Prepared for submission to JINST

Topical Workshop on Electronics for Particle Physics
September 26TH – 30TH2016
Karlsruhe, Germany

Evaluation of GPUs as a level-1 track trigger for the High-Luminosity LHC

H. Mohr,[1] **T. Dritschler, L. E. Ardila, M. Balzer, M. Caselle, S. Chilingaryan, A. Kopmann, L. Rota, T. Schuh, M. Vogelgesang, M. Weber**

Karlsruhe Institute of Technology,
Hermann-von-Helmholtz-Platz 1, 76344, Eggenstein-Leopoldshafen, Germany

E-mail: h.mohr.hd@googlemail.com

Abstract: In this work, we investigate the use of GPUs as a way of realizing a low-latency, high-throughput track trigger, using CMS as a showcase example. The CMS detector at the Large Hadron Collider (LHC) will undergo a major upgrade after the long shutdown from 2024 to 2026 when it will enter the high luminosity era. During this upgrade, the silicon tracker will have to be completely replaced. In the High Luminosity operation mode, luminosities of $5-7\times10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$ and pileups averaging at 140 events, with a maximum of up to 200 events, will be reached. These changes will require a major update of the triggering system. The demonstrated systems rely on dedicated hardware such as associative memory ASICs and FPGAs. We investigate the use of GPUs as an alternative way of realizing the requirements of the L1 track trigger. To this end we implemeted a Hough transformation track finding step on GPUs and established a low-latency RDMA connection using the PCIe bus. To showcase the benefits of floating point operations, made possible by the use of GPUs, we present a modified algorithm. It uses hexagonal bins for the parameter space and leads to a more truthful representation of the possible track parameters of the individual hits in Hough space. This leads to fewer duplicate candidates and reduces fake track candidates compared to the regular approach. With data-transfer latencies of 2 μs and processing times for the Hough transformation as low as 3.6 μs, we can show that latencies are not as critical as expected. However, computing throughput proves to be challenging due to hardware limitations.

Keywords: Trigger concepts and systems (hardware and software), Trigger algorithms, Computing (architecture, farms, GRID for recording, storage, archiving, and distribution of data), Data processing methods

[1]Corresponding author.

Contents

1 Introduction

1.1 Motivation

The increased luminosity and the large number of pile-up events at the High Luminosity Large Hadron Collider (HL-LHC) makes triggering a major challenge. To address this challenge, CMS is implementing a first-level track trigger. Subsequently, starting with the upgrade during the long shutdown 3, information from the silicon tracker will be used for the first time in the level-1 triggering decision, called the L1 track trigger. The latency will be raised from 3.4 μs to 12 μs, leaving around 4 μs for the L1 track trigger processing. CMS is persuing several approaches based on ASICs and FPGAs to meet the increased trigger requirements.
However, modern Graphics Processing Units (GPUs) have considerably increased in computing performance and interconnect capabilities over the last years. Their widespread availability and easy programming, combined with their high computational complexity, make the use of GPUs as a possible level-1 track trigger platform an option worth considering.

1.2 Graphics processing units

GPUs are massively parallel, multi-core processing units. They consist of streaming multiprocessors that are specialized in performing the same operation on different data. This is called a SIMD (Single Instruction Multiple Data) model. Computational tasks are arranged in so-called thread blocks. Each thread block consists of numerous threads that perform the same operation simultaneously. If there are different logical paths, they have to be computed sequentially. This is called *algorithmic branching*. The concept of a thread block is especially important for GPU scheduling as it allows threads residing in the same block to share data among one another using fast internal memory. There are different memory types. The biggest but also slowest one is *global memory*, which is commonly connected to the GPU's die externally and can be accessed by all threads at all times. The aforementioned fast internal memory is called *shared memory*. It is considerably faster than global memory (by a factor of 100), but also much smaller than global memory, with only some kilobytes of storage. It is only visible to the threads within one block. The fastest form of memory is *register memory*. It is available only to a single thread and only holds a few bytes of data.

In this work, we use a late 2013 Tesla K40c and the CUDA 8 framework, developed by NVIDIA to facilitate general purpose computing on their GPUs [1].

2 The CMS trigger system at the high luminosity LHC

After the upgrade, the Phase-2 CMS detector will have to face luminosities of $5-7\times10^{34}\,\mathrm{cm^{-2}s^{-1}}$ and pileups (*PUs*) averaging at 140 events up to a maximum of 200 events. The trigger system currently used can not handle the resulting data rates. Neither can it discriminate the background from the physics interactions of interest. In Phase-2, data from the silicon tracker will be used in the online triggering procedure for the first time. This Level-1 track trigger will provide the global trigger system with a list of found track candidates, their fitted track parameters and their respective hits. To reduce the data flow from the silicon tracker, an on-detector front-end data reduction scheme will be used. This will help cope with the increased data rates [2]. It will only allow detector hits to pass if they are above a minimum transverse momentum. This process is called stub building [3].

The design goal of the new trigger system is to maintain physics acceptance, while dealing with a trigger acceptance rate 750 kHz, compared to the current rate of 100 kHz. The involved challenges, and thus the need for new advances in both technology and algorithms, are apparent.

Currently three hardware concepts are being discussed. They include the Associative Memory approach that uses pattern banks to find track candidates [4]. The tracklet approach that finds promising pairs of stubs, called tracklets. Combining two stubs in this way improves their momentum resolution, [5] and allows the initial track segments to be extrapolated to neighboring layers, thus facilitating the identification of full tracks [6]. The time-multiplexed track-trigger approach performs a Hough transformation in FPGAs followed by a fitting step, like a linear regression or a Kalman Filter [7]. The track finding procedure discussed below is based on the time-multiplexed track-trigger approach.

3 Hough transformation

The Hough transformation is a global method for detecting image features and has wide applications in both image analysis and particle physics. It scales linearly with the amount of detector hits, since each hit gets transformed into Hough space independently. The underlying assumption for the transformation is a helix track model with the impact parameter d_0 assumed to be zero so that all tracks are coming from the middle of the detector.

The charged particles are bent inside the magnetic field of the detector. They move in a circle of radius R, according to

$$R = \frac{p_\mathrm{t}}{1.14q}. \tag{3.1}$$

Where R is in m, p_t is the transverse momentum in GeV and q the charge of the particle. 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. Furthermore, we ignore energy losses due to interactions of the particles. Following the approach from [7], we carry out the transformation with respect to a pivot point $r' = r - r_\mathrm{c}$, where $r_\mathrm{c} = 65\mathrm{cm}$. This optimizes the parameter space. Furthermore a shift in ϕ is introduced as $\phi' = \phi - \phi_\mathrm{c}$, enabling us to carry out the calculation in independent ϕ-sectors. This leads to

$$\phi'_0 = \phi' - \frac{0.57q}{p_t} r', \tag{3.2}$$

where ϕ'_0 is the tracks production angle. The set of possible track parameters for a given detector hit is thus represented by a line in the $r - \phi$ parameter space. The boundaries of the parameter space are fixed. The allowed momentum range is given by the front-end momentum cut, whereas the production angle range is given by the size of the current ϕ-sector. Lines from different detector hits cross and form clustering points. Those clusters correspond to track candidates. We accept a cluster as a track candidate if consists of at least 5 hits from different detector layers.

3.1 Hexagonal Hough transform

We developed an algorithm that makes use of the GPUs floating point capabilities. It divides the parameter space into regular hexagons. We will describe the basic idea behind the algorithm and point out some of its benefits.

The maximum slope $m_{\max}$ for a track in parameter space is realized when r' is maximized. $r'_{\max}$ is determined by the detector geometry. Since the parameter space boundaries are fixed, the effective maximum slope is given by the amount of bins we choose. We can guarantee to allow only one hexagon per row, where hexagon centers are atop each other by demanding

$$m_{\max} < \frac{h}{w} \approx 1.15. \tag{3.3}$$

Here $h = 2R$ is the hexagons height, and R the outer radius. Its width w corresponds to twice the inner radius. If we arbitrarily choose $w = 1$, then $h = \frac{2}{\sqrt{3}}$. If (3.3) is fulfilled, it suffices to calculate the ϕ-axis value, according to (3.2), once in the hexagon's center (with respect to the momentum axis) for each row. The valid bin is either directly known, because we are already inside a given hexagon, or found by linear extrapolation. The linear propagator δh is based on the ratio of the current and maximum slope. It is given by

$$\delta h = 0.5 \frac{r'}{r'_{\max}} s. \tag{3.4}$$

This way we define thresholds for making it to the upper and lower hexagon. The principle is illustrated in figure1.

3.2 Correctness and implications on the parameter space

Using 32 bins in $\frac{q}{p_t}$ we find the number of bins in ϕ_c that guarantees (3.3) to be 23. The method assumes that we get arbitrarily close to the value. This overestimates the allowed parameter space and thus allows for an uncertainty in the measured quantities. Effectively we allow lines that barely miss a given cell to still contribute to the cell. If we use more bins, and allow for higher values of the slope, we achieve the opposite and apply a cut in the parameter space. The latter reduces the amount of valid cells around the clustering points with effects on both efficiency and fake duplicate rates. The results of the two approaches are discussed in section 3.3.1.

3.3 Benefits

The algorithm uses the geometric properties of the hexagons, the underlying helix track model and the properties of the tracker to its advantage.

The layout of the grid allows for a maximum of three cells per cluster, as compared to the four cells using rectangles. The neighborhood relations in a hexagonal grid are equidistant and well-defined. We only have to check six neighboring cells as compared to the eight potential neighbors of a rectangle. The greater slope (1.15 as compared to 1) decreases the width of clusters in the momentum axis, leading to fewer duplicate track candidates, as each cluster covers fewer bins. On the other hand the clusters are more spread out in the production angle axis, which would normally lead to more votes. This effect gets mitigated by the alternating layout of the hexagonal grid. The reduced number of fake track candidates leads to a decrease in the workload of the duplicate removal. The overall higher amount of bins — approximately a factor of two — leads to more and more densely packed seed values for the track parameters. This is due to the hexagon being the two-dimensional polygon that segments the plane most effectively. Generally, the distance to a cell's center is minimized while the area of each cell is smaller. This leads to a finer resolution of the

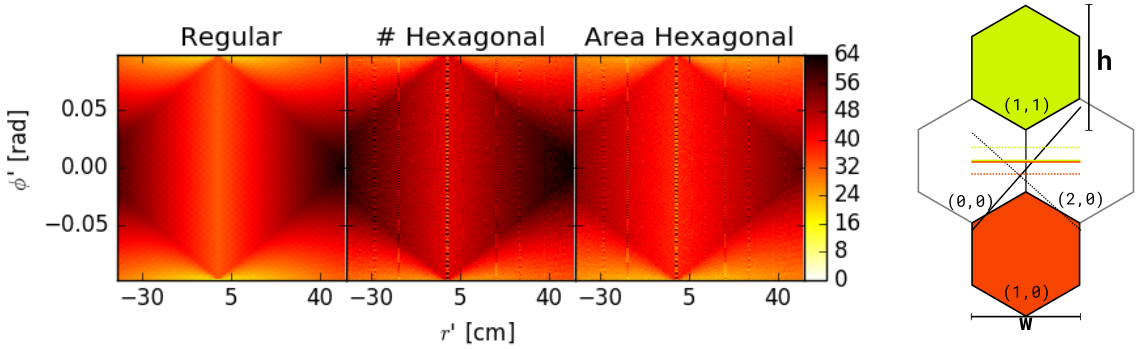

Figure 1. Comparison of the regular and hexagonal parameter spaces coverage with respect to the regular approach (left). The histograms give the number of cells needed to represent a line in the parameter space. The histograms cover all of the allowed input values of r' and ϕ' for the regular approach, hexagonal approach and for the hexagonal transformation, weighted by the relative area with respect to the regular approach. Graphical representation of the binning procedure (right). This shows the procedure for determining whether a line enters either the upper or lower hexagon inside the current row. Both lines will be counted in the lower, red hexagon because they are both beneath their threshold, also shown in red. The solid line has maximum slope. Its thresholds (also solid) coincide at the center. The dashed line would miss both hexagons, if it was between its thresholds (dashed).

allowed track parameters. In figure 1 a 2D histogram of the needed number of cells to represent a line is shown. The axes of the histogram range over all allowed input values for r' and ϕ'. This value is higher for the hexagonal approach shown in the center of the graph, as compared to the regular one shown on the left. The finer rasterization of the line leads to a more truthful representation of the allowed track parameters. The plot on the right is the number of cells used for each possible line, but weighted with relative size of each cell in parameter space. This illustrates that we use more individual cells but in many cases cover less of the parameter space. Figure 2 shows the filled Hough maps for the regular and hexagonal approach for a qualitative comparison. It is noticeable that the maxima in the hexagonal case are well defined, whereas in the regular one they spread out over two neighboring cells in both cases.

From a computational point of view the implementation on a GPU benefits from the property of needing only one cell per momentum value, to avoid algorithmic branching. While the needed memory increases by a factor of about two, the number of calculations stays the same, as we do not have to check for a second possible production angle value as is the case for the regular grid.

3.3.1 Tracking efficiency and rates of duplicate and fake candidates

To evaluate the algorithms performance with respect to the track finding efficiency, a TTBar dataset was analyzed, for both pile up scenarios. As is to be expected, the overall track finding performance of the hexagonal approach is similar to that of the normal version of the algorithm. For an in-depth discussion see [7]. The efficiency is slightly higher, by around 0.35% for the uncut hexagonal version. The amount of produced tracks drops by around 33%. The fewer number of found track

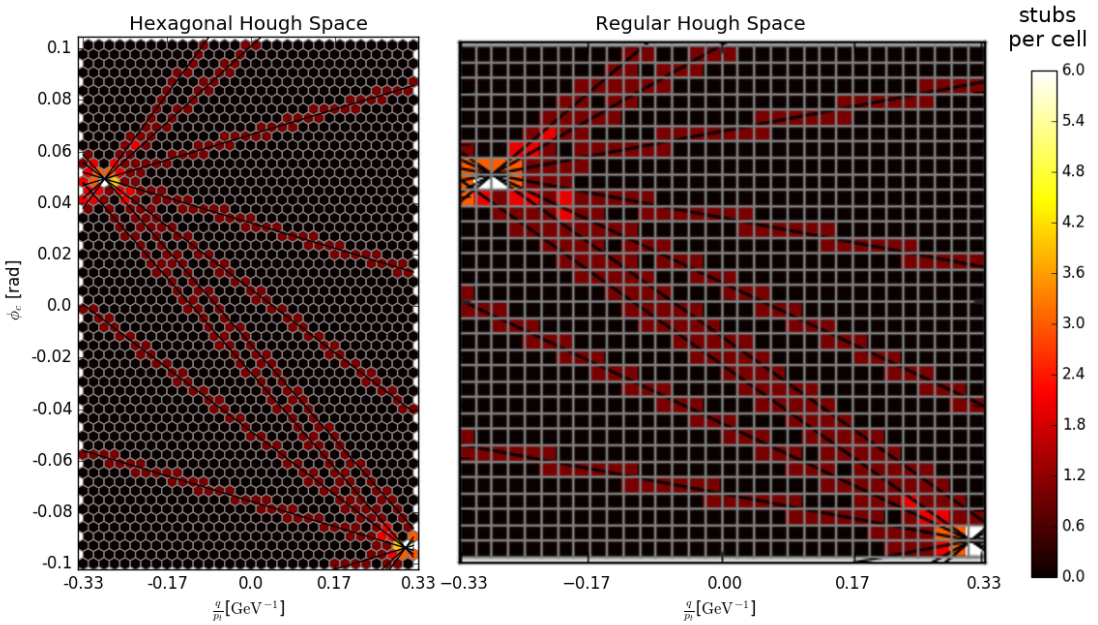

Figure 2. Comparison of the regular and hexagonal binning behavior. The shown hexagonal transformation (left) has 29 bins in ϕ_0, corresponding to the momentum cut version. Nevertheless, the binned representation of the line is without gaps. For the regular transformation (right) the width of the clusters is bigger, not just in the index space, but also in the parameter space.

candidates manifests itself mainly in a reduction of duplicates. We attribute this behavior to the aforementioned properties. The slightly higher efficiency is due to the allowed uncertainty in the input parameters as discussed in section 3.2. As for the parameter space cut approach using 29 bins, we find a drop in efficiency of about 0.5% compared to the rectangular approach. The number of found track candidates drops in this case by around 55%. This is to be expected due to the nature of the approach. The amount of fake candidates drops significantly as compared to the uncut version, which makes this an interesting option for reducing the overall workload on the system.

4 Implementation of the GPU track trigger

We use an FPGA-based data transfer infrastructure based on the Direct Memory Access (DMA) architecture described in [8]. The FPGA gets loaded with our test-data in advance and is programmed with custom DMA firmware that enables low latency data transfer from the FPGA directly into the GPU's memory [9, 10]. The data is assumed to be compressed, making use of the detector's finite resolution. Each stub is compressed to a size of 64 bits. We assume a detector segmentation of $32\,\phi \times 9\,\eta$. The stub coordinates are transmitted in accordance to the local coordinate system of the detector segment in which they were detectd.

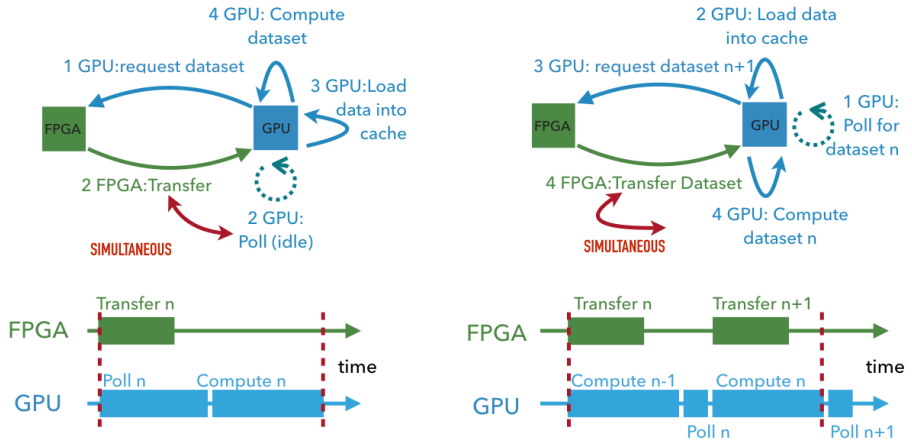

Figure 3. Comparison of the regular data transfer scheme, optimized for low latency (left) and the pipelined data transfer scheme, optimized for maximum throughput (right).

4.1 RDMA interconnect and low latency requirements

Traditionally the usage of GPUs introduces a latency penalty that would render them unusable for low latency applications. The overheads associated with traditional memory transactions are typically of the order of tens of microseconds. Additionally, there are overheads involved with starting kernels, due to the allocation of shared memory as well as the scheduling on both the host and device side.

Our setup uses a NVIDIA Tesla K40c (late 2013) GPU for the computations and a Xilinx Virtex-7 XC7VX1140T FPGA as a data source. The interconnect is established using an RDMA data transfer scheme. All memory transactions are initialized from within the GPU program (*kernel*). To avoid kernel launch overheads, our kernel runs in a continuous loop. In each iteration, it initializes data transfer, waits for data transfer completion and then performs the Hough transformation. Together with the aforementioned RDMA transfer scheme, this mitigates most of the involved overheads. It also brings with it some restrictions concerning workload balancing (section 4.2) that will need to be addressed in future work.

4.2 Implementation details

The filling of the parameter space histogram used in the Hough transformation is currently done with one thread block per momentum bin. This means we use a total of $n_{\text{bins}}^{p_t}$ thread blocks, each with a number of threads corresponding to the maximum allowed number of stubs. This is a necessity, given the current implementation and hardware restrictions.

Calculating a histogram on a GPU can not be performed completely in parallel. To update a counter inside a given histogram bin, the current value has to be read from memory, updated and then written back. This leads to *race conditions* if two or more execution units write into the same bin at the same time, which needs to be prevented. The CUDA framework offers so called *atomic operations* which guarantee read-modify-write operations without interference from other threads.

This has a negative effect on the throughput, as it forces other concurrent threads to wait while memory is being modified, which makes atomic operations costly. To reduce the impact of the enforced waiting times, we limit atomic operations to shared memory which is significantly faster than global memory. The needed atomic operations can only operate on regions of 32 or 64 bits in size. The used memory consists of a counter for each bin, a bit mask keeping track of the hit layers, and the list of stub indices belonging to the cell. All of the above take up 32 bits in size each. The needed shared memory is thus 130 kilobytes for the regular transformation and 260 kilobytes for the hexagonal one. This exceeds the amount of available shared memory per thread block for the given card, which only provides a maximum of 48 kilobytes per block. Because of this limitation, we are forced to spread our calculation across multiple thread-blocks. This procedure also increases the amount of concurrently used threads by a factor corresponding to the number of used thread blocks. In our current implementation, we chose to spread the calculation across 32 blocks. This has a positive effect on the systems latency, but a negative impact on its throughput. There are ways around these problems, however, they are beyond the scope of the current proof-of-concept minimum latency implementation.

5 Benchmarks

To assess GPUs as a possible track trigger we benchmark the data transfer and computational performance in terms of their latency. The timings were measured both on the GPU and the FPGA. Inside the running kernel we measure the time at certain points using the GPU clock cycles. Inside the FPGA there is a hardware timer that counts the FPGA clock cycles with a resolution of 4 ns. The former allows us to perform fine grained measurements of the individual steps, while the latter assures correctness of the results. When the transfer is initiated, the FPGA writes data into the GPU memory. At the same time the GPU starts polling for the arrival of this data by continuously checking a specific memory region for a special completion-marker, which the FPGA will write to this location once the data transfer is complete. Afterwards, the GPU immediately starts to process the received data. We call the combined time for the data transfer and the time it takes the kernel to notice the completion-marker the *polling time* (table 1). The total computation time includes the transfer from global to shared memory, the decompression of the data, the filling of the histogram as well as the filtering step based on our threshold condition.

The polling step performed by the GPU is done by only one single thread, halting all other operations for that time. Therefore, most of the GPUs computing resources remain idle, while the GPU waits for the data transfer to finish. For this reason, we implemented a second variant that maximizes throughput at the cost of some latency (figure 3). In this case the GPU takes the transmitted data from global memory and copies it into shared memory. It then immediately requests new data to be written to global memory. While this new data is being transmitted to global memory, the GPU simultaneously starts to process the data that was copied to shared memory. This way, we hide the transfer time behind the computations. We call this the *pipelined* approach. Except for the modified ordering of data transfer and computation, it is otherwise identical to the normal approach.

The data used in our benchmarks is from a TTBar PU140 data set. The data transmission time shows a very stable behavior and takes on the order of 2 μs for 1280 bytes of data. The computation time exhibits larger spread and dominates the total time taken for both algorithms. It is at around

3.7 μs for the normal Hough transform and around 4.9 μs for the hexagon version. Checking for newly arrived data still takes around 0.4 μs. This is due to the relatively large penalty that comes with accessing global memory, which is on the order of 300 GPU clock cycles.

It should be noted that the currently used card does not have sufficient concurrent threads to perform these calculations for a larger number of stubs in parallel. We have decided to allow for 160 stubs per sector in these benchmarks, even though that surpasses the amount of available threads by about a factor of two. This allows a measurement to be given where the effects of high occupancy are accounted for.

Table 1. Execution times for polling and computation for both algorithms with both measured setups. All times given in μs.

Sequential	mean	σ	max	Pipelined	mean	σ	max
Regular							
Polling and Transfer	1.96	±0.02	2.01	Polling	0.41	±0.01	0.43
Computation	3.73	±0.13	4.19	Computation	3.63	±0.10	3.82
Total	5.84	±0.14	6.33	Total	4.17	±0.07	4.37
Hexagonal							
Polling and Transfer	1.91	±0.11	1.99				
Computation	4.94	±0.11	5.20				
Total	7.07	±0.12	7.40				

6 Conclusion

Our results show that a latency of a few μs is in fact achievable using a GPU for track finding purposes. The requirements for achieving this sort of result come at the cost of flexibility, especially in terms of load balancing. The work group dimensions need to be fixed prior to performing any calculations. This currently leads to suboptimal behavior in terms of achievable throughput. Overall, the performance in terms of latency is very promising, whereas the throughput is not yet competitive. Using the tested generation of cards, we would have to use an unfeasibly large amount of GPUs to process all the detector data. Currently, due to restrictions given by the available shared memory, as well as some limitations in the memory layout of the current implementation, we are unable to process more than one sector per GPU. However, we are confident that newer generation cards, as well as a more refined memory layout, will allow us to achieve comparable execution times for a larger number of sectors in the future. In addition, technological advances in GPU interconnects, like NVIDIA's NVLink technology [11], might further improve transfer latency, resulting in overall better turnaround times.

The newly developed algorithm, using floating point calculations, produces better results in terms of duplicate candidates, as well as, given the discussed momentum cut, better results in terms of fake track candidates. This should help reduce the overall workload on both the duplicate removal as well as the subsequent fitting step in the track trigger. In terms of computation time it shows a comparable behavior to the regular approach when implemented on a GPU.

References

[1] CUDA Nvidia. Programming guide, 2008.

[2] G Boudoul. A level-1 tracking trigger for the cms upgrade using stacked silicon strip detectors and advanced pattern technologies. *Journal of Instrumentation*, 8(01):C01024, 2013.

[3] D Contardo, M Klute, J Mans, L Silvestris, and J Butler. Technical Proposal for the Phase-II Upgrade of the CMS Detector. Technical Report CERN-LHCC-2015-010. LHCC-P-008. CMS-TDR-15-02, Geneva, Jun 2015. Upgrade Project Leader Deputies: Lucia Silvestris (INFN-Bari), Jeremy Mans (University of Minnesota) Additional contacts: Lucia.Silvestris@cern.ch, Jeremy.Mans@cern.ch.

[4] D. Sabes. L1 track triggering with associative memory for the cms hl-lhc tracker. *Journal of Instrumentation*, 9(11):C11014, 2014.

[5] E Salvati. A level-1 track trigger for cms with double stack detectors and long barrel approach. *Journal of Instrumentation*, 7(08):C08005, 2012.

[6] L. Skinnari. L1 track triggering at cms for high luminosity lhc. *Journal of Instrumentation*, 9(10):C10035, 2014.

[7] C. Amstutz, F. A. Ball, M. N. Balzer, J. Brooke, L. Calligaris, D. Cieri, E. J. Clement, G. Hall, T. R. Harbaum, K. Harder, P. R. Hobson, G. M. Iles, T. James, K. Manolopoulos, T. Matsushita, A. D. Morton, D. Newbold, S. Paramesvaran, M. Pesaresi, I. D. Reid, A. W. Rose, O. Sander, T. Schuh, C. Shepherd-Themistocleous, A. Shtipliyski, S. P. Summers, A. Tapper, I. Tomalin, K. Uchida, P. Vichoudis, and M. Weber. 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), jun 2016.

[8] L. Rota, M. Caselle, S. Chilingaryan, A. Kopmann, and M. Weber. A pcie dma architecture for multi-gigabyte per second data transmission. *IEEE Transactions on Nuclear Science*, 62(3):972–976, June 2015.

[9] L. Rota, M. Vogelgesang, L.E. Ardila Perez, M. Caselle, S. Chilingaryan, T. Dritschler, N. Zilio, A. Kopmann, M. Balzer, and M. Weber. A high-throughput readout architecture based on pci-express gen3 and directgma technology. *Journal of Instrumentation*, 11(02):P02007, 2016.

[10] M. Caselle, L.E. Ardila Perez, M. Balzer, S. Chilingaryan, T. Dritschler, A. Kopmann, H. Mohr, L. Rota, M. Vogelgesang, and M. Weber. A high-speed daq framework for future high-level trigger and event building clusters. *Journal of Instrumentation*, 2016.

[11] Denis Foley. Nvlink, pascal and stacked memory: Feeding the appetite for big data. *Nvidia. com*, 2014.