1cm} (26)

Higgs boson self-coupling will be studied in the di-Higgs process

\[ H + H \to \mu\mu. \]  \hspace{1cm} (27)

Prior to the upgrade the small cross section of only 40fb made this process undetectable. The coupling to charged leptons is of great interest as well. The coupling to τs will be well-measured by the end of phase II. Muon coupling, with a branching ratio of only \( \approx 10^{-4} \) might at last become accessible. Vector boson scattering will be used to measure the role of the Higgs in electro-weak symmetry breaking.
5 The CMS Trigger System

Most of the data coming from the detector does not stem from interactions that are of interest in the search for new physics. It would be outright impossible to store all of the data coming from the machine, if we did not discard some of it. The task of the trigger system is to reduce the data rates coming from the detector down to a manageable amount that can then be stored for later analysis. There is a strict timing constraint, based on the length of the pipeline of the online system. The local information is complemented by a global decision, that combines the information of the local triggers. This second stage starts searches more specifically for concrete physical phenomena of interest. All of this happens online and not at full detector resolution.

Only if the criteria are met the complete data is read out and sent to a server farm, called the High-Level Trigger. It performs a more in-depth offline triggering decision based on the full detector information. Some parts are not used at all. The current CMS trigger is based on a two-stage triggering process. We will go into some more detail concerning the current online triggering decision in the following. The first stage is the Level 1 (L1) Trigger. It is implemented entirely in hardware, using FPGAs and custom ASICs. It receives data at a rate of 40 MHz, corresponding to the LHC’s bunch crossing time of 25 ns, and reduces this rate down to 100 kHz. If this first stage of the trigger accepts the current event (L1 Accept (L1A)), the complete data from the detector is passed on to the second stage, the High Level Trigger (HLT). The HLT consists a large farm of computers that perform a complete reconstruction of the detector data and then writes the data out for permanent storage. When deciding whether to accept the current event, the Level 1 Trigger (L1T) uses information from different parts in different layers of the detector. It does not use all information at this stage. The used parts in the decision include the three muon detector systems, the Drift Tubes (DTs), Cathode Strip Chambers (CSCs) and Resistive Plate Chambers (RPCs), the calorimeter data from the HCAL and ECAL, as well as a resolution-reduced portion of the Forward Hadronic Calorimeter (HF).

In the current operation mode of the CMS this decision has to be carried out within a time window of 3.4 µs, called the triggering latency. Data from the silicon tracker is completely disregarded at this stage and is only sent to the HLT in case of a positive triggering decision. The triggering decision is based on numerous factors, such as the momentum of observed tracks, energy deposited in the calorimeters, as well as meta information that combines all of this and looks for hints of certain events, such as two back to back photons with a certain energy for example, hinting at processes of interest for which
the data should be stored.

Most of the hits in the detector come from interactions, where particles only deflected, without causing any meaningful interaction.

Such processes are called minimum bias events. They are predominantly present in the forward directions of the detector and can be characterized by their relatively low transverse momentum.

90 percent of charged particles from minimum bias collisions have a transverse momentum smaller than 1 GeV, 97 lower than 2 GeV.
5.1 The upgraded Trigger System

As a result of the increased requirements of the HL-LHC the trigger system of CMS will have to undergo a major update in the trigger system, to be able to fully use the higher rates of physics processes.

To this end, the pipeline latency of the system will be increased from $3.4\,\mu s$ to $12.5\,\mu s$. The time available to the L1TT will be around $4\mu s$. It is determined by the length of the silicon tracker readout pipeline, e.g. the size of their buffers.\cite{16, p. 5} After this time-window the global trigger will make a decision of whether or not to keep the current event, based on the combined information of all the trigger systems of the detector. For the first time, information from the silicon tracker, priorily only used offline, will be used in the online triggering decision. A possible use of the data from the pixel data is also under consideration. \cite{11, p. 191}.

5.2 Benefits of a L1TT

To aid the triggering decision, the track trigger will send track candidates to the global trigger. The track candidates are made up of stubs belonging to the fitted track, as well as the track parameters obtained from the fitting step. All of this will have to be performed online, within the given timing restrictions.

This information will greatly benefit the suppression of background signals from the actual physics events of interest. Some of the benefits from this approach include the discrimination of electrons from hadronic background jets $\mu^0 \rightarrow \gamma\gamma$ suppression of fake high-$p_t$ tracks coming from the muon trigger as well as the resolution of the $z$-coordinate, to a precision of $\approx 1\,\text{mm}$ \cite{11, p. 191}.

We will briefly present some examples of the benefits of using additional Tracker information in the trigger decision, following the discussion by \cite{27}. Figure \ref{fig:acceptance} shows the flattening of the acceptance rate of the muon trigger during large pile up conditions. The false tagging of low $p_t$ tracks as high $p_t$ tracks produces high backgrounds to the accepted signal. It is not just the increased luminosity, but also the effects of the added pile up the algorithms performance, such irreducible events, get aggravated by accidental coincidences \cite{1, p. 2} The aid of using momentum information from the Tracker mitigates that problem.

Furthermore the Tracker information can aid in discriminating jets found in the hadronic triggers, by means of the $z_0$ value of the fitted production vertex.

Defining hadronic trigger conditions, jets will be required to originate
from a shared $z_0$ coordinate to within $5 - 10$ mm\cite{11, p. 191].

Furthermore electron triggering will be improved, by comparing the tracker stubs to isolated L1 calorimeter objects\cite{25, p.3}:

This information is then used to find a second stub.

**Design goal**

The design goal of the new trigger system is to maintain the current online acceptance rate of about 1%. The rate of the recorded data will thus increase to 5 kHz for PUs of 140 and 7.5 kHz, \cite{11, p. 191} for PUs of 200. Comparing this to the current rates of the order of $10^2$ Hz, the involved challenges, and thus the need for new advances in both technology and algorithms becomes apparent. The algorithms should be robust against pile-up and capable of dealing with the immense data rates.

An overview of the input data rates from the front-end into the detector back-end where L1 tracking is performed, are on the order of 50 TB/s, as illustrated in figure\cite{15}.

Figure 13: Vertex consistency requirement for hadronic triggers ($|\Delta z_0| < 1\text{ cm}$)

Figure 14: Muon trigger rates for the barrel as a function of momentum
Figure 15: Expected data rates for the upgraded L1TT system of the CMS detector
5.3 Proposals for L1 Track Triggers at the HL-LHC

We give a brief overview of some of the proposed solutions, as currently under consideration by the CMS collaboration \[11\]. The proposed track trigger will have to provide a 5 parameter track fit together with the information of the validity of the fit in terms of the fit’s $\chi^2$ and the stubs making up the track candidate to the Global Trigger (GT). The algorithms will have to be able to operate at efficiently at any pile-up. \[22\].

Multiple possible solutions are currently being considered to achieve this goal. All of them rely on the use of specialized hardware. In the following we provide a brief overview of the proposed solutions for the L1 Track Finding and Fitting stages, based on the discussion \[11\, p.47–48\]

**AM Based Track Finding**

Associative Memory (AM) Based-L1 Tracking, relies on specialized hardware, it uses ASICs to feed the incoming stubs into a pattern bank that was pre-calculated based on Monte Carlo (MC) simulations, see \[24\].

In the simulation data, both the real tracks as well as the associated produced hits are known. It is thus possible to produce hit pattern banks, that associate every possible hit pattern to a corresponding track. The AM approach uses massively parallel operations to tackle the combinatorics of the problem. The proposed partitioning of the detector is 6 in $\eta$ and 8 in $\phi$.

The big advantage lies in the scaling with the PU of the detector. The execution time depends linearly on the occupancy of the detector. This makes it a good choice for high PU scenarios. It is a well established method, already used in numerous HEP experiments. A similar system is already in use at the ATLAS detector. For a silicon based detector the size of CMS the required amount of information to be transferred combined with the strict timing limitations make the design and implementation of such a system very challenging.

**Tracklet approach, FPGA-based Road Finding**

The seeding step of this approach relies on finding pairs of stubs in neighboring layers, that are in accordance with a possible track candidate of interest. \[28\] These pairs are called tracklets. Combining two stubs to a tracklet also improves their momentum resolution, \[25, p.2\] This makes it possible to reduce the initially prohibitively large combinatorics of the calculations by means of extrapolation. Starting from each tracklet seed, a projection is performed, and stubs in accordance to the projection are added
to the collection. Track parameters are then obtained by performing a linearized $\chi^2$ fit \cite{27}. The tracklet algorithm also assumes the stub to come from a non-displaced origin, \cite{25} p.6

**Time multiplexed trigger (TMT) approach**

The collaboration of the KIT and the UK track trigger group is working on a time-multiplexed track triggering approach that utilizes an FPGA-based Hough transformation as the track finding step, followed by an additional least squares fit, also realized in FPGA.

The Hough transformation reduces the amount of stubs, and provides initial conditions to the subsequent fit. It is a global method that scales linearly with the number of stubs and also strictly relies on the track’s origin to coincide with the origin.

In this work we will reimplement their approach based on GPUs.
6 GPUs

In this section, we give an overview of what a GPU is. The technological advances of recent years, as well as some of the specifics that are important for low-latency systems are discussed.

Modern General Purpose Graphics Processing Units (GPGPUs) are fully programmable, multi-core processing units. GPUs have a Single Instruction Multiple Data (SIMD) computing model, meaning GPUs are especially capable of performing the same operation on different datasets. Compute units are arranged in streaming multi-processors (the NVIDIA terminology) or stream units (the AMD terminology). Inside such a compute unit a single execution branch is called a thread. Threads carry out single computations. They are the smallest ‘unit’ of execution and can synchronize and exchange data. The synchronization and exchange happen inside the very fast on-chip memory. Synchronization across compute units is relatively expensive and comes with some limitations. Threads are organized in warps (NVIDIA terminology) or wavefronts (AMD terminology). The software used for NVIDIA cards is the proprietary CUDA framework. The software by AMD, called OpenCL, on the other hand is open and can run not only on NVIDIA and AMD cards, but also on regular CPUs and other devices.

A program running on a GPU is called a kernel. A warp in CUDA consists of 32 threads, a wavefront in OpenCL consists of 64 work items. Inside a warp or wavefront, synchronous execution is guaranteed. The execution happens in lockstep. Threads in a warp can share and swap register memory. Warps and wavefronts are organized in thread blocks (NVIDIA) and work groups (AMD). Inside such a group, data can be exchanged using shared memory. This is slower than register memory but still very fast. Synchronization between thread blocks is very expensive in terms of latency, as it uses much slower global memory. Implications of this for our use case are discussed in some detail in section 11.1.

6.1 Technological Advances

Moore’s law states that computational power doubles roughly every two years [20]. The end of Moore’s law has been declared many times, with manufacturing processes starting to reach the limits of the physically possible. It will become harder to keep building devices that outperform their predecessors in the same way as before. One solution lies in building systems that grow more powerful in terms of parallel computational power. GPUs have undergone rapid development in recent years. Figure 16a illustrates the relatively higher growth rate of GPUs relative to CPUs in terms of computational power. The
Table 2: Bandwidths for different versions of the PCIe standard with respect to the number of used lanes.

<table>
<thead>
<tr>
<th>PCIe Version</th>
<th>Line Code</th>
<th>Transfer Rate</th>
<th>Per lane BW</th>
<th>16-lane slot BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.x</td>
<td>8b/10b</td>
<td>2.5 GT/s</td>
<td>2 Gbit/s</td>
<td>32 Gbit/s</td>
</tr>
<tr>
<td>2.x</td>
<td>8b/10b</td>
<td>5 GT/s</td>
<td>4 Gbit/s</td>
<td>64 Gbit/s</td>
</tr>
<tr>
<td>3.x</td>
<td>128b/130b</td>
<td>8 GT/s</td>
<td>7.877 Gbit/s</td>
<td>126.031 Gbit/s</td>
</tr>
<tr>
<td>4.x</td>
<td>128b/130b</td>
<td>16 GT/s</td>
<td>15.754 Gbit/s</td>
<td>252 Gbit/s</td>
</tr>
</tbody>
</table>

Data shown is given in terms of theoretical single precision GFLOPS. In order for this statement to be valid, however, the specific algorithm at hand has to be intrinsically parallelizable. With respect to the latency of a system, the speed-up is given by Amdahl’s law (15)

\[ s_{\text{latency}}(f, n) = \frac{1}{(1 - f) + \frac{f}{n}}, \]  

where \( s_{\text{latency}} \) is the theoretical speed-up, \( f \) is the fraction of the calculation that benefits from parallelization, and \( n \) is the number of processors used. For the case of the Hough transformation, \( f \) is in principle close to 1, although some caveats apply. An in-depth discussion of the parallelizability of the Hough transformation is given in section 11.1.

Memory, clockrate and number of cores

Up till recently, GDDR5 was the dominant type of memory used in GPUs. High Bandwidth Memory (HBM) technology is a novel way of building the memory of a GPU (2). Beginning this year, the new standard was introduced by Advanced Micro Devices (AMD) in their Fury series cards. Instead of placing the Dynamic Random Access Memory (DRAM) side-by-side, which has evolved into a bottleneck for various reasons, they are instead stacked on top of each other. The chips are arranged in this way using Through-Silicon Vias (TSVs) to connect the memory regions to the GPUs processor more directly. This decreases connection length and, thus, increases throughput. Table 3 gives an overview of some key differences in performance between the old and new technology. For example, the new memory type drastically increases the bandwidth for accessing the external global memory. Those accesses are typically very costly in GPU-based calculations. The increased bus width should help reduce memory-related latency by a lot. Furthermore, HBM reduces the power consumption with respect
(a) Theoretical Peak Performance for single precision calculations for different GPUs and CPUs. Showcases the relatively higher growth rate of GPUs.

(b) Same as above, but showing the development of theoretical peak memory bandwidth over time.
Table 3: Comparison of the properties of High Bandwidth Memory and GDDR5. HBM should reduce memory access latency drastically [17].

<table>
<thead>
<tr>
<th>Feature</th>
<th>GDDR5</th>
<th>HBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus Width</td>
<td>32</td>
<td>1024</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>1750 MHz</td>
<td>550 MHz</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>28 GB/s per chip</td>
<td>&gt;100 GB/s per stack</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.5V</td>
<td>1.3V</td>
</tr>
</tbody>
</table>

to its predecessor. The power consumption had become a major bottleneck that threatened the growthrate of computational capabilities. Since GPUs equipped with HBM were not yet available when this work was started, the results shown here still use GDDR5. We expect our results to be significantly better, when upgraded to this new technology. Figure [16] showcases the possible implications of the boost in bandwidth associated with such an upgrade. The Tesla K40 card is the one used in this work, whereas the P100 is the first NVIDIA card to utilize HBM (skipping HBM v1.0 and going directly to 2.0).

6.2 Interconnects

Peripheral Component Interconnect Express (PCIe) is a high-speed serial computer expansion bus standard. The data transfer scheme to the GPU used in this work relies on this standard. Here, we give a brief introduction, as well as an outlook of the upcoming version of PCIe.

It can transmit data in full duplex and communicates via a logical connection called an interconnect or link. The current version is PCIe v3.0 and was made available in November 2010. Its successor, PCIe v4.0, is expected to be released in 2017. It will lead to a doubling of the available bandwidth. Other interconnects, most notably NVIDIA’s nv-link, are starting to see adaption and wide-spread usage in High Performance Computing (HPC) applications. The increase in throughput of nv-link, as compared to PCIe 3.0, is a factor of ~ 5–10 depending on the application. Future work will have to evaluate the impact of using nv-link as our interconnect. It will be necessary to evaluate the difference in throughput, as well as the possibility to establish connections between different GPUs. This could potentially open up new ways to optimize the workload occupancy, see Section [11.1]. The PCIe communications used to transmit data to the GPU normally involve the host operating system’s Central Processing Unit (CPU). Some graphic cards allow the usage of a communication protocol called RDMA.
is a technology that facilitates remote access from one device’s memory to another device’s memory without the involvement of the operating system.

The memory mapping used in modern operating systems involves memory virtualization. Memory accesses are not direct, but rather a device sees a virtualized representation of the memory that needs to be translated back into an actual memory address by the operating system kernel. This produces large amounts of transaction overhead, making it unfeasible for the low latency performance we are aiming for. Usually, the bottleneck for low latency systems built on GPUs is the overhead produced by the data transfer from CPU host memory to GPU device memory.

RDMA allows for zero-copy operations. It removes the need to copy application memory to data buffers inside the Operating System (OS). This technology allows us to partly mitigate the costs of those transactions, making further investigation of its use for GPU triggers feasible in the first place. The Compute Unified Device Architecture (CUDA)-specific implementation of RDMA we are using is called GPUDirect. Figure [17] provides a conceptual description of the technology.

gdrcopy is ‘A fast GPU memory copy library based on NVIDIA GPUDirect RDMA technology’[12]. With the introduction of CUDA 8, it becomes feasible to perform relatively low-overhead and low-latency copies via RDMA. We use a modified version of this library to pin the GPU memory and make it accessible to the FPGA.

Avoiding Overheads

In order to create a trigger system that can meet the latency requirements of the proposed L1TT, we have to exclude the overheads associated with launching kernels. These overheads are typically of the order of 50 µs or even more. They depend on various criteria, such as the size of allocated buffers, the chosen work group dimensions, as well as the size of the memory region that needs to be dynamically allocated inside the kernel. Shared memory allocation needed to allow synchronization between different work groups produces a large overhead.

This makes it necessary to have a continuously running kernel, called a spinning kernel. In order to achieve this, the DMA engine on the FPGA will be completely controlled by the GPU from inside the running kernel.

6.3 Comparison of FPGAs and GPUs

Since we are proposing to replace a system that would traditionally be realized using custom hardware such as FPGA, it is instructive to compare
Figure 17: The basic concept of RDMA connections. The additional latency penalties of virtual memory are cut out almost entirely by avoiding the CPU.

Table 18 gives a brief overview of some key features. In order to make a fair comparison, we choose two devices that are manufactured using 28 nm technology. The main strength of the FPGA is its data transfer capability. The numbers given here are for custom interfaces. It is clearly superior to the GPU in this respect. For GPUs, the data transfer is given assuming a standard protocol, namely PCIe (PCIe) v3.0. It is noteworthy that GPUs have only been in use for HPC applications in the last decade. Although they are becoming more widespread, communication standards and protocols, especially for low latency applications, are not yet well established. Instead of being able to implement custom protocols, we rely on proprietary solutions.
Looking at the other features, we notice that a modern GPU has many advantages compared to a FPGA. For accurate floating point operations and when using larger amounts of memory during calculations, the GPU has clear advantages over the FPGA. It should be stressed that we compare single precision floating point operations here, a task at which the FPGA is weaker by design. In the regime of bit-level operations and fixed-point integer calculations, which a lot of HEP trigger applications design towards, the FPGA is superior. This is true in terms of computational power as well as energy consumption.

The current trigger system’s electronics are made up of customized Versa Module Europa (bus) (VME) modules. The complexity of the system makes maintenance very difficult. For this reason, there are already ongoing efforts to use as many commercial off-the-shelf component (COTS) components as possible [16]. A system based on commercially available GPUs with standardized interconnects, such as the one proposed in this work, would certainly be beneficial.
7 The Software Trigger

In this section we give an overview of the structure of the proposed track trigger solution.

The data coming from the front-end is expected to have undergone sparsification as well as the process of stub building. The stubs arrive compressed, with coordinates relative to the center of the corresponding sector. In our benchmark setup, that is not yet a fully functional trigger system, the data is preloaded into an FPGA and transferred into the GPU over the PCIe bus using RDMA technology.

The GPU computes the Hough transformation in $r$-$\phi$ and seeds track candidates. It takes the stub bend into account, as given by the detector electronics. Additionally it keeps track of the layers the stubs are coming from and applies filtering criteria. Track candidates fulfilling the criteria are then fitted using a linearized $\chi^2$ fit. While an implementation exists, it is not shown here. This is due to the lack of optimization and to open questions regarding load balancing, see \[15\].

The current implementation only performs the track finding step, using different variations of the Hough transformation. To this end, we reimplemented the algorithm used by \[4\] as well as our own version of the Hough transformation.

Tracker layout

The tracker layout used for this study is the current baseline geometry. It uses 5 disks in the endcaps and 6 disks in the barrel. A good discussion of its performance is given by \[1\]. An overview of the tracker layers is given in figure \[19\]. It was generated and tested using the tklayout tool \[6\].

Partitioning

The tracker is assumed to be divided into 9 sectors in $\eta$ and 32 in $\phi$.

$\eta$, the pseudorapidity, is defined as

$$\eta = -\ln \tan \frac{\theta}{2}. \quad (29)$$

Track finding can be done inside each of the sectors independently. This leads to a lower workload for each sector, but the overall workload actually goes up, due to the necessary duplication of stubs in the overlapping regions of the sectors. Future implementations will have to increase the sector size, as
Figure 19: Baseline geometry of the tracker used in this work, 3PS×32S 5 disks. The more sensitive PS modules provide a better resolution. The design allows a momentum cut and has good enough resolution to aid the triggering decision in the global trigger. The layout was designed by the collaboration, using the tklayout tool ([6], [5]) soon as hardware restrictions allow for it, see Section 11.1. The partitioning of the detector is done in such a way that high enough \( p_t \) tracks will not cover a \( \phi \) or \( \eta \) range bigger then the overlap of two sectors.

**Partitioning in \( \eta \)**

Normally, equal ranges in \( \Delta \eta \) have approximately the same number of particles. Due to the stub building process and the associated cut in transverse momenta, this property is lost. Thus, in order to maintain roughly the same amount of particles per sector, the partitioning in \( \eta \) is instead based on straight lines in \( z \). This partitioning scheme uses a predetermined, optimal pivot point in \( r \), called \( r_z \). To decide whether a stub lies in a given \( \eta \)-sector, the beam window \( w_z \) is used. It specifies the region around the collision point in the direction of the beam axis in which collisions occur. For the CMS it is \( w_z = 15 \text{cm} \). Furthermore, we choose a pivot point \( r_z = 45 \text{ cm} \). This value was determined by [4], who ran simulations of the detector. It minimizes the stub duplication, while maintaining high efficiency. As a consequence, the overlap of \( \eta \) regions within the inner detector regions is kept small. Given those values, we calculate

\[
z_{\text{min,max}} = \frac{r_z}{\tan \theta(\eta_{\text{min,max}})},
\]  

(30)
where $\theta(\eta_{\text{min}, \text{max}})$ corresponds to the values of $\theta$ at the boundaries of the $\eta$-sector.

**Partitioning in $\phi$**

The center of the $i$-th $\phi$-sector, assuming $n_\phi$ total sectors, is given by

$$\phi^i_c = \frac{2\pi(0.5 + i_\phi)}{n_\phi}.$$  \hspace{1cm} (31)

Stubs are duplicated in the overlaps of the sector, depending on the maximum curvature as given by the momentum cut in the transverse momentum, determined by the simulated front-end electronics. Stubs coming from different clusters in the strips of a module are dismissed as fake.

Furthermore, they are assumed to be given to the trigger, using local sector coordinates.

$$\phi_{\text{loc}} = \phi - \phi_{\text{center}}$$

### 7.1 Data Compression

In order to save bandwidth and reduce the relatively costly access to global [GPU] memory, the local stub coordinates are transmitted in a compressed format. The compression is very straightforward and uses the limits for the values given by the detector geometry.

The choice of the amount of bits per stored variable depends on the detector resolution of the variable.

We perform a binning on each variable within its allowed range. That is, we find a number of bits $n_{\text{bits}}$, such that the size of one bin, representing the value is below the achievable resolution of the detector:

$$\frac{x_{\text{max}} - x_{\text{min}}}{2^n_{\text{bits}}} < \delta x_{\text{det}}.$$  \hspace{1cm} (32)

The data for each stubs is compressed into two 32 bit words. The validity bit for the stub is put at the beginning of the first word. We assume a constant size, corresponding to 160 maximal stubs to be transmitted to the [GPU]. This means that most of the time we transmit more data than we would need for the actual stubs. If there is no stub at a given region in memory, this bit is set to zero. It will also be set to zero by the bend filter that is assumed to take place before the trigger.

In theory, it is possible to fit two stubs into a total of 96, using 48 bits per stub. We do not compress the data down to this level, because the GPU can only read in data regions with a minimum size of 32 in one read
Table 4: Overview of the used compression scheme for the transmitted data words. The detector resolution is given, as well as the resulting resolution for each variable. The method is adapted from the one used by [4], but slightly changed to suit the GPU.

<table>
<thead>
<tr>
<th>stub valid</th>
<th>$r_t$</th>
<th>$\phi_s$</th>
<th>bend</th>
<th>blank</th>
</tr>
</thead>
<tbody>
<tr>
<td>[bool]</td>
<td>[mm]</td>
<td>[rad]</td>
<td>[strips]</td>
<td></td>
</tr>
<tr>
<td>det resolution</td>
<td>1.56</td>
<td>5e-5</td>
<td>0.25</td>
<td>-</td>
</tr>
<tr>
<td>resolution</td>
<td>-1.01</td>
<td>4.79e-5</td>
<td>0.25</td>
<td>-</td>
</tr>
<tr>
<td>range min</td>
<td>-515</td>
<td>-0.19</td>
<td>-8</td>
<td>-</td>
</tr>
<tr>
<td>range max</td>
<td>515</td>
<td>0.19</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>bits</td>
<td>10</td>
<td>13</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5: Same as above, second data word. This package also includes information needed by the fit, that is not yet implemented. Includes information of the allowed bin range for the Hough transformation. Also includes the layer number, using the numbering scheme discussed in Section 9, for use in the minimum layer filtering condition.

<table>
<thead>
<tr>
<th>$z$</th>
<th>module type</th>
<th>$\sigma_x$</th>
<th>$\sigma_x$</th>
<th>layer id</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>[mm]</td>
<td>[a.u.]</td>
<td>[bool]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>det resolution</td>
<td>1.56</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>resolution</td>
<td>1.56</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>range min</td>
<td>-3200</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>range max</td>
<td>3200</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>7</td>
</tr>
<tr>
<td>bits</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

operation. This would lead to duplicate memory access in the same working group, which we avoid.

### 7.2 Detector Layers

As discussed in Section 9, we want to make sure that track candidates have stubs coming from at least 5 different layers. To reduce the amount of data transferred, as well as the amount of data needed to store this information, we again follow work previously done by [4]. Using the partitioning scheme in $\eta$ to our advantage, we note that it is not possible for a particle to pass through arbitrary sections of the detector.

As discussed, the particle trajectory, resembles that of a straight line in the $r$-$z$-plane, not changing it’s value of $\eta$, we can exclude certain layers, for
a given track parametrization. This makes it possible to reuse the same layer number for different layers, as long as there is no chance of the track belonging to the same track. In this way we find that it is sufficient to have a total of seven different layer numbers while assuring validity in counting the number of layers. We thus assume this information to be transmitted within three bits for each stub, as seen in table 4. This further reduces the needed bitmask for the layer condition in the Hough transformation to the size of just 8 bits. Where seven bits would be possible are but are padded for simplicity, as each read operation is bound to be 32 bits in size.

8 Validation System

The results obtained from the triggering procedure are tested against the MC truth Tracking Particle (TP) information, as obtained from CMS Software Framework (CMSSW) 1. The data comes from MC simulations of the detector geometry carried out with GEANT, see 3. In the analysis of the MC data, CMSSW provides ways of distinguishing produced hits that come from a TP or are just from Pile-Up. Furthermore it allows to check whether the hit is from an IT Bunch Crossing (BX) event. To use it with GPUs, a software framework was written that reimplements some of the CMSSW functionality locally, as there are no available CMSSW workstations with GPUs available. The data is taken from the Worldwide LHC Computing Grid (WLCG) and stored in ROOT data framework (ROOT) files, that contain all relevant information that comes from the detector for local analysis and benchmarking. This establishes compatibility of the GPU results with the CPU version running in CMSSW. The software performs an in-depth validation of the physics performance of the track triggering algorithms. This validation procedure was developed by 4, which we were thankfully allowed to use.

8.1 Tracking Particles

The TP are produced during the simulation step, within the CMSSW. They provide MC truth information for the stubs coming from the detector. They are not restricted to actual physics events of interest, but include all sorts of background occurring in the detector, such as cosmic rays and short-lived secondary particles. Not all tracking particles are actually useful for measuring the efficiency of a track trigger. CMSSW provides a utility for picking TPs according to the problem at hand, called TrackingParticleSelector, which is part of the RecoSelector class.
It allows the users to define restrictions on the tracking particles (TP) and determines whether a given TP meets those requirements.

The following properties were chosen for tagging tracking particles for use in measuring algorithmic tracking efficiency:

The quantity minimum number of associated stub layers, is different from the also available numberOfTrackerHits. The former counts multiple stubs on overlapping detector layers as one hit. The latter counts each stub separately. We want the former definition, as it suppresses fake track candidates.

In general it should be stressed that we define our efficiency in such a way that disregards tracks that we are intrinsically not able to reconstruct. This applies to tracking particles that have fewer than two layer clusters associated with them. This can happen for multiple reasons. If, for example, a tracking particle has a high enough momentum to pass our detection threshold, but manages to slip through a gap in the silicon layers. It will not form a valid stub. It might also be the case, that a background event passed the layer shortly before the tracking particle, leading to dead time. The incoming tracking particle then fails the stub building, for the lack of a second cluster belonging to it. If those effects happen often enough for the tracking particle to fail the minimum layer requirement, it will be disregarded for the efficiency measurement of the algorithm.

The stubs are matched to the TP during the simulation step. This information is gathered from the CMSSW.

8.2 Stubs

In order to quantify the efficiency of track reconstruction, we distinguish between fake and real stubs. Stubs are either primary, secondary or fake. A stub is fake if there is no associated tracking particle. Primary and secondary stubs do have a tracking particle associated to them. Furthermore, we can make sure that both cluster hits making up the stub actually come from the same tracking particle. A stub is primary if its associated tracking particles distance from the IP is smaller than 25cm and secondary otherwise. This makes sense as the algorithm assumes an impact parameter coinciding with the interaction point.

8.3 Subsequent Fit

A fitting procedure based on the code used by [28] was implemented on the GPU. It uses a linearized $\chi^2$ fit to obtain optimal track parameters assuming helixoidal tracks. It was used within the framework of CMSSW to establish the performance of the track finding. Due to unresolved issues concerning
proper work load balancing, it is not included in the current benchmark setup and its further discussion had to be excluded from this work. Preliminary, unoptimized results show execution times below 10 µs.

8.4 Used Datasets

The main dataset used in this work is an official release validation sample consisting of TTbar Events. It can be obtained from WLHC! (WLHC!) Data Aggregation System (DAS)¹/RelValTTbar_14TeV/tomalin-CMSSW_6_2_0_SLHC27_PU204544b4197e280451be55a8f4b257b4ea/USER. The samples were created by Ian Thomalin.

It consists of both 140 and 200 PU scenarios. We use the simulation data mainly to establish the general correctness of our results, especially in terms of the proposed hexagonal approach (discussed in Section 10).

We did not perform an in-depth physics analysis of the achievable performance with regards to efficiency and trigger rates. This not necessary, as the result is equivalent to that presented by [4]. For an in-depth analysis of the performance in terms of efficiency and trigger rates, we would like to refer the interested reader to their results.

Our result shows differences in behavior with respect to the produced fake and track candidate rates. While this has benefits for the overall work load put on the system, the efficiency and momentum resolution are not affected, as discussed in Section 13.

Additionally, low statistic samples with electrons, pions, kaons and muons have been performed. The trends in efficiency, the effects of pile-up and duplicate/fake rates stayed the same, and are not presented in-depth.
9 Hough Transform

The Hough Transformation (HT) is a commonly used feature extraction technique for corner and edge detection. It was widely studied and is well understood both in terms of image analysis and for track finding. Implementations for GPUs exist as well. These implementations do, however, depend on fixed image sizes, as opposed to the varying number of stubs we have as our input. This makes the commonly used techniques in image analysis not applicable here, see Section 11.1. It was originally introduced in 1936 for analysis of bubble chamber tracks.

9.0.1 Theory

We express the lines to be detected, using the Hesse normal form

\[ r = x \cos \theta + y \sin \theta. \]  

(33)

We iterate over the possible values of \( \theta \). In order to do so, we have to choose a binning in \( \theta \), giving us \( n \) possible values between between 0 and \( \pi \). The choice of the binning is crucial to the performance and in-depth discussions for our use case are given in Section 10. For each value of \( \theta \) the corresponding value of \( r \) is calculated, as given in equation 33. The \( n \theta \)-r-pairs obtained in this way, lead to \( n \) votes in the Hough space. Each of these votes represents a possible \( \theta \)-r parametrization of the line.

This leads to sinusoidal waves in the Hough parameter space that form cluster points at the true values for the line. This accumulation procedure produces a histogram of the evidence for the set of all possible lines. The actual line parameters are then obtained by searching for local maxima in the parameter space histogram.

Features of the transformation

The Hough transform is a global method. The computation time grows linearly with number of hits present in the event. This makes it a relatively robust choice for the upgraded system where the event pile up is expected to be very high. Because it is a global method, it does not face the same combinatorics as road-based track finding approaches typically do. It is also not as sensitive to uncertainties in the measurement itself, due to its binning and discretization of the parameter space. Generally, in order to get a good result, we have to be careful not to choose a too small cell size. This ensures that enough votes belonging to the same track fall into the correct bin and can be detected. This is especially true with regard to deviations from the
idealized track model underlying the method. On the other hand, votes should be in as few bins as possible. Ideally there should be only one local maximum that indicates the true location of the seeded track parameters. The Hough transformation will always find both tracks that are duplicates of true tracks as well as find fake candidates, that do not belong to any real track.

9.1 Parametrization

The Hough transform maps points from the $r$, $\phi$ plane to the space of inverse momentum and production angle $m$, $c$.

The underlying assumption is that tracks are parametrized by an ideal helix.

Since the detector is already applying a filtering step for tracks by means of stub building, we can consider the stubs entering the Hough transform to be high $p_t$ tracks. We can make the small angle approximation

$$\sin \Delta \phi \approx \Delta \phi.$$  

(34)

to the helix. The approximated $\phi(r)$ is then given by:

$$\phi(r) \approx \pm \frac{0.006}{p_t} r + \phi_{\text{stub}}$$

Where $\phi_0$ is the production angle of the track candidate and $p_t$ it’s transverse momentum. We interpret the approximated result as a linear equation

$$\phi(r) = -mr + -\phi_0$$  

(35)

with slope $m$ and y-intercept $\phi_0$.

We then iterate over the allowed values of $m$, each corresponding to a different momentum ranging from 3 GeV to $-3$ GeV. We note that in order for the Hough transformation to work, we have to assume that the transverse impact parameter $d_0$ is equal to zero.

9.2 Localized Coordinates

As discussed before, the Hough transformation receives local coordinates, with respect the center of the $\phi$-sector currently processed. This has multiple benefits. For one, it allows a more effective compression of the data, as it severely limits the possible range of the $\phi$-coordinate.

In this way, we can process smaller amounts of data, as well as simplify the equation for the Hough transform. If the coordinates were not sector-centric, we would have to adapt them online, with respect to the sector they
are in. Otherwise the memory footprint of the Hough space would reach an
unmanageable size.

The sector specific coordinates make it possible to reuse the exact same
kernel for each sector. This greatly improves upon the capability to use
multiple GPUs according to current availability, e.g. idle ones, without having
to take care of passing them sector specific information.

9.3 Size of parameter space

The range of the axis is proportional to \( p_t^{-1} \), it is simply given by

\[
\Delta m = 2 \frac{1}{p_{t,\min}} \approx 0.66. \tag{36}
\]

The range for the axis representing the production angle \( \phi_0 \) is given by:

\[
\Delta c = 2 \frac{\pi}{n_{\phi-bins}} \approx 0.20 \tag{37}
\]

9.4 Centralized Hough transform

Instead of just calculating the Hough transform as given in 35, we adopt the
approach of [4].

In this approach, we do not calculate the production angle at the pro-
duction point near the beampot. Instead we use the track bend relative to
a chosen distance from the beampot \( r_\phi \). This has the benefit of producing
lines in Hough space that have both positive and negative slopes, to the ef-
fact that lines that do not belong to the same track grow apart faster inside
the parameter space. The result is less fake and duplicate track candidates
as opposed to the naive approach. The approach has the added benefit of
decreasing the effects of changes in the track’s curvature, caused by energy
loss, as well as deviations from the idealized track caused by scatterings.

As those effects grow over the distance traversed by the particle, choosing
a pivot point in the middle of the detector causes the deviations with respect
to this point to be smaller than they otherwise would be. This stabilizes
the method and increases the efficiency of the track seeding step. It also allows
us to tweak the parameter space, by choosing the relative amount of bins for
each axis that maximizes efficiency and minimizes fake candidates.

This centralized Hough transform then takes the form

\[
\phi = m(r - r_c) + c. \tag{38}
\]
This allows us to precalculate the values taken on by $m$, which are now the same for all sectors, and store them in local memory of the kernel, reducing the amount of performed computations drastically.

This calculates the values at the edge of every $m$-bin of the Hough space. Since two cells share a value at each edge, we do not store them twice, saving local memory by roughly a factor of two.

### 9.5 Choosing the amount of bins

In order to reduce the computational effort required to calculate the Hough transformation, it is a good idea to limit the allowed values for the slope $s$ of possible of lines in Hough space to be within the range

$$-1 < s < 1.$$  \hfill (39)

This way, it is not possible for a given $m$-bin in the Hough array to produce more than two possible values in $c$. Therefore, it is only necessary to calculate the two $c$-values for each $m$-bin at the outer edges of the $m$-bin, as opposed to the case where one would have to look for more possible $c$-values.

But we also want to maximize the value of the slope, as larger slopes will lead to fewer votes in the Hough space, because the lines diverge more quickly. Which is beneficial, as it creates more localized clusters in the parameter space.

In order to achieve this, we have to choose the relative amount of bins accordingly.

If we use centralized coordinates, the maximum line gradient is solely determined by the maximum distance from the chosen pivot point. For the slope $s$ we get

$$s = \left| \frac{(r_{\text{max}} - r_c) B c}{2} \right| 10^{-11}.$$  \hfill (40)

For a given starting value of bins in $m N_m$, and a desired total number of cells $N_{\text{total}}$ we can then calculate the number of bins in $c N_c$ to fulfill the requirement of limiting the slope, via:

$$N_m = \text{ceil}(\sqrt{N_m^2 N_{\text{total}}})$$  \hfill (41)  

$$N_c = \text{floor}(\sqrt{N_m^2 N_{\text{total}}})$$  \hfill (42)
9.6 Threshold condition

The accumulator of the Hough map is not solely based on the number of hits, as this would dramatically increase the number of fake track candidates. Instead, we keep track of the detector layers present in every given cell of the Hough map. In the implementation, this is done by performing a bit-wise OR operation on a bitmask representing the six different layers of the barrel. The condition for a valid track candidate coming from the Hough map is a minimum of 5 different layers contributing to a given cell. Dregarding the amount of total hits, a cell not fulfilling this criterium is not considered a possible track candidate.

9.7 Filling the Hough map

9.7.1 Bend Filter

The bend filter approach relies the available information of a stub’s bend, as given by the detector electronics. This rough estimate of the transverse momentum allows us to limit the allowed $p_t$ range for each stub, sometimes even completely removing them, and thus limit the allowed ranged of m-bins within the Hough transform. This approach drastically reduces the number of produced track candidates passed from the Hough algorithm to the track fit. Given the stubwidth we can make an estimate of bend of the particle producing this stub. From this information, we can make an estimate of which values of $m$ in the produced Hough array are allowed.

9.8 Hough parameters

The amount of cells in the m and c direction as well as their absolute and relative sizes are chosen in such a way, that we get a maximum of 2 distinct c-values for one stub and one Hough map bin respectively. This means that the next cell to be filled into the Hough space has to be either to the right, the left or the top of the current cell.

9.9 Resulting Track candidates

In figure 21b we see an example of the resulting Hough transformation for a TTbar PU140 event. The clustering points indicate possible track candidates. The lines that do not go through all of the momentum bin ranges are a result of the effects of the bendfilter. We use the stub bend information, as given by the detector layers to reduce the fake rate.
In the top right we can see a clustering point that comes solely from incidental overlaps of stubs not belonging to the same candidate. A filtering based entirely on the number of votes, would have produced a fake candidate at this point. The resulting filtered transformation is shown on the right. We see that the fake candidate has been suppressed by demanding the stubs in valid cells over the threshold to come from at least 5 different detector layers as discussed previously.

We get a total of three resulting track candidates in this case, where two of those candidates are actually duplicates of the one true track present in the sector for this sample.
Figure 20: Visualization of the concept of the Hough transformation. Each entry in the histogram corresponds to a possible curvature of the track.
(a) Filled Hough map for one sector, before applying the filtering step. Dataset is TTbar PU140 event.

(b) Same as to the left, but after the filtering condition has been applied.
10 Hexagonal Hough Transform

In this part, we present our own version of the Hough transform. It utilizes a histograming method using a hexagonal binning scheme and allows for a better representation of the allowed parameter space for each stub. It shows superior properties in terms of the ratio of true and duplicate track candidates as compared to the regular approach, see [13]. The implementation exploits fundamental geometric properties of the binned space.

10.1 Motivation

The Hough transform both benefits and suffers from its binning. On one hand the binning diminishes the negative impact of the finite detector resolution, as well as the inadequacies of our underlying assumption of helixoidal tracks. Those assumptions lose validity due to energy loss and scattering. Its binning helps to smooth those effects out by allowing the lines not to meet each other exactly. Rather, they are spread due to their loss in momentum and subsequent deviation from the ideal helix. A finer binning allows us to give better approximation of the track candidate parameters. On the other hand, a binning that is too coarse leads to finding more track candidates than there are tracks in the dataset. A coarser binning leads to less well defined local maximal in the parameter space for a given track candidate. It also increase the rate of fake candidates. Our layer condition mitigates the latter effect somewhat, but especially at higher pile-ups this becomes important. Those fake and duplicate candidates are artefacts of the transformation and we have to try to avoid them.

Assuming perfect helixoidal tracks, we could make the binning infinitely small and just look for local maxima. This would allow us to find track candidates with near perfect estimates of the track parameters. Given the effects discussed above, we can not make the cells infinitely small. We have to allow for an uncertainty in both coordinates in order to find maxima.

The lines in the parameter space need to be rasterized as smoothly as possible. To this end, a new algorithm was developed where the bins in the Hough space are not rectangular, but instead hexagonal. The aim of this approach is to represent the straight lines of track candidates more truthfully, thus reducing duplicate and possibly also fake track candidates.

The hexagon is the highest-sided regular polygon with which it is possible to tesselate the two dimensional plane. The only other polygons capable of tesselating the plane are triangles and squares. The advantage of hexagons lies in the more densely spaced grid.

The details of this approach are discussed in the following.
10.2 Properties

The layout in terms of hexagonal coordinates and what we mean by columns is shown in figure [22b]. While the amount of bins in each row and column stays approximately the same, the amount of possible values for the seed parameters quadruples. The number of calculations carried out to provide a possible production angle value is equal to the number of possible momentum values and doubles. Even though we have twice as many values than in a rectangular grid, each hexagon is the same width in parameter space.

While the amount of calculations needed, to perform the transformation is twice as large as the amount of bins in one row (as adjacent rows are offset) the difference in computation time is minute. This is because we have to perform the calculation twice for each cell in the regular approach. There, the calculation has to performed at each edge, because each edge possibly leads to another value for the production angle. Using the geometric properties of the hexagonal grid, we only have to perform the calculation once, at the center of the bin. Additionally, we are able to guarantee only one vote per column. This is by construction. Therefore, the number of values that needs to be calculated remains the same as for a rectangular grid.

In the case of an implementation on the GPU, this property has the added benefit of doing away with some of the algorithmic branching present in the rectangular approach, see [11.1].

Parametrization

The relation of the width of a hexagon and the size of one of its sides is given by:

\[ s = \frac{w}{\sqrt{3}} \] \hspace{1cm} (43)

The height is given by \( h = 2s = \frac{2w}{\sqrt{3}} \). Throughout, we will refer to \( s \) as the size, \( h \) as the height and \( w \) as the width of a hexagon.

Since we have an irrational relation between the width and the height, we will introduce the dimensionless hexagonal axes \( h_x \) and \( h_y \). We arbitrarily set the width \( w \) of a hexagon to be equal to 1 in \( h_x \). In parameter space they have the same range as the regular transform. The dimensionless space will be used to calculate the index in the voting step.

Within a given row, there are \( n_x \) hexagons, each corresponds to a size \( \delta x_p = \frac{\Delta p_t}{n_x} \) in parameter space. Requiring the same bin width in the momentum axis as in the regular transformation, we define \( h_x^{\text{max}} = 32 \), whereas \( h_y \) will have to be carefully chosen.
10.2.1 Choice of bins

The size of the hexagon space in the $y$ direction has to be a multiple of $s$ for the binning to work as expected. Furthermore, we want to have the same amount of cells in all columns to simplify it with regard to a parallel implementation. We also want the hexagonal grid to cover all of the available parameter space.

As there are no straight boundaries, we put the cell centers on the edges of the allowed space, as indicated in figure 24 by the black lines. Laying out the grid in this way, we find that for a given number $n_y$ of bins per column, we have a total size $h_y^{\text{max}} = n_y(h + s) + s$.

The choice of $n_y$ depends on the maximum gradient we want to allow and is discussed in the following.

10.2.2 Allowed line gradients

Whereas we choose the maximum line gradient in the regular Hough transformation so as not to exceed a value of 1, in this approach we have more freedom to choose the gradient.

We can choose a maximum line gradient of

$$m_{\text{max}} < \frac{h}{w} \approx 1.547$$ (44)

while still satisfying our conditions that we want only one possible vote per row (see 22c). We can guarantee that there will be only one $\phi_0$-cell for each bin in $m$, while allowing for a larger gradient.

In order to do so, we have to choose the number of bins $n_\phi$ in such a way that

$$\frac{\Delta h_y(h_c^{\text{max}})}{\Delta h_x} < m_{\text{max}}$$ (45)

where $\Delta h_y$ depends on $\delta\phi^{\text{max}}$ in parameter space and $n_y$.

Benefits

Firstly, the increased value of the slope representing the possible track parameters leads to a better resolution of the clusters in $\frac{q}{p}$. This is because of the lines making up the clusters are tighter, due to the larger gradient. This should help to reduce the number of duplicate track candidates for a given clustering point and it indeed does, as we show in section 13.
10.2.3 Choosing the relative amount of bins

We want to choose the amount of bins in such a way that we restrict the maximum slope in the binned parameter space, just as we did in 9.5. Again the maximum line gradient in parameter space is determined by the maximum distance from our pivot point \( r_c = 65 \text{ cm} \). The maximum slope as a function of the number bins has to be smaller than \( \frac{h}{w} \) for our assumptions to hold. This leads to

\[
\frac{\tilde{\phi}_2 - \tilde{\phi}_1}{\Delta \phi} \frac{n_y s}{n_x} < \frac{h}{w}.
\]

(46)

If solved for \( n_y \), we get

\[
n_y < \frac{n_x}{s} \frac{h}{w} \frac{\Delta \phi}{\tilde{\phi}_2 - \tilde{\phi}_1}.
\]

(47)

The biggest possible value fulfilling the condition is \( n_y = 23 \).

10.2.4 Parameter space area

Given in units of dimensionless index space, the area of a single cell is

\[
A = \frac{3\sqrt{3}}{2} s^2 = \frac{\sqrt{3}}{2} \approx 0.87
\]

(48)

compared to the regular transform, where it is unity. Just as in the regular transform, the estimated track parameters are given by the values in the middle of each cell. The hexagon is much closer to a circle, (of radius \( s \)) around the center. Hence, we decrease the allowed spread around our track estimates. The allowed parameter space area for a given stub, is qualitatively analyzed in figure 23. Generally speaking, the allowed space decreases while the quality of the rasterization of the line increases by a relatively bigger amount due to the tessellation properties of the hexagonal grid.

10.2.5 Finding the correct bin

We take the value of \( \tilde{\phi} \) from the midpoint of a hexagonal column. The columns are equidistantly spaced including the edges of the allowed parameter space as given by the momentum cut. Using the formula from the Hough equation \( 38 \) we find the value in parameter space for the production angle. Finding the index in the hexagonal space is conceptually more difficult, but of comparable computational complexity.

We want to avoid algorithmic branching inside the GPU, so we apply a shifting scheme, depending on the currently evaluated column. This enables
us to read the shifting value from memory, depending on the bin under consideration, and then apply the same procedure for all bins. This avoids the added complexity of the alternating binning scheme. Effectively we perform a linear transformation, shifting each bin to be on the same height. We then perform the calculation of the histogram index. Then we undo our shift, by changing the index depending on the current column.

The applied shift on the dimensionless space is given by:

\[ h_{\text{shift}}^{\text{even}} = h + s \quad h_{\text{shift}}^{\text{odd}} = \frac{-s}{2} + h + s \] (49)

The additional shifting of \( h + s \) is applied to enable us to correctly find bins that are below the lowest bin, but depending on their slope might still enter the cell.

We are interested in finding the distance to the next lowest bin edge of this value, which we label \( \delta h \).

Now, we just have to apply discussed shift to \( h_y \), before finding the bin as discussed above. Expressing the result in terms of the combined height and gap of the hexagon \( (h + s = 3s) \), we find the value \( \tilde{h} \) in the usual way and get

\[ \delta h = \frac{h_y + h_{\text{shift}}}{3s} - i_{\text{bin}}(h + s) \] (50)

where the bin number \( n_{\text{bin}} \) is given by

\[ n_{\text{bin}} = \lceil \frac{h_y + h_{\text{shift}}}{3s} \rceil. \] (51)

If we are inside a bin, it holds that

\[ 0 < \tilde{h} - n_{\text{bin}}(h + s) < h. \] (52)

But we want to be able to tell if the line will cross the cell at its edge. We introduce a correction factor, that represents the distance in \( \delta h \) that the line will rise over the distance of the bin size. It is based on the maximum allowed slope, as constructed above:

\[ \delta h_{\text{err}} = 0.5 \frac{r}{r_{\text{max}}} s \] (53)

This exploits the choice of our bins and the allowed range of slopes. In effect, we assume that a stub with maximum distance from our pivot point will hit a hexagon at its edge if its distance is smaller than half the gap between two hexagons. For an illustration, see figure 22d. To accommodate for stubs having a smaller slope, the result is scaled by the ratio of the current value of \( r \) and \( r_{\text{max}} \). Which is also the ratio with which the slope scales.
This assumption is not actually true as the irrational relation between the axes introduced by the hexagons doesn’t allow a choice in the number of bins $n_y$ that realizes the assumed maximization of the slope. But we will exploit this to our advantage, as illustrated in section 10.3.

There are two cases that need to be distinguished. The first possible scenario is realized, when we are either already inside a given cell or about to enter it.

$$0 < /\delta h < 2s + \delta h_{\text{err}}$$ (54)

will be true and our bin is given by $n_{\text{bin}} - 1$. If we are below the cell, but are bound to pass through it, the relation

$$\tilde{h} - n_{\text{bin}}3s > 3s - \delta h_{\text{err}}$$ (55)

holds and our bin is given by $n_{\text{bin}}$. If neither relation is realized, we do not make a vote at all.

10.3 Effects on valid parameter space

As we have seen, we can find a bin but we do so using an assumption that does not actually hold. Here we will investigate this curious choice.

Given our choice in the number of bins being $n_y = 23$, we are not actually achieving the maximum allowed slope, but nevertheless pretend to and do the voting accordingly. What is the resulting behavior from a physical point of view in this case? The first thing to note is that the error does not propagate through the parameter space. It is solely present locally within a given cell. As the true slope is smaller, we effectively overestimate our parameter space, by voting for cells that are not actually present. This means we mark too many cells.

But what does correct mean in this case. We have a detector with a finite resolution and there is an error associated with the measurements of both $\phi$ and $r$. We have for example in the case of the detector barrel, an intrinsic uncertainty of half the strip length. On top of that, there are errors in our ideal helix model, as discussed before. By choosing bins that are not quite allowed by our track model, we are not actually making a mistake. We simply allow for cells that, given the uncertainties, are actually allowed to be marked as such, instead of discarding them solely because of an arbitrary binning of the parameter space.

In light of this observation, we stick to the bin choice as described by equations 54 and 52. We expect to achieve a slightly higher efficiency, due to the reasoning given above, as we accommodate for the error in uncertainty. We expect this choice to have a negative impact on the performance of the
track finding, that should result in higher rates of both fake and duplicate candidates. This effect is due to the higher number of allowed values in the parameter space.

On the other hand, we hope that in the case of duplicate track candidates, the properties of the hexagonal binning make up for this shortcoming. We are able to better resolve their true parameter space representation, after all. Relatively speaking, the penalty regarding fakes should be higher, as they appear randomly and the better resolution of clustering does not affect them at all. But more votes in the parameter space should see their relative rate go up by allowing for this uncertainty, as compared to duplicate candidates. We see this assumption confirmed in the results presented in section 13.

This brings to mind another possible approach for choosing the binning. Instead of taking into account the error and enlarging the allowed parameter space, we could also restrict it. In this case we hope to find a parametrization of the binned space, that reduces that amount of both fake and duplicate candidates at the cost of some efficiency. Using the exact same algorithm but changing the number of bins to $n_y = 29$, we allow the transform to ignore lines scratching at the edges of the hexagons. In this case we underestimate the lines ability to pass through a given cell. We effectively apply a momentum cut to the parameter space, filling \textit{fewer cells}. The resulting Hough map for our choice of bins is shown in figure \ref{fig:24} in section \ref{sec:11}.

The allowed parameter space gets smaller and we thus reduce the number of found candidates. In this case we expect a drop in efficiency, hopefully somewhat diminished by the grid’s superior properties. We expect a significant decrease in the number of both produced fake and duplicate track candidates. This expectation is confirmed in the section 13.
(a) Labeling of the variables, shown for the regular pointy top hexagon.

(b) Hexagons arranged in a grid. The numbers given indicate our choice of indexing the space.

(c) The solid line has maximal slope, at the hexagon center it fails to pass our prediction threshold (green box) for reaching the upper cell. The dashed line passes our slope corrected threshold condition (red box) for reaching the lower cell.

(d) The line represents the maximum allowed slope that fulfills our condition of having one cell per row.
11 Comparison of regular and hexagonal grid

In figure 23, we show a two dimensional histogram of the number of cells used for representing a stub in Hough space. The histogram spans all of the allowed parameter space for both $r$ and $\phi$.

The asymmetry, present in all three plots stems from our choice of $r_c$, which is closer to the innermost layers than it is to the outermost ones. That is why everything is shifted to one side. This also illustrates that choosing $r_c$ this way, we reduce the number of cells for one part of the detector, as only lines coming from the outer layers have maximum slope, thus crossing more bins in $\phi_0$.

The leftmost plot shows the number of cells, which in this case is equal to the area in dimensionless parameter space. The counts represent the bins needed to represent a given line for each possible combination of $\phi - \phi_c$ and $r - r_c$ in the regular Hough space.

The mid plot also shows the number of cells needed to represent a line, but this time for the hexagonal transform. We notice an overall higher number of cells.

From a computational point of view, this leads to less total workload for the regular transform as compared to the hexagonal one, if we just count the number of total write operations. If we are able to parallelize the operations, which we are able to do on the GPU, this seeming deficiency vanishes.

The third plot is the same as the one in the middle, but weighted by the area in parameter space. We can immediately see that this area is mostly smaller for the hexagonal Hough space as compared to the regular one.

From a physical point of view, it is better to represent the possible parameter space using more cells of with smaller total area. This allows us to simultaneously in our parameter space more truthfully. This should lead to a tighter approximation of the allowed parameter space, that should help to reduce the amount of duplicate and fake track candidates. Furthermore, given that the amount of starting parameters in the hexagonal grid is higher, as well as more tightly packed, the greater number of cells representing a line and the tighter rasterization should also lead to better starting parameters for the subsequent fitting stage.

In figure 24, we showcase a quantitative side-by-side comparison of two produced Hough maps for a total of 12 stubs belonging to two different tracking particles. The cleaner representation of the lines shown in black is quite obvious.

We observe that the maximum of the cluster is more well defined in the case of hexagons, where we have only a single cell with six votes. The regular transform has its local maximum spread over two cells. The same behavior,
Figure 23: Comparison of the regular and hexagonal parameter spaces. left: Number of cells for the regular transformation (equals the area in parameter space). middle: number of cells, hexagon. right: Area in parameter space hexagons.

this time with the duplicate track in a neighboring momentum bin can, be seen for the track candidate in the top left of the histogram.
Figure 24: Comparison of the regular and hexagonal binning behavior. Left: hexagonal transformation, 29 bins in $\phi_0$. Offers tighter binning and by means of the bigger slope decreases width of clusters in the parameter space. The cluster height increases. The negative impact of this is mitigated by the gap between hexagons. This is the momentum cut version, the binned representation of the line is without gaps. Right: regular transformation. Uses the standard rectangular grid. Is limited to slopes smaller then one, so as not to produce too many bins in $\phi_0$. The width of the clusters is bigger, not just in the index space, but also in the parameter space.
11.1 Implementation details and specifics of the GPU architecture

In this part we will discuss the specifics of the implementation with respect to the GPU. There are certain problems arising with the algorithm that have to be addressed, as well as certain limitations on the hardware side, that make the implementation with the card in use difficult.

Implications for memory structure

As we want to have a low latency system, we have to try to avoid the use of the external GDDR5 memory access whenever possible and move as much of the algorithm as possible into shared memory, or better yet register memory. All variables used within the calculation can be made to reside in register memory. This way we mostly use the fastest memory available. The voting process in the Hough transformation is a histograming step. While the GPU scheduler can hide global memory access, by having thread warps perform calculations while others perform the memory access, global memory access is still quite time consuming. It is on the order of hundreds of clock cycles. GDDR5 can perform multiple reads at once.

This brings with it certain restrictions in terms of data synchronization. In order for different threads to share data, they have to be within the same workgroup and the memory to be shared has to lie in the shared memory of the workgroup.

Synchronization

GPUs are fastest when the computation is *embarrassingly parallel*, meaning each computation thread is completely independent from the other steps. The problem, namely voting in the parameter space is not embarrassingly parallel. It involves a binning step, which is not ideal for a GPU. The problem is that of *race conditions*.

Adding a value into a histogram bin is a three step process:

- 1: read value from memory
- 2: increase value by one
- 3: write value back into memory

When doing this on a GPU, we encounter race conditions. Two threads can read in the same value, both increase it by one and then write the result back. The result will be incorrect. In order to circumvent this, we are forced
to use an what is called an atomic operation. These operations lock a certain region in memory, for the duration of the operation. This solves the problem of inconsistent results, but it comes at a cost.

During an atomic operation every other thread that would like to write to the same region in memory, is forced to wait. This results in poorer performance. This is especially true if we perform the atomic operation in global memory. Global memory access is relatively slow, so in order to remain good latency we have to avoid it whenever possible, especially when the operation forces the other threads in a warp to wait on it to finish.

This basically forces us to move all of the Hough arrays memory into the shared memory. The GPU is, in contrast to the FPGA not very good at performing bit-level instructions. It can do them in principle, as we do so successfully when we decompress the stub data. But this is an operation that does not need any synchronization and can be done in register memory. In the case of the atomic operation that we need for the binning, it is currently prohibitive to work on memory regions smaller than 32 bits. This forces us to define the Hough map as unsigned int variables each 32 bit in size. See section [15] for a possible solution.

The total size of the Hough array in memory is given by:

$$(2 + n_{\text{stubs}}^\text{max})bin_{p_t}bin_{\phi}32\text{bits} = 32768\text{bytes}$$

(56)

Where $n_{\text{stubs}} = 30$ is the maximum number of stubs we allow per cell and $n_{p_t}$ and $bin_{\phi}$ are the number of bins and 2 is comprised of one 32 bit bitmask storing the information of concerning hit layers and one for counting and one 32 bit value for storing the current stub count in each cell. The value of 30 for the maximum number of stubs, is required, given our large pile-up.

This size in memory is at the limit of what can be allocated by the Tesla K40c card used here. In fact, while it fits in shared memory, it is too big to be contained within a single threadblock. This basically forces us to split the computation of the Hough transformation into 32 threadblocks. This currently restricts our implementation to the following computational layout.

**Computational Layout**

We are forced to subdivide our problem into 32 threadblocks. Each block having as many threads as there are stubs. Each threadblock, carries out the calculation for one transverse momentum bin. This is possible as the calculations are independent of one another. On one hand this leads to a very low latency, on the other hand it introduces a lot of computational redundancy. This significantly reduces our achieved bandwidth as it wastes
resources. This means we are currently using a total of

\[ n_{\text{threads}} = n_p n_{\text{stubs}} \]  \hspace{1cm} (57)

The total number of available threads for the Tesla K40c is 2880, see \[\text{in the Appendix. This limits the total number of stubs that can be processed to 64 if we want to reach minimum latency. This is not a fundamental problem as the number of threads keeps increasing continuously, in fact using the K80 variant would already mitigate this problem. A } 4\times \text{ Tesla P100 solution, which is the newest iteration of NVIDIA GPU cards at the time of writing offers on the order of 12000 threads, potentially enough to process an amount of stubs corresponding to the whole tracker for 1 BX in parallel. As soon as we have sufficient shared memory at our disposal new possible solutions arise, see Section 15.}\]

11.1.1 Workload

Different solutions for managing and dispatching the workload have been investigated, but the aforementioned hardware restrictions with the given card limit our abilities. A solution based on the dynamic parallelism technology, offered by NVIDIA cards was tested. It allows to launch kernels, from within other kernels. This technology will see adaption in the trigger system of the Anti-Proton ANihilation DArmstadt (PANDA) experiment, at Facility for Antiproton and Ion Research (FAIR). While they also employ a version of the Hough transformation, that do not have the same latency requirements as is the case for the CMS trigger system. We have benchmarked the technology and found a decrease in kernel launch time overheads by a factor of \[\approx 4\]. The latency was around \[20 \mu s\] and thus above our maximum allowed threshold.

This leads to an inability to launch kernels, which makes it impossible to adapt the computational layout to the current amounts of incoming data for a single [GPU]. The consequence is, that once the number of maximum stubs to be processed is set, we can not alter in the case of lower workloads. The amount of incoming stubs varies wildly, forcing us to choose a rather large number. The rest of the threads thus remains idle.

11.1.2 Conclusion and Outlook

We have to acknowledge at this point, that in terms of flexibility with respect to load balancing, the current hardware severely restricts our computational throughput. The GPU architecture, that is designed to process images with static pixel layouts, loses a lot of its computational power when dealing with
varying workloads. In a normal environment, where limitations as strict as the current ones apply, this can be mitigated by just launching kernels, possibly speeding up the results using technologies such as dynamic parallelism. In our case, this is not feasible, due to our restrictions.

In order to balance the workload, at the moment we see two possible solutions, both beyond the scope of this work.

We could have multiple GPUs launched with kernels of varying grid setups. Events with different amounts of stubs would be dispatched by the multiplexer accordingly. This would require additional calculations, namely counting the stubs and then making a decision of where to push them to. Currently we do not have a good solution to this problem, as it raises questions about the feasibility of the currently applied approach, where the GPU requests the data. In theory it is possible to have the GPUs poll from different locations within, for example, a FPGA. This would require a partitioning of the sector that takes into account the number of stubs and then dispatches the data to differently setup GPUs. This seems unlikely given the latency requirements.

With regard to a possible circumvention of our of shared memory shortcomings, please see Section 15.
12 GPU Benchmarks

12.1 RDMA setup

A Xilinx Virtex-7 XC7VX1140T programmed using the Institute for Data Processing and Electronics (IPE) Firmware is used to transfer the data over the PCIe 3.0 bus to the graphics card. The FPGA is preloaded with data and the Direct Memory Access (DMA) engine is set up optimized for minimum latency transmission. The DMA engine is then started from inside the running kernel. The kernel can check a flag in its global memory to see if data has already arrived. If this is the case, it starts the necessary computations.

12.2 RDMA Benchmark setup

The benchmarks are performed both from inside the GPU and FPGA. The GPU measurements use global variables that take note of the current clock time. The result is translated into actual processing time using the formula

\[ t_{\text{exec}} = \frac{n_{\text{stop}} - n_{\text{start}}}{f} \]  

where \( f \) is the frequency at which the GPU operates. For the Tesla K40 c used in the benchmarks the clock frequency is set 745MHz.
Figure 26: Implemented RDMA data transfer scheme, optimized for maximum throughput, has a higher latency

These measurements were cross checked using measurements from the FPGA. The firmware on the FPGA automatically starts a timer, counting clock cycles. This timer starts as soon as the first data transfer begins. It measurements have an accuracy of around 4 ns, corresponding to the FPGA’s clocking frequency.

The timer is stopped as soon as a certain register gets written. This is done inside from within our running kernel. The timings are cross-checked the timings gathered from the GPU clock cycles to ensure validity. Both ways of measuring time were in good accordance, establishing the clock cycle measurements on the GPU as a valid way to test the performance in terms of the system latency. The gathered timings were used to estimate the latency of the different steps of the calculation. This is useful for identifying the latency bottlenecks.

It is however noteworthy that measuring execution time using GPU clock cycles inherently affects the results of the computation. In order to write the current clock cycle into global memory, it is necessary to halt all other calculations inside the current thread block. Furthermore the usage of global memory causes a relatively high access time penalty. Normally those access times are ‘hidden’ behind computations done by other units, so as not to increase overall execution time. In our case, as we have to stop everything, this hiding of access times can not be performed, altering our benchmarks slightly.
AMD OpenCL

In the case of the Open Compute Language (OpenCL) FPGA GPU data transfer, the transfer times into the GPU seem to be comparable to the results obtained with the NVIDIA card. Unfortunately, for technical reasons beyond our capability to change, it is not currently feasible to use OpenCL for low latency triggering purposes. 

AMD GPUs are currently unable to flush their cache, while the kernel is already running. This makes it impossible to notify the GPU of the arrival of new data after it has started running. As a consequence it would become necessary to relaunch the program running on the GPU after new data has arrived. The latency penalty arising from this is of the order of $60 \mu s$, making it unusable for our low latency purposes. In light of this finding only NVIDIA benchmarks are considered in the following section.

NVIDIA CUDA

In the case of CUDA the gdrcopy library is used. It uses the GPUDirect RDMA Application Programming Interface (API) to provide a mapping of CPU memory to GPU memory.

Instead of mapping CPU host memory, we provide a pinned buffer of the FPGA’s DDR memory to the graphics card. The connection is established over the PCIe bus, with a PCIe switch between GPU and FPGA. Unfortunately, we are not able to directly measure the arrival time of the data inside the GPU memory, but can only pin down the moment when the GPU is aware of the data. It is not clear how the resulting total polling time presented in the following is divided into the actual data transfer and the discovery and copying into the GPU cache memory. This makes it difficult to provide a conclusive figure for the achieved transfer rates.

To quantify the different overheads involved in the data transfer, several measurements have been performed. A CUDA kernel is started as soon as the data transfer is complete, the kernel does nothing but writing back into a register inside the FPGA. The FPGA subsequently stops a counter that started when the DMA transfer began. This provides an estimate of how large the pure overhead from launching a kernel is. And whether there is a difference as compared to non-RDMA setups. The measured overhead is again of the order of 20 to 50 $\mu s$. Making kernel launch overheads prohibitive in the RDMA environment.
spinning kernel pingback

A kernel with only one thread is started, whose sole purpose is to spin in a while loop, that gets resolved as soon as it actually sees the data inside the GPU memory. It then writes back to the FPGA, again stopping the counter that started on DMA transfer start. This gives an overview of the combined time it takes for the data to go through the PCIe bus into the GPU memory combined with the time it takes for the data to be actually available inside the kernel running on the GPU. In accordance to the results found by [23], the latency stays more or less constant at about 2µs, up to transfer sizes of around 8kB. This hints at the GPUs ability of loading the data into the cache, to be the dominant bottleneck in terms of transfer time.
12.3 Transfer Latency

The transfer latency to the GPU is of the order of $2\mu s$, given proper configuration of the DMA engine inside the FPGA. This result is made possible and in accordance to the measurements done carried out previously by [23]. As a next step, we measure the different overheads with respect to transfer and memory access. This includes the DMA overhead, the PCIE! (PCIE!) transfer time combined with the time it takes until the data is ready to be processed, referred to as polling, as well as the time for writing into GPU memory, referred to as transfer.

A spinning Kernel requests data, polls until it discovers the data is arrived and writes into a second memory region. The amount of data corresponds to 160 stubs. After 20 iterations of reading and writing, the GPU writes into the FPGA to stop the FPGA timer. We show mean values for all quantities and give the maximum time of all measurements. The first measurement in each iteration was always discarded as it is dominated by the initial warmup overhead of the DMA engine. The overhead is constant at around 1 s and only present during the first iteration.

In this case, the timing is dominated by the polling step. The polling step also shows the greatest spread in execution time. This makes it likely that it is indeed the GPUs memory discovery that dominates it, as everything coming from the FPGA ought to be more stable. The read write operation on the other hand is very stable in it’s behavior and well below $1 \mu s$. This result is quite good, taking into account the usually necessary times for data transfer with GPUs, and makes further investigations worthwhile.
12.4 Combined Latency

The following results show measurements for performing the actual Hough transformation on a dataset corresponding to 160 stubs. This is a realistic, relatively large number of stubs. The average number produce by the detector at PUs of 140 and with the given sector choice of $32 \times 9$, is around 60. Given the insufficient number of threads, caused by the limitation in terms of shared memory, as describe in Section 11.1 we face the problem that this calculation can not be carried out in parallel with the given amount of stubs, using this specific card. (Although only slightly newer cards could). We could present the results with fewer stubs, few enough to be processed completely in parallel. We decide against this, to simulate the overheads associated with the higher number of stubs and instead give latencies that are valid as soon as there are more threads available.

We show results for both the regular and hexagonal approach. The computation time is relatively stable, at around $3.9 \mu s$, with a standard deviation of around $0.2 \mu s$ and a maximum for the computation time of around $4.1 \mu s$.

The performance of the hexagonal approach is comparable, even though it uses twice the amount of local memory. In this case the calculation happens for two columns in one block, to accomodate the different memory layout of this approach as discussed in Section 10. This showcases the ability of the GPU to perform a more complex task at a small additional cost. In addition to the first two plots, there is another variant of the computation, which we call the interleaved approach. In this case, we do not wait for the data transfer to finish, before starting the computation.

Whereas in the other cases the GPU was idle, while waiting for new data, in this case we skip the polling part almost entirely and directly start the computation. More precisely, we read in one dataset into register memory and after we are finished doing so, directly instruct the FPGA to start transferring the next dataset. This way, since our computation takes more time than the polling, we get rid of the dead time, increasing the computational throughput. This of course has negative consequences for the overall latency. It is not given by the sum of polling and computation, but instead by roughly twice the computation time. It nevertheless shows that we can hide the transfer behind the computation and that the GPU can take care of providing the new memory behind the scenes. Whatever the exact reason for the high polling time was in the first measurements we took, whether there it was some sort of deadlock caused by the polling itself, or something related to the scheduling of the GPU, we should be able to mitigate its effects, provided the current hardware restrictions get resolved and proper work load balancing becomes feasible, see Section 15.
13 Efficiency and fake rates of the algorithms

In this section we discuss the efficiency of the track seeding step, both of the regular as well as the hexagonal approach. We are mainly interested in properties such as tracking efficiency, as well as the amount of produced fake and duplicate tracks. But also in the algorithm’s capability to produce perfect tracking candidates, that is, candidates that have no additional stubs, but the ones belonging to the track’s associated tracking particle.

We present the performance of the hexagonal approach in three different variations. They are distinguished by their choice of binnings, as discussed in Section 10. The version with 23 bins in $\phi$ corresponds to the correct version, meaning all of the allowed parameter space of each stub should be covered. The other two versions represent different cuts in the parameter space. In table 6 and 7 we show the performance for the whole detector, for PU’s of 140 and 200 respectively.

Overall we see the expected efficiency of around 98%, where the results for the HHT$_{23}$ in terms of algorithmic tracking efficiency are slightly better than the ones for the regular HT. The performances are overall comparable throughout, with the expected drop in efficiency for the versions utilizing the cut on the parameter space. The reduced rate and overall number of fake and duplicate candidates comes at the cost of some loss of efficiency. It is however noteworthy that the loss in efficiency is only around half a percent for the 29 bin version where as it is at a catastrophic 2% for the most aggressive 35 bin version.

While having a slightly better tracking efficiency, the HHT$_{23}$ version of the algorithm produces only around 75% of track candidates as compared to the rectangular version in both pile-up scenarios, the high pile-up result is even slightly better by 2%.

The momentum cut approaches provide better suppression of produced candidates and hence discrimination of stubs belonging to the true tracks. The HHT$_{29}$ produces only around 50% of candidates as compared to the HT and still only 65% when compared to the uncut version. This overall trend stays the same even for the PU200 scenario.

The fraction of fake candidates get lower for the more aggressive cuts, again as expected. The version without the cut is acutally slightly worse, than the regular transform in this respect, at least in the sense that the fraction of the candidates is lower. This is not surprising, if we turn to the plot 23 in Section 10. Whereas the array in the parameter space is slightly smaller, the amount of votes per stub is higher for the HHT. This leads to a relatively higher susceptibility to fake candidates, as compared to that of duplicates. This trend vanishes, as we introduce the momentum cut, where the amount
of fakes gets smaller again.

The same reasoning applies to the amount of tracks that are reconstructed perfectly, where the HHT process is superior to the HHT\textsubscript{29} and the performance is about equal as compared to the small cut HHT\textsubscript{29} version. In the high pile-up dataset, the combination of better fake and duplicate suppression allows the small cut version to beat the regular HHT.

In light of the sub-par tracking performance, we will mostly drop the aggressively cut version of the HHT algorithm in the following discussions.
Table 6: Comparison of tracking efficiencies, total number of found track candidates and reconstructed tracks without additional stubs, that do not come from a tracking particle.

**TTbar PU140 Dataset. 3525 Events.** Total tracks valid for algorithmic tracking: 64593

<table>
<thead>
<tr>
<th></th>
<th>HT $32 \times 32$</th>
<th>HHT $32 \times 23$</th>
<th>HHT $32 \times 29$</th>
<th>HHT $32 \times 25$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracks found per event</td>
<td>$354 \pm 2$</td>
<td>$255 \pm 20$</td>
<td>$169 \pm 1$</td>
<td>$123 \pm 2$</td>
</tr>
<tr>
<td>Mean stubs per track</td>
<td>$7,4915$</td>
<td>$7,5089$</td>
<td>$7,3128$</td>
<td>$7,260$</td>
</tr>
<tr>
<td>Fraction fake candidates</td>
<td>$0.5339$</td>
<td>$0.5412$</td>
<td>$0.4615$</td>
<td>$0.3978$</td>
</tr>
<tr>
<td>Fraction duplicate candidates</td>
<td>$0.3305$</td>
<td>$0.2627$</td>
<td>$0.2544$</td>
<td>$0.2233$</td>
</tr>
<tr>
<td>Correctly found candidates</td>
<td>$63140$</td>
<td>$63390$</td>
<td>$62835$</td>
<td>$61610$</td>
</tr>
<tr>
<td>Algorithmic Tracking Efficiency</td>
<td>$0.9775 \pm 0.0006$</td>
<td>$0.9814 \pm 0.0005$</td>
<td>$0.9728 \pm 0.0006$</td>
<td>$0.9547 \pm 0.0020$</td>
</tr>
<tr>
<td>Perfect Tracking Efficiency</td>
<td>$28693$</td>
<td>$26942$</td>
<td>$28996$</td>
<td>$30146$</td>
</tr>
<tr>
<td>Fraction</td>
<td>$0.4442 \pm 0.0020$</td>
<td>$0.4171 \pm 0.0019$</td>
<td>$0.4489 \pm 0.0020$</td>
<td>$0.4670 \pm 0.0020$</td>
</tr>
</tbody>
</table>

14 Efficency as a function of $p_t$

Here we compare the efficiency as a function of pile-up of all discussed approaches.

We see that the overall trend is the same for all approaches and in agreement with the findings in Section [13]. The regular [HT] is again overall slightly worse than the [HHT] and better than the cut approaches. The drop in efficiency, observed at low transverse momenta is explained by the interactions of the TTbar events with the detector material. Overall the performance is equal to what one would expect from the tracker and in accordance to what was found in the carried out simulations [11].

Looking at the region of around $12\text{GeV}$ we see a relative dip in performance, for the tranform without the cut. The dip is present across all datasets, but most prominent in this one. Unfortunately the statistics on the dataset is not very good, as apparent from the relatively large errors. The fact that it is most prominent in the uncut transformation hints at an artefact associated with the choice of the number of bins that have been used.
Table 7: Same as above, but for a higher PU TTbar PU200 Dataset. 2400 Events. Total tracks valid for algorithmic tracking: 44526

<table>
<thead>
<tr>
<th></th>
<th>HT 32 × 32</th>
<th>HHT 32 × 23</th>
<th>HHT 32 × 29</th>
<th>HHT 32 × 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracks found per event</td>
<td>842 ± 7</td>
<td>616 ± 5</td>
<td>363 ± 3</td>
<td>150 ± 15</td>
</tr>
<tr>
<td>mean stubs per track</td>
<td>7.3040</td>
<td>7.3182</td>
<td>7.1680</td>
<td>7.4067</td>
</tr>
<tr>
<td>Fraction of fake candidates</td>
<td>0.7589</td>
<td>0.7662</td>
<td>0.6887</td>
<td>0.4333</td>
</tr>
<tr>
<td>Fraction of duplicate candidates</td>
<td>0.1698</td>
<td>0.1331</td>
<td>0.1488</td>
<td>0.2267</td>
</tr>
<tr>
<td>Correctly found candidates</td>
<td>43418</td>
<td>43574</td>
<td>43187</td>
<td>192/194</td>
</tr>
<tr>
<td>Algorithmic Tracking Efficiency</td>
<td>0.9751 ± 0.0007</td>
<td>0.9786 ± 0.0007</td>
<td>0.9699 ± 0.0008</td>
<td>0.9897 ± 0.0073</td>
</tr>
<tr>
<td>Perfect Tracking Candidates</td>
<td>14499</td>
<td>13228</td>
<td>14982</td>
<td>80/194</td>
</tr>
<tr>
<td>Fraction</td>
<td>0.3256 ± 0.0022</td>
<td>0.2971 ± 0.0022</td>
<td>0.3365 ± 0.0022</td>
<td>0.4124 ± 0.0353</td>
</tr>
</tbody>
</table>

Perhaps even a slight problem with the algorithm itself.

We nevertheless do not drop below the efficiency of the regular approach, but note that further investigations into the matter using better statistics and refined analysis tools to detect the origin of such effects, which are at present not available are advisable.
Figure 28: Comparison of the algorithmic tracking efficiency as a function of the transverse momentum $p_t$. Top Left: HT. Top Right: HHT\textsubscript{23}. Bottom Left: HHT\textsubscript{conservative cut}. Bottom Right: HHT\textsubscript{aggressive cut}. 
15 Discussion

15.0.1 Preferring high \( p_t \) candidates

It is possible that the amount of track candidates given by a specific bunch crossing exceed the amount that is processable while maintaining the aimed for latency requirements. In such a case track candidates will need to be dropped. Due to the at times unwieldy amount of data from the detector this is the case for all approaches currently discussed as solutions.

Therefore it is desirable to ensure that, given the inevitability of losing tracks, the tracks lost are those of least interest. One generally approved on concept is to drop tracks with lower transverse momentum first. This may be easily achieved using a Hough transformation, by first reading out the middle values of the m-axis.

Those are, by construction, associated with the tracks of highest transverse momentum. Given our promising results, in terms of latency the question remains when and if an up-to-scale solution will become feasible and what it will look like. With the hardware used in this work, an unsurmountable amount of GPUs would be needed.

Given the rise of parallel computation power we have seen over the last decade, the ideal solution would minimize the duplication of data coming from the tracker, that intrinsically follows the segmentation of the tracker. Not only does this increase the amount of data needing to be processed and transferred, but it also leads to Track candidates being duplicated, as they might be resident in different sectors at once.

New work by the collaboration of KIT and the UK Track Trigger group, too recent to be part of this work, already acknowledges this idea by splitting the tracker into only two segments in \( \phi \) and 18 segments in \( \eta \), where the duplication is more manageable.

If we look into the development of the maximum number of concurrent threads for NVIDIA Corporation GPUs cards, we see this number doubled in just 3 years. Whereas the Tesla K40c (released in late 2013) used in this work has a maximum of \( \sim 2800 \) number of concurrent threads, the new Pascal generation of cards, coming in clusters of 4 cards offer up to \( \sim 12000 \) threads.

This development means that it should become feasible to process all data coming from one BX within just one or two GPUs. This would lead to a minimum duplication of data, as well as minimize the amount of duplicate track candidates coming from different sectors, but belonging to the same particle.
15.1 Required Throughput

A rough ballpark estimate of the amount of data needing to be transferred, within the latency window of the L1 Trigger System, we can give a minimum number of the memory bandwidth needed to meet these requirements. In this estimate we use 64 bits per stubs. It is sufficient to take only the input rate of data into account, as the outgoing amount is by design roughly a factor of 100 lower and PCIe supports full-duplex transmission, meaning we do not lose bandwidth for writing the triggering decision and the found tracks out, even if we saturate the bus in the ingoing direction. If we further manage to saturate the PCIe bus, a feat by far not yet accomplished in low latency data transfer operation mode.

We are positive that given ongoing technological advances fueled by the industry, future detectors will sooner or later use software triggers. Whether or not the technology will advance fast enough and in the direction needed for the CMS during the HL-LHC phase of operation will hopefully become
clear in the coming years.
16 Conclusion

In this work we present a partial realization of a L1 Track Trigger system, for the silicon tracker of the upgraded CMS detector for its phase 2 run at the High-Luminosity LHC (HLLHC). The computations rely on the Hough transformation, a well established method used in many triggering systems throughout HEP.

We have implemented and benchmarked an analogous version of the Hough transformation, as developed by the collaboration of the KIT and English track trigger group ([4]), as well as our own version of the algorithm, using a hexagonally binned parameter space. While their version runs on FPGAs, our implementation relies on GPUs, using a custom DMA data transfer scheme, based on the work by [23]. The implemented RDMA data transfer scheme, uses an FPGA to transfer the stub data into the GPU for calculating the Hough transformation. The data is compressed to 64 bits per stub, using local coordinates, with respect to the center of the currently analyzed tracker sector. The data transfer is controlled from within the GPU, avoiding the prohibitively large amounts of overhead that normally exclude GPUs from use in low-latency real time environments. The normally present overheads of the order of $50\,\mu s$ for starting programs on the GPU, as well as allocating their cached memory, have been mitigated by our approach. To do this, we deployed a computation scheme that has a continuously running kernel performing the calculations, as well as controlling the RDMA data transfers.

Using these techniques, we were able to achieve latencies surpassing our initial expectations. There is an initial DMA warm-up time, that happens exactly once, at the beginning of the first transfer. All subsequent data transfers, measured from the point of requesting the data to when it is inside the GPU local cache, show a stable transfer time of around $2\,\mu s$. This includes the polling time of the GPU, for when it sees the data inside its global memory as well as the time when it has uncompressed and written the data into its local cache.

The time this process takes up is quite stable showing very little jitter of around $0.02\,\mu s$. While this is not as good as the comparative performance of FPGAs custom interface data transfer, it is nevertheless an excellent achievement given the usually involved latency in regular GPU data transfers and computations. The time spent polling by the GPU stops all calculations and the GPU is idle. To circumvent this gap in throughput, we devised an interlaced operational mode. At an additional cost in latency of the order of the difference between the computation and transfer time, we are able to maximize the time spent computing, given the discussed limitations.

The validation was performed using code from the English track trig-
ger group. A linearized $\chi^2$ fit, analogous to the one used by the tracklet groups approach is performed within the CMSSW framework to evaluate the performance of the Hough transformation in this context.

In addition to the implementation of the regular, rectangular based Hough transformation, we have developed an algorithm of our own. It uses a hexagonally binned parameter space, and provides the ability to perform a cut, reducing duplicate track candidates. The algorithm allows us to perform the binning operation in the more complex space, with comparable computational effort, albeit a higher requirement in terms of memory by about a factor of two. The efficiency of the calculation uses the properties of the underlying helix track model, to perform the rasterization process, minimizing the amount of needed calculations. Compared to the regular transformation, due to the geometric properties of the used parameter space, this version of the algorithm offers better performance with respect to tracking efficiency, as well as suppression of fake and duplicate track candidates.

Even though the algorithm uses the assumed track model to its benefit, its performance does not degrade for particles show greater deviations from the idealized track parameters due to higher interactions with the detector material, e.g. electrons. A modified version of the algorithm applies a cut in parameter space, and further increases the slope of the lines in parameter space. Doing so, it increases the cluster resolution with respect to the transverse momentum even further. It also requires a slightly higher memory, as compared to the uncut variant, by a factor of $\frac{29}{23} \approx 1.26$. Doing so it loses about 0.5% tracking efficiency, but increases the suppression of duplicate track candidates drastically. In addition, the number of stubs associated with each track also slightly decreases. The effects of these properties on the workload of the duplicate removal and subsequently fitting procedure would evidently be beneficial to the performance of the overall track trigger, as the needed filtering, both for duplicate track candidates and stubs not belonging to a track, is a challenging task that usually scales with the possible permutations of the input. Both methods have been validated using simulated event data, corresponding to the expected pile-up conditions present at the LHC for PUs 140 and 200. The results were compared against the expected performance in terms of efficiency and resolution and have been found to the trackers design goals.

A subsequent study on the impact of the applied Hough transformation is beyond the scope of this work. Our work merely presents a possible improvement on the workload on the L1 Track Trigger system. As the algorithmic approach taken is conceptually very close to that of [4], and our results in terms of efficiency are comparable, we expect the subsequent trigger rates for the GT to be essentially unchanged. Summing up, we believe the approach
taken has at least partly demonstrated the flexibility involved with building a possible L1 Track Trigger on GPU. The results in terms of latency are not yet comparable to that of FPGAs, although the achieved performance in this respect, given the initial expectations and usual performance, quite promising.

On the road to a full scale system a lot of open questions remain. We need to establish full transfer back out of the GPU and measure the true total latency. A lot of pending optimizations could conceivably still performed on the algorithms in the future, especially in terms of blocking memory accesses and workload balancing. The latter proves to be a challenge using a GPU for the task at hand. Current GPUs, such as the late 2013 Tesla K40c we used in testing the system, are not yet fit for the task. This applies to the introduced computational latency, the memory access induced latency, as well as the maximum number of concurrent threads preforming the calculation. While this may seem grim at first, recent developments have shown tremendous amounts of progress across all the aforementioned aspects. The same applies for suitable data transfer schemes for GPU-based computing such as nv-link.

Meeting the requirements might become achievable by the start of Phase 2. Even though relying on it might not be the way to go for a project as complex and expensive as the CMS, the added benefits of using more standardized, off-the-shelf components, as well as the superb adaptability involved with using a software based trigger system can hardly be denied.

In general we are quite confident that over the long run, more complex trigger systems will see use at experiments in the not too distant future, even if they have to face latency conditions as challenging as the ones present in the LHC.
Table 8: Comparison of the hardware between two subsequent cards. The Tesla K40 used in this work and its successor, the Tesla K80. The later has two GPU cores. Processor names are given in the table. Data taken from [19]

<table>
<thead>
<tr>
<th></th>
<th>Tesla K80</th>
<th>Tesla K40</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Chip(s)</td>
<td>2x Kepler GK210</td>
<td>Kepler GK110b</td>
</tr>
<tr>
<td>Peak SP (base)</td>
<td>5.60 TFLOPS (comb.)</td>
<td>4.29 TFLOPS</td>
</tr>
<tr>
<td>Peak DP (base)</td>
<td>1.87 TFLOPS (comb.)</td>
<td>1.43 TFLOPS</td>
</tr>
<tr>
<td>Peak SP (boost)</td>
<td>8.73 TFLOPS (comb.)</td>
<td>5.04 TFLOPS</td>
</tr>
<tr>
<td>Peak DP (boost)</td>
<td>2.91 TFLOPS (comb.)</td>
<td>1.68 TFLOPS</td>
</tr>
<tr>
<td>GDDR5 Mem</td>
<td>24GB</td>
<td>12 GB</td>
</tr>
<tr>
<td>Memory Bandwidth1</td>
<td>480 GB/s (comb.)</td>
<td>288 GB/s</td>
</tr>
<tr>
<td>PCI-E Gen.</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>PCI-E transfer BW</td>
<td>12 GB/s</td>
<td>12 GB/s</td>
</tr>
<tr>
<td>SMX Units</td>
<td>26 (comb.)</td>
<td>15</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>4992 (2496 per GPU)</td>
<td>2880</td>
</tr>
<tr>
<td>Memory Clock</td>
<td>2500 MHz</td>
<td>3004 MHz</td>
</tr>
<tr>
<td>GPU Base Clock</td>
<td>560 MHz</td>
<td>745 MHz</td>
</tr>
<tr>
<td>Compute Capability</td>
<td>3.7</td>
<td>3.5</td>
</tr>
<tr>
<td>Workstation Support</td>
<td>â€œYesâ€</td>
<td>Yes</td>
</tr>
<tr>
<td>Server Support</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Wattage (TDP)</td>
<td>300W</td>
<td>235W</td>
</tr>
</tbody>
</table>

17 Appendix
17.1 Seeding Quality
Figure 31: Comparison of the quality of the seed for the production angle $\phi_0$
Top Left: \textcolor{red}{HT}  Top Right: \textcolor{red}{HHT}_{23}
Bottom Left: \textcolor{red}{HHT} conservative cut. Bottom Right: \textcolor{red}{HHT} aggressive cut
Figure 33: Comparison of the quality of the seed in $\frac{4}{pt}$
Top Left: HT  Top Right: HHT$_{23}$
Bottom Left: HHT conservative cut. Bottom Right: HHT aggresive cut
API  Application Programming Interface
HLLHC  High-Luminosity LHC
KIT  Karlsruher Institut für Technologie
DMA  Direct Memory Access
ROC  Read-Out Chip
OpenCL  Open Compute Language
WLCG  Worldwide LHC Computing Grid
PANDA  Anti-Proton ANnihilation DArmstadt
DAS  Data Aggregation System
FAIR  Facility for Antiproton and Ion Research
DRAM  Dynamic Random Acces Memory
SS  Strip Strip
SIMD  Single Instruction Multiple Data
L1TT  Level 1 Track Trigger
AD  Analog Digital
AMD  Advanced Micro Devices
TGC  Triple Gauge Coupling
QGC  Quartic Gauge Coupling
AM  Associative Memory
ASIC  Application-specific integrated circuit
ATLAS  A Toroidal LHC AparatuS
OS  Operating System
ALICE  A Large Ion Collider Experiment
APV  Analogue Pipeline Voltage
LHCb  LHC beauty
LHCb  LHC forward
L1   Level 1
L1A  L1 Accept
BX   Bunch Crossing
CP   Charge Parity
DAQ  Data Acquisition
DT   Drift Tube
CSC  Cathode Strip Chamber
CPU  Central Processing Unit
RPC  Resistive Plate Chamber
TP   Tracking Particle
ECAL Electromagnetic Calorimeter
HCAL Hadronic Calorimeter
HF   Forward Hadronic Calorimeter
HAPS Hybrid Active Pixel Sensor
HLT  High Level Trigger
HHT  Hexagonal Hough Transform
L1T  Level 1 Trigger
NVIDIA NVIDIA Corporation
TMT  Time multiplexed trigger
FPGA Field Programmable Gate Array
GPU  Graphics Processing Unit
GPGPU General Purpose Graphics Processing Unit
CERN European Organization for Nuclear Research
HBM  High Bandwidth Memory
HPC High Performance Computing
HEP High Energy Physics
HL-LHC High Luminosity Large Hadron Collider
HT Hough Transformation
HPD Hybrid Photodiodes
IP Interaction Point
IPE Institute for Data Processing and Electronics
IT In Time
OOT Out Of Time
PS Pixel Strip
SM Standard Model
2S 2 Strip
VME Versa Module Europa (bus)
COTS commercial off-the-shelf component
LARP LHC ACcelerator Research Program
LHC Large Hadron Collider
LS Long Shutdown
LS3 Long Shutdown 3
LUT Lookup Table
MIP Minimum Ionizing Particle
MOS Metal-Oxide-Semiconductor field-effect transistor
GT Global Trigger
MC Monte Carlo
MoEdal Monopole and Exotics Detector At the LHC
PDF Parton Distribution Function
**RDMA**  Remote Direct Memory Access

**SM**  Standard Model

**SUSY**  Super Symmetric

**SSD**  Solid State Drive

**TOTEM**  Total Cross Section

**TP**  Tracking Particle

**KDE**  K Desktop Environment

**CMSSW**  CMS Software Framework

**CMS**  Compact Muon Solenoid

**CUDA**  Compute Unified Device Architecture

**PHBR**  Pre Houghmap Ring Buffer

**CPHRB**  Compressed Pre Houghmap Ring Buffer

**PU**  Pile Up

**PCA**  Principal Component Analysis

**PCIe**  PCIe

**ROOT**  ROOT data framework

**TSV**  Through-Silicon Via

**OpenCL**  Open Compute Language
18 Bibliography

References


[4] C. Amstutz et al. “An FPGA-based track finder for the L1 trigger of the CMS experiment at the high luminosity LHC”. In: 2016 IEEE-NPSS Real Time Conference (RT). Institute of Electrical and Electronics Engineers (IEEE), June 2016. DOI: 10.1109/RTC.2016.7543102 URL: http://dx.doi.org/10.1109/RTC.2016.7543102


Ich versichere hiermit wahrheitsgemäß, diese Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Unterschrift Hannes Mohr