A Parallel-Computing Algorithm for High-Energy Physics Particle Tracking and Decoding Using GPU Architectures

Placido Fernandez Declara, Daniel Hugo Campora Perez, Dorothea Vom Bruch, Niko Neufeld, Javier Garcia-Blas, J. Daniel Garcia

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 26, 2019, accepted June 27, 2019, date of publication July 8, 2019, date of current version July 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2927261

A Parallel-Computing Algorithm for High-Energy Physics Particle Tracking and Decoding Using GPU Architectures

PLACIDO FERNANDEZ DECLARA[1,2], DANIEL HUGO CÁMPORA PÉREZ[1,3], JAVIER GARCIA-BLAS[2], DOROTHEA VOM BRUCH[4], J. DANIEL GARCÍA[2], AND NIKO NEUFELD[1]

[1]EP-LBC, CERN, 1211 Geneva, Switzerland
[2]Department of Computer Science and Engineering, University Carlos III of Madrid, 28911 Madrid, Spain
[3]ETSI Informática, Universidad de Sevilla, 41012 Sevilla, Spain
[4]LPNHE, Sorbonne Université, Paris Diderot Sorbonne Paris Cité, CNRS/IN2P3, 75005 Paris, France

Corresponding author: Placido Fernandez Declara (placido.fernandez@cern.ch)

This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Programme under Grant 724777 "RECEPT," and in part by the Spanish MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD though Project Grant TIN2016-79637-P TOWARDS UNIFICATION OF HPC AND BIG DATA PARADIGMS.

ABSTRACT Real-time data processing is one of the central processes of particle physics experiments which require large computing resources. The LHCb (Large Hadron Collider beauty) experiment will be upgraded to cope with a particle bunch collision rate of 30 million times per second, producing 10^9 particles/s. 40 Tbits/s need to be processed in real-time to make filtering decisions to store data. This poses a computing challenge that requires exploration of modern hardware and software solutions. We present *Compass*, a particle tracking algorithm and a parallel raw input decoding optimized for GPUs. It is designed for highly parallel architectures, data-oriented, and optimized for fast and localized data access. Our algorithm is configurable, and we explore the trade-off in computing and physics performance of various configurations. A CPU implementation that delivers the same physics performance as our GPU implementation is presented. We discuss the achieved physics performance and validate it with Monte Carlo simulated data. We show a computing performance analysis comparing consumer and server-grade GPUs, and a CPU. We show the feasibility of using a full GPU decoding and particle tracking algorithm for high-throughput particle trajectories reconstruction, where our algorithm improves the throughput up to 7.4× compared to the LHCb baseline.

INDEX TERMS CUDA, GPGPU, track reconstruction, particle tracking, parallel programming.

I. INTRODUCTION

High-energy physics experiments produce large data streams that must be processed, filtered, and analyzed. The LHCb (Large Hadron Collider beauty) experiment is one of the four big physics detector experiments collecting data at the Large Hadron Collider (LHC). LHCb aims to explore the matter-antimatter asymmetry problem [1]. It is being upgraded and expected to restart operation in 2021; producing data at a rate of 40 $Tbit/s$ [2]. Its event[1] filter will be run solely on general purpose computing resources, also known as software filter, where the LHCb data analysis framework has to process data in real-time, and decide which collision events may be discarded and which must be kept for further analysis. The software based event filter must be modernized to be able to handle the increased throughput [3], [4].

The LHCb experiment will have to increase its compute power needs to handle the continuous deluge of data from the detector. The big cost of the necessary increase in computer power lead to the exploration of alternative hardware architectures. As heterogeneous data centers comprised with multi- and many-core CPUs and coprocessors/accelerators emerge, LHCb and other CERN experiments are currently considering different hardware options to reach the aforementioned performance goals for the coming years. The current LHCb computing farm consists of servers based on the x86-64 architecture. However, alternative architectures

The associate editor coordinating the review of this manuscript and approving it for publication was Jihad Aljaam.

[1]A collision event corresponds to the crossing of two bunches of protons in the LHC beams.

and accelerators are being tested in different trigger systems [5]–[7]. This is an indication that systems requiring high-throughput can be met in such alternative architectures.

LHCb computing farm needs to treat 30 million events per second, producing around 10^9 particles per second. Reconstructing particle trajectories, known as particle tracking (from here on shortly referred to as "tracking"), plays a central role in processing these events. Introducing an architectural change, poses multiple challenges in terms of software to perform particle tracking in real-time. Existing algorithms must be redesigned to fully exploit parallel architectures. Furthermore, the expected long life cycle of these algorithms demands not only a high degree of performance optimization but also maintainability and portability. Those goals are ubiquitous in the scientific and engineering software areas and different solutions have been proposed. Among these, GPU-based approaches have been a successful alternative in providing high-throughput in different scenarios [8]–[10]. This paper presents the implementation of a data-oriented approach, focusing on creating algorithms for SIMD (Single Instruction Multiple Data) architectures, minimizing thread divergence, reducing data movements and memory footprint of the algorithm, which have been successful strategies to optimize algorithms for GPUs [11], [12]. We run as part of the LHCb GPU sequence framework defined in [13], which allows multiple concurrent GPU stream execution.

The main contributions of this paper are as follows:

a) We present a fast tracking algorithm for high-energy physics detectors targeting SIMD architectures called *Compass*. The proposed algorithm can deal with deviated particle trajectories by a magnetic field.
b) We introduce a parallel version for the decoding of the raw input data, which ensures coalesced data write patterns and produces a sorted SoA data structure, beneficial to our tracking algorithm.
c) We investigate the impact of our algorithm configuration on the physics quality of the results and analyze its computing performance on a variety of GPUs and CPUs.

The rest of this paper is organized as follows. Section II explores the state-of-the-art on high-throughput, real-time, and scientific usage of GPUs. Section III briefly introduces the concepts used in high-energy physics for tracking, specifically for the LHCb experiment and UT tracking. On Section IV, the implementation of the decoding of the raw input data is explained, whereas in Section V the main algorithm design and implementation are presented. Section VI shows the experimental evaluation carried out and presents the obtained physics efficiency. Finally Section VII closes the paper with concluding remarks and future research lines.

II. RELATED WORK

We focus on high-throughput computing fields that process large scientific datasets and have similarities to those encountered in track reconstruction algorithms, this is, they process numerous small units of work. We discuss real-time approaches and other scientific applications which need to deliver high-throughput.

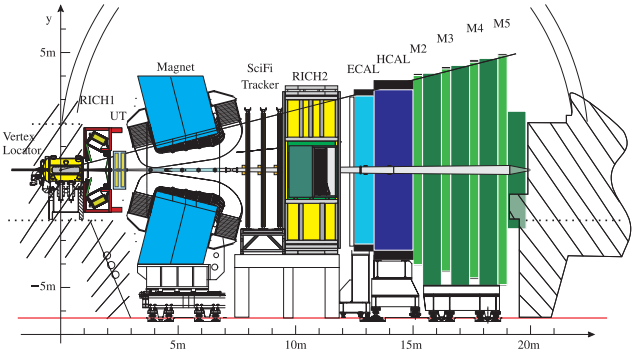

FIGURE 1. Schematic view of the LHCb upgrade detector.

GPUs have been used before in the field of high-energy physics with success. The ALICE experiment at CERN implemented track reconstruction in GPUs obtaining different speedups compared to the previously used hardware [14]. We note how the approach we follow is different than the one implemented in ALICE, as we aim to implement the full *High Level Trigger* to run in GPUs, including the decoding and tracking of all subdetectors, thus avoiding much of the needed data transmission between main memory and GPU memory. Other HEP experiments have seen significant improvements when using GPUs to amend the performance of online selection [7], [15], or using a common code base to target both CPUs and GPUs using OpenCL, which shows the performance improvement of GPUs while supporting the x86-64 architecture [16].

The performance of DNA sequencing problems has been improved with GPUs in different high-throughput scenarios. The *Arioc* read aligner showed how using parallel algorithms with GPUs improved DNA sequencing throughput, achieving an order of magnitude faster alignments [17], [18]. Pawar *et al.* benchmarked various DNA sequencing algorithms with different GPU-based tools against a CPU one; concluding that GPUs will replace CPUs in DNA sequencing for its higher-throughput processing [19]. Other DNA-related fields exhibit similar speedups: Samsi *et al.* [20] demonstrated how a single GPU is able to compare millions of DNA samples in seconds, Cadenelli *et al.* [21] compared offloading a genomics workload into FPGAs and GPUs from a CPU, resulting in the GPU outperforming both, although the GPU consuming more energy.

Other scientific fields benefit from high-throughput, real-time processing in GPUs. Radio telescopes need to filter data in their data acquisition systems; where software frameworks employing GPUs like *Bifrost* [22] have shown significant performance improvements. Other real-time radio telescope experiments studied the viability of using GPUs, where they encountered large computing speedups at a local level, but were limited by I/O when using multiple GPUs [23]. Others in the same field have successfully implemented GPU optimization schemes [24] achieving a 6× speedup compared

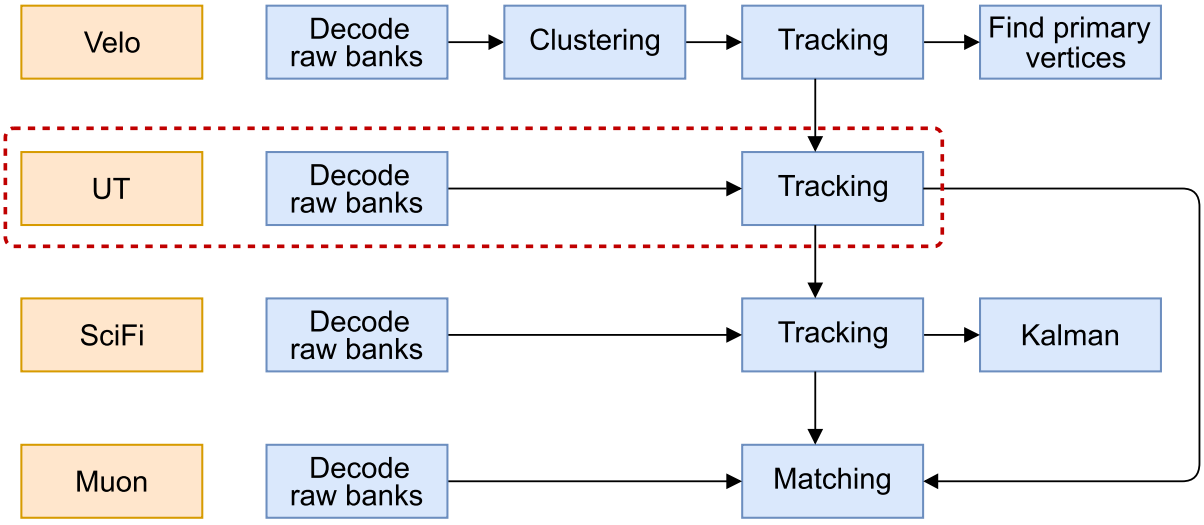

FIGURE 2. Complete High Level Trigger 1 sequence of algorithms at LHCb. We highlight the UT algorithms described in this paper (dotted lines). UT is the second tracking sub-detector in the chain of algorithms, and it receives input from the UT raw banks and the VELO tracks. UT outputs reconstructed tracks for other sub-detectors.

to the CPU scenario, or used a GPU-based software framework and aggressive optimizations to be able to process data rates close to 1Tbit/s, like the *CHIME Pathfinder* radio telescope [25].

GPUs have also been studied in scenarios requiring real-time processing at fusion experiments [26] greatly reducing the wall-time compared to the CPU version. Real-time split-and-merge executions have been improved in multi-GPU scenarios by Han *et al.* [27], and X-ray computer tomography reconstruction in GPUs has shown how different optimizations can be implemented and combined to speedup GPU computations [28].

Our approach for using GPUs in high-energy physics presents a parallel tracking algorithm which reconstruct particle trajectories that are bent under the influence of a magnet, describing a non-straight trajectory. We focus on achieving high-throughput to meet the collision rate and real-time constraints of the LHC at CERN. Other scientific fields have been successful on implementing real-time high-throughput solutions with GPUs, where fields like DNA sequencing are already ditching CPU-based architectures to process their large datasets. Successful results in the HEP fields suggest that implementing a full filter with GPUs, including the decoding and tracking of charged particles, is a feasible task that will increase the filtering throughput capabilities of LHCb.

III. BACKGROUND

In Figure 2 we depict the full chain of algorithms needed to run the High Level Trigger 1 at LHCb required to filter events. In this section we describe the UT (Upstream Tracker) sub-detector, which provides part of the input data needed for the tracking algorithm. UT algorithms are second in the chain, receiving input from the UT raw banks and the reconstructed tracks from the VELO (Vertex Locator). This paper covers all the UT steps highlighted in Figure 2: the decoding of UT raw banks, and the UT tracking. [2]

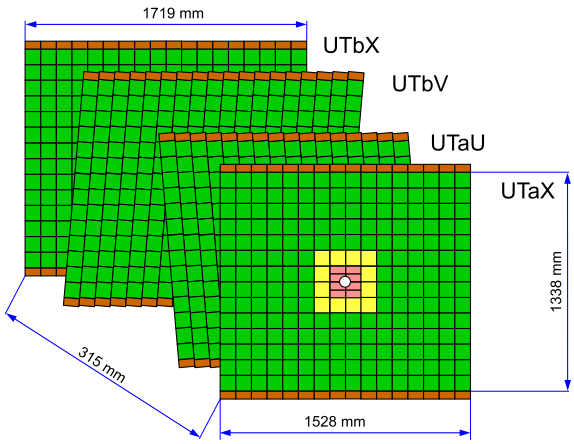

FIGURE 3. UT planes. The four UT planes are presented in this figure. The plane in the front (UTaX) is the closest to the VELO. Planes have a height of around 1.3 m, where the width is determined by the plane and changes between roughly 1.5 to 1.7 m. Different colors indicate the different types of sensors which accommodate different number of strips. The sensors around the center have higher resolution. This design follows the simulation data, which indicates higher number of particles around the beam in the center.

A. UT SUB-DETECTOR

The LHCb detector is composed of various sub-detectors, as shown in Figure 1. In order to reconstruct particle trajectories, information from various sub-detectors is required. The sub-detectors that provide tracking information are the VELO, the UT, the SciFi Tracker and the μ (Muon) tracker. The UT is located in between the VELO and the SciFi Tracker [4].

[2]UT decoding and *Compass* algorithms are available at https://gitlab.cern.ch/lhcb-parallelization/Allen

The UT sub-detector is composed of four planes, where each plane is a single sided silicon strip detector. We refer to the four consecutive planes as UTaX, UTaU, UTbV, UTbX respectively, as can be seen in Figure 3. These are sorted into two layers containing 2 planes each, the *a* and *b* layers. The *X* planes are composed of vertical strips whereas the *U* and *V* planes are tilted around the Z axis at $+5°$ and $-5°$ respectively. By combining the measurements from the tilted *U* and *V* planes, the Y coordinate can also be determined. Each UT plane is composed of micro-strip sensors arranged in vertical staves [29]. A UT plane can be divided into 3 regions with different geometry, where the inner-most region has a finer granularity, and the outer regions have coarser granularity. Each stave measures 160 cm high and 10 cm wide, where various sensors are placed alongside each stave. The sensors in a stave overlap with their neighbor sensors, to avoid gaps, and the vertical staves also overlap for the same reason. The *X* planes are composed of 16 staves while the *U* and *V* are composed of 18 staves. The acceptance of the UT sub-detector is defined by its volume in space, the UT planes for the UT sub-detector. Only particles that traverse this volume can leave signals and are measured.

The UT detector serves various purposes in the LHCb experiment: noitemsep

- Reconstructs charged particles trajectories that decay after the VELO sub-detector.
- Reconstructs low momentum particles that are bent by the magnet, and go out of acceptance before reaching the SciFi Tracker.
- Gives additional information in the form of hits, that can be used in conjunction with the VELO and SciFi Tracker information to reject tracks.
- As the UT is influenced by the magnet, it can provide momentum resolution for charged particles.
- It can reject low momentum tracks.
- Decreases time to extrapolate VELO tracks to SciFi Tracker by at least a factor of 3.

Finally, UT plays an important role by marking tracks that won't be used by the next tracking detector, the SciFi Tracker. This allows for a faster processing of the whole track reconstruction in the LHCb detector.

B. TRACK TYPES, EFFICIENCY, AND FAKE RATES

When performing particle tracking in the LHCb detector, tracks are classified according to the sub-detectors they traversed.

The tracks that traverse the UT sub-detector or serve as input for it are classified as follows:

- *Long tracks*: contains hits detected from the VELO to SciFi Trackers, and they may contain hits in the UT. Long tracks analyzed here have hits in the UT.
- *Upstream tracks*: comprise hits recorded in VELO and UT detectors, but not in SciFi Tracker. These tracks are bent by the magnetic field, so they travel outside the SciFi tracker, without crossing it. We refer to them as VELO+UT tracks.

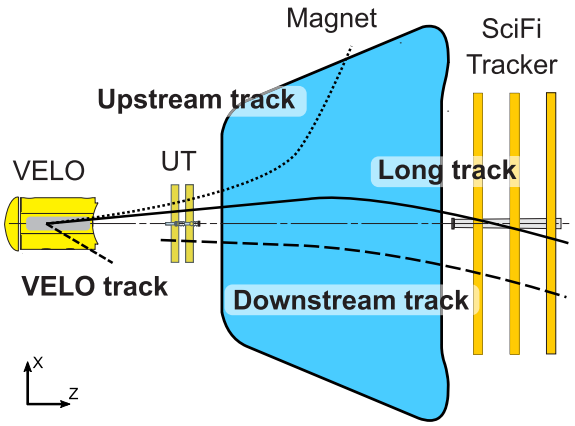

FIGURE 4. LHCb track types. Each track type is classified according to the sub-detectors it traverses. This figure represents a top view of the tracking sub-detectors, where particles travel from the collision point at the VELO to the right, crossing the UT and the SciFi Tracker, or travelling out of acceptance.

- *Downstream tracks*: contains hits recorded in UT and SciFi detectors, but not in the VELO, so their origin is external to the collision point. These tracks are not relevant for the tracking algorithm covered in this paper, but they leave hits in the UT sub-detector that are not matched to a VELO track.
- *VELO tracks*: contains hits recorded in the VELO. During UT tracking, these tracks are extended to other types of tracks if matching hits are found.

In the context of the LHCb experiment, long tracks play an important role as they traverse the full magnetic field and therefore have the most precise momentum information [30].

When doing the track reconstruction, a particle is considered to be *reconstructible* in the UT sub-detector if it has hits in three of the four layers. Various parameters are measured to determine physics efficiency [31]:

- *Track reconstruction efficiency*: It is measured with simulation data comparing the number of tracks correctly reconstructed against the number of tracks that are reconstructible. To be considered *reconstructed*, 70% of the hits on a track need to be associated to the particle from the Monte Carlo simulation. The reconstruction efficiency is given as:

$$\frac{N_{reconstructed\ \&\ reconstructible}}{N_{reconstructible}}$$

- *Clone rate*: When two or more tracks are associated to the same Monte Carlo particle, only one is considered to be reconstructed correctly and the others are counted as *clones*. The clone rate is the number of clone tracks relative to all reconstructed tracks. The clone rate is defined as:

$$\frac{N_{clone\ tracks}}{N_{reconstructed\ tracks}}$$

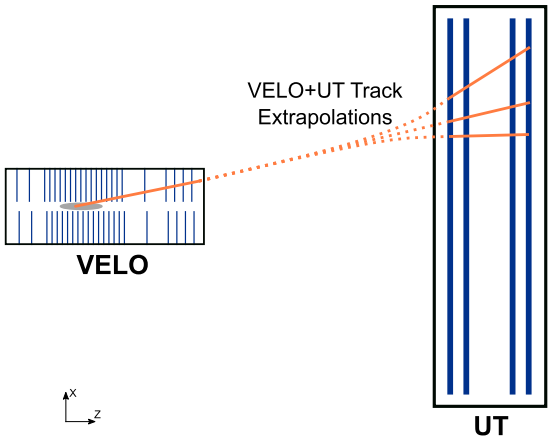

FIGURE 5. VELO track extrapolation to UT hits. A VELO track can be associated to various UT hits, where the UT track extrapolation does not necessarily follow a straight line. This leads to high combinatorics between the hits in the four panels, holding the main complexity of the algorithm.

- *Fake rate*: A track is considered a fake when it is reconstructed, but it cannot be associated with a Monte Carlo particle. The fake rate is defined as follows:

$$\frac{N_{fake\ tracks}}{N_{reconstructed\ tracks}}$$

We refer to physics efficiency to describe how good our tracking algorithm is performing, analogous to a cost function that uses the three parameters, reconstruction efficiency rate, clone and fake rates. There is no analytical form of such cost function, where an algorithm is said to attain good physics efficiency if the reconstruction efficiency is high, and the clone and fake rates are low.

C. UT TRACKING

Particles collide at the interaction point, and the resulting particles from the collisions are first reconstructed by the VELO sub-detector. A percentage of those particles travel out of the acceptance range of the UT, and the rest of them, in acceptance, leave activation signals with a high probability which are decoded in software to hit information. Using the VELO tracks and the UT hit information, combined with the geometry information and magnetic field influence from the magnet, we are able to perform the UT tracking.

Tracking is done by finding matching UT hits for every input VELO track, where a VELO track is a straight line. UT hits are considered to be compatible with a VELO track, resulting in a curved track bent proportionally to the track momentum. As the UT sub-detector is under the influence of the magnetic field, multiple possible matching hits can be matched for different slightly bent tracks [32]. This situation is represented in Figure 5, where a real situation is better represented with hundreds of tracks, and makes the problem of finding matching hits an exponential combinatorics problem [33].

The p-Kick method [34] is used to estimate the momentum of the track. Using it allows to perform a χ^2 fit providing the momentum of the particle. This method is used instead of a Kalman filter, used in other tracking algorithms, as it yields a better computing performance [35]. To take into account the magnetic field during the algorithm, look-up tables are used, which give quick access to the influence of the magnetic field in different parts of the particle trajectory. Using the look-up tables, the deflection a track is expected to experience can be determined.

A UT tracking algorithm is expected to achieve a high reconstruction efficiency with a low fake and clone rates for various types of tracks. The computing performance of the algorithm is determined by how many events per second can be processed for a given hardware configuration. This is a key aspect of event filtering in high-energy physics, especially for the LHCb experiment which will rely only on software for its event filter system. The combination of hardware and optimized software for it will need to process the 30 MHz rate of events in real-time.

IV. UT DECODING ON GPU

Before being able to execute the tracking algorithm, the raw input from the subdetector needs to be decoded into hit information. The decoding step needs to perform efficiently to run in real-time. We parallelize it by processing different chunks of raw input using GPUs, as it is a fundamental previous step for the tracking algorithm.

UT detector data is encoded into raw banks, in a highly compact format, containing the information required to obtain the UT hits. These raw banks are decoded into the parameters that define a UT hit. We reduced the decoded parameters to the minimum to run the UT tracking algorithm, lowering the memory footprint of the algorithm. The decoded parameters are the following:

- *LHCbID*: a unique 32 bit identifier for the hit, which indicates the spatial position of the detection element.
- Z *at* $Y = 0$: the Z coordinate of the hit at the $Y = 0$ position, which is the center of the panel in the Y axis. The Z coordinate indicates the panel for a specific hit.
- X *at* $Y = 0$: similarly to the previous parameter, this is the X position at the center of the panel in the Y axis. This coordinate is given by the activated strip in a sector and it is different for the U and V layers.
- *yBegin* and *yEnd*: as the UT subdetector is a strip detector where the strips are arranged vertically, the specific Y coordinate of a hit cannot be gotten. Instead, a range on the Y axis delimits where the hit is located.
- *weight*: the uncertainty of the hit position.

The decoded parameters are stored in a structure of arrays (SoA). We use a SoA layout storing the hits in a coalesced manner to maximize the memory bandwidth usage. To access the hits efficiently, a separated array is used to store the offsets between the hits. Using the offsets, we are able to determine which panel we are referring to when accessing the hits, and so every GPU thread can access its specific hit.

TABLE 1. Kernel configuration for UT decoding. *events_in_execution* are the number of selected events to process, where *array_size* is defined as the *events_in_execution* × 84. 84 is the number of pre-defined sectors, where the number 4 used in various kernels is the number of panels.

kernel	blocks	threads
calculate number of hits	$events\_in\_execution$	(64,4)
prefix sum reduce	$(array\_size + 511)/512$	256
prefix sum single block	1	1024
prefix sum single scan	$((array\_size + 511)/512) - 1$	512
pre-decode	$events\_in\_execution$	(64,4)
find permutation	$(events\_in\_execution, 84)$	16
decode raw banks in order	$(events\_in\_execution, 4)$	64

We compute the events by processing them in parallel, assigning single events to single blocks to distribute them in the GPU. An event results in various tracks, where we apply different nested parallelization schemes for different kernels, which are described here.

We group the decoded hits into *sector groups*, which are composed of various sensors. Each *sector group* carries a number of hits that are guaranteed to be within certain X coordinates. Within a *sector group* hits are not sorted by X coordinate, making it faster to sort. This also allows for quick look-up of hits in the tracking algorithm, targeting specific sector groups and searching hits only in those. Hits are sorted into pre-defined regions of the *sector groups*, then sorted by Y coordinate within the *sector group*. We divide the complete decoding into 7 GPU kernels, where we found the configuration in Table 1 to be the fastest for the UT decoding.

- *Calculate number of hits*: the first kernel uses pre-defined regions in the X axis, where the regions in the center of the panel are narrower due to the increased number of tracks expected based on previous LHCb data takings. Raw banks are processed to calculate the number of hits, used to create the array to store the offsets between the hits in memory, in a coalesced manner. To process the raw banks in parallel we set a two-dimensional kernel, parallelizing over the raw banks and over the number of hits in each raw bank.
- *Prefix sum*: we implement a parallel prefix sum of the hits, specifically a *two-step Blelloch scan* composed of a reduce and down sweep operations. It results in an array with the sums of the offsets, so their positions and sizes can be obtained [36]. After doing the prefix sum the total number of hits is obtained, which allows us to pre-allocate the memory for the hits. The prefix sum is implemented here in three separate kernels, as seen in Table 1.
- *Pre-decode*: using the data structure created during the prefix sum, the coordinates of the hits for each raw bank can be decoded. Parallelising over the raw banks and over the number of hits in each raw bank, the strip information to get the subdetector region, panel and sector of the hit is extracted. Using this information we decode the *X at Y=0*, and *yBegin* coordinates to delimit the hit in the Y axis.
- *Find permutation*: it calculates the required permutations to sort the hits by Y coordinate, based on their decoded Y coordinate limits. Hits are sorted within every group defined by the previously decoded X coordinate. We implement an insertion sort in shared memory, storing the Y coordinate in it, and parallelizing over the hits found in each sector group.
- *Decode raw banks in order*: to perform the actual decoding of the UT hits a gather operation is used. It gets geometry and panel information from the subdetector, and stores the parameters in a coalesced manner. The hit information is stored in its correct position using the pre-defined X coordinate regions and the permutations calculated in the previous kernel. For this kernel, we parallelize over the hits found on each layer.

V. COMPASS TRACKING ALGORITHM

We designed the *Compass* tracking algorithm so it can be configured by two parameters: the number of sectors to search for hit candidates, and the number of valid found candidates to test to form a track. Different configurations of these parameters gives us a configurable trade-off between computing and physics performance.

Compass is focused on the SIMD many-core parallelism offered by GPUs and its memory characteristics to develop a high-throughput algorithm. To achieve high-throughput we perform tracking on thousands of tracks in parallel, in real-time, where each particle trajectory can be computed independently one from each other. We benefit from this to design the algorithm around an SIMD model, where GPUs implement it in a SIMT (Single Instruction Multiple Thread) execution model. The operations needed to calculate the particle trajectories require arithmetic and matrix operations with single precision floating point numbers, where GPUs have shown to offer speed-ups in scientific computations. We access the decoded window ranges stored in a SoA data layout. Other multi-threaded architectures like modern x86-64 should also benefit from a SoA layout, as the access pattern by the different threads also benefit from data locality and coalesced access. The NVIDIA Profiler was used to optimize and find the spots to parallelize.

Compass is divided in two main components: searching for the UT window ranges in the indicated sectors, and using those window ranges to perform the tracking. In both cases, VELO tracks are used as input, and are extrapolated to the UT panels.

A. SEARCH UT WINDOWS

UT window ranges are defined by the indexes of two hits, one at the beginning of the window and the other at the end, where hits in between these two are considered for creating a track. The search for UT windows is performed using the information about how hits are sorted during the decoding. A two-dimensional kernel is used to search the windows: the first dimension parallelizes over the four UT panels, where the second does it over the input VELO tracks. We define the kernel like this to optimize for the windows ranges to be stored in SoA layout, where we tested different kernel

configurations, concluding this one to yield the best performance. Window ranges are stored in a coalesced manner for a panel, where panels are also stored contiguously between them. The two-dimensional kernel is used to favor the access pattern, first over the panels, then over the different tracks. We found this configuration to be faster than setting the kernel the opposite way, or just parallelizing over the tracks in a one-dimensional kernel.

For each input VELO track, the extrapolation to the UT panels is calculated taking into account the magnetic field. The extrapolation defines the sector group in the UT to search for. Since sector groups are sorted by X into known regions, a binary search is used to efficiently locate the region where the extrapolation is pointing to. With the region delimited by X, a tolerance window based on the VELO track extrapolation is used to delimit the Y region. Searching with two binary searches over the Y axis, one to delimit the beginning of the region and another to delimit the end of it, leaves us with the window range that indicates the valid UT hits for the associated VELO track. Only two pointers to the hits are used to indicate a window range. Finally the window range is refined by checking the hits to be valid within the VELO tolerance window. Iterating forward for the beginning hit, and backward for the end hit, hits are tested to meet the conditions for the VELO track tolerance. This calculation is performed here to reduce the window ranges, which we found to be faster compared to only perform it in the tracklet finding kernel. When computing the tracking kernel combinations between the hits in different panels are tested. Using a larger window range during the tracking has a larger impact in the complexity to compute the kernel compared to refining the window range during the window search. As the hits in a sector group are not sorted, the VELO tolerance check has to still be performed again in the tracking kernel because hits could be out of the tolerance window.

When looking for window ranges, a VELO track may be outside the UT acceptance region or may be directed in backwards direction, making the track unsuitable for UT tracking. When a thread is assigned to a track that meets any of those conditions, the whole thread is left unused until the rest of the threads in its *warp* finish finding the window regions. Some threads are left unused for every event, lowering the throughput capacity of the algorithm. To maximize thread occupation an array of pointers to tracks in shared memory is used, which is filled with valid tracks only. The array is filled until it holds at least the same amount of tracks as number of threads per block. We search windows parallelizing over the array of pointers to valid tracks, maximizing thread occupation.

We implement the window search to look for hits in one, three or five sectors. We do this because we found the number of hits found in only one sector to be insufficient to achieve good enough physics performance. The selected sector and its neighbors are used to get hit candidates, as can be seen in the Figure 6. If the extrapolated VELO track is pointing to a sector close to the borders of the UT panel, less sectors are searched. The window ranges are stored in a pre-allocated

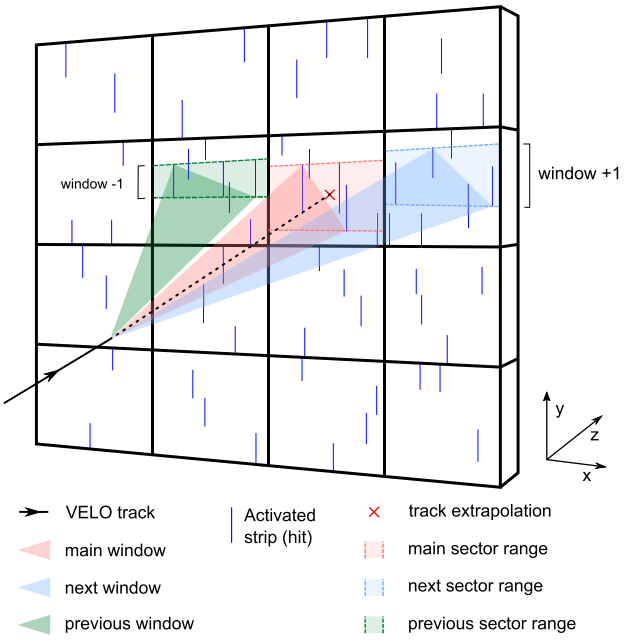

FIGURE 6. UT window ranges: Representation of a VELO track extrapolation to a sector. Window ranges are set for the sector and its neighbors. Several hits lie within the range of the windows, which are considered for UT tracking.

memory space, as the number of sectors to use and VELO track is already known, so they can be stored in parallel for every thread. When an invalid window range is found, it is stored with `(−1, −1)`, indicating that no valid hits were found. By doing this the kernel presents a lower branching ratio, leaving a similar code path for all tracks searching the windows, making it efficient for GPUs.

Finally, window ranges are stored as pairs composed of a beginning hit and the size of the window. As we will iterate over the hits in the window, knowing in which window the hit starts and the size of the window is all the information we need to access the hits. To store the hit and the size of each window we use two signed 16-bit types (`short`). The hit index is set to be relative to its own track, for all the possible indexes to fit in a `short` type, thus reducing the memory footprint. Hit pointers and window range sizes are stored grouped so all hits are contiguous between them, and per track, as can be seen in Figure 7.

B. TRACKLET FINDING

To perform UT tracking, a search for the best compatible hits needs to be performed in all the UT panels to form a tracklet. A tracklet is composed of at least 3 hits on different panels. The combination that best matches the extrapolation from the VELO track is searched, considering the influence of the magnetic field that introduces a small kink in the particle trajectory. The window ranges calculated in the previous kernel are used to find a tracklet of one hit per UT panel, allowing for one missing UT hit. The main complexity of *Compass* lies in the tracklet search, where compatible hits

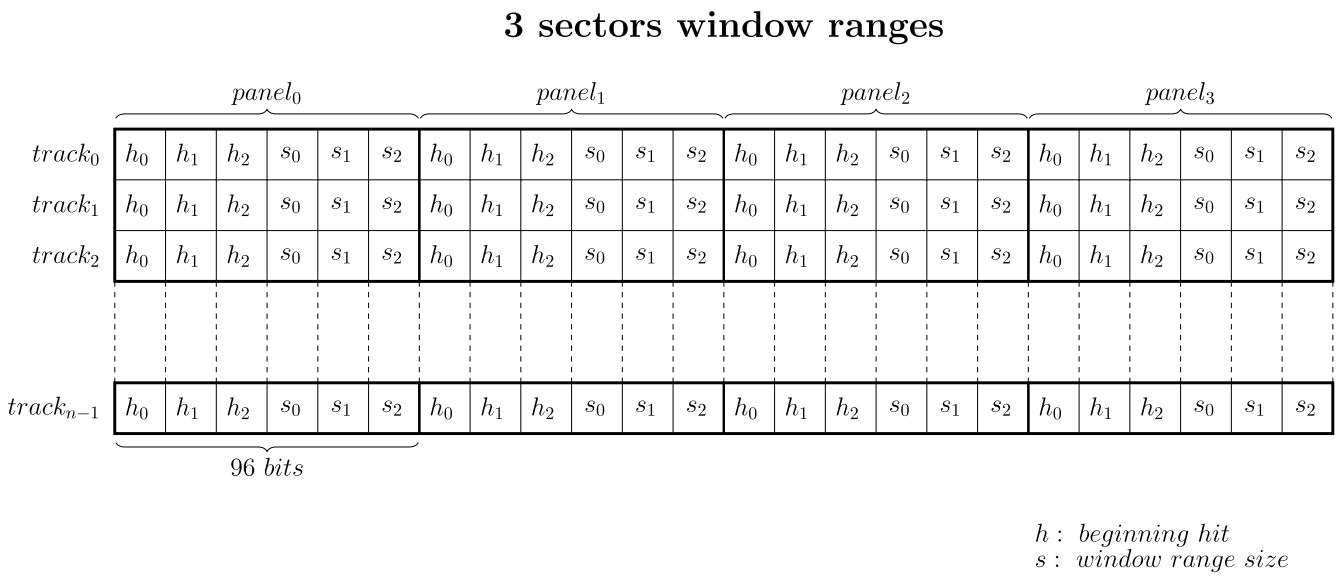

FIGURE 7. Memory layout of window ranges. A beginning hit, and a size are stored per window range, using 16 bits for each element. In this figure, a 3 sectors window ranges is shown, where each elements has a size of 16 bits, making it a total of 96 bits for all the elements of a panel.

between all panels are tested for compatibility, increasing the multiplicity of the combinations.

When a valid hit is found in the first panel, it is selected to be combined with a valid hit from the third panel. If a valid hit is also found in the latter the slope formed between them is calculated. The just calculated slope and the one of the VELO track are used to define a tolerance window in the second and fourth panels. Compatible hits are searched in these panels to form the final tracklet, as can be seen in Figure 8. Finding a third hit is enough to from a tracklet, where a tracklet of four is preferred if it is found. The complexity of tracklet search is $O(n^3)$, as the search for third and fourth hits are not nested between them. The tracklet search is performed both in forward and backwards directions, where the same algorithm is applied changing the order of the panels. Forward and backwards search is merged into one single loop, where hits are searched first in forward direction and if no hits are found, the backwards direction is tested to find a tracklet.

The algorithm may be configured to use more than one window range, in this paper for one, three or five window ranges. Instead of looping independently over the ranges to find a tracklet, these are combined into one single loop, as if these were one single range. A pointer to a selected hit within a window range is used to iterate. The ranges are combined so the central one is used first, then its immediate neighbors. If five sectors were selected, the sectors in the extremes are searched the last. Forward and backward searches are combined, as we found this way of iterating over the hits to be faster than performing two separate searches for forward and backward direction, as thread divergence is removed. We parallelize the searches for every VELO track, where all the threads in a *warp* will have to wait if a divergent branch is encountered in one of the threads. When we split the hit search into two loops, a divergent branch is introduced if

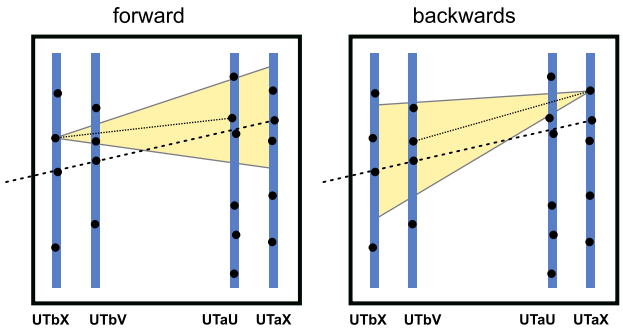

FIGURE 8. Tracklet finding kernel. Combinatorics between all 4 panels when searching for hits candidates to form a tracklet are shown. The fine dotted line represents the slope between the two first hits found in the first and third panels. The coarse dotted line represents the VELO track slope. A tolerance window defined by them is calculated to search for a tracklet.

different tracks are searching in forward and backward direction within a *warp*. A small divergent branch is introduced at the beginning of the loop when combining the window ranges. This is done to set the pointer to the correct hit, which allows the *warp* to run all tracks in a parallel fashion even if they diverge in both ranges or direction.

Compass implements a configurable number of search hit candidates that will be considered. When a valid tracklet is found, if more than one candidate was configured, the next valid hits within the window ranges are tested to form a different tracklet. For every tracklet the χ^2 fit of the track is obtained in combination with the VELO track. If more than one tracklet is found, we perform a selection favoring tracklets with 4 hits instead of 3, and with the lowest χ^2 fit value. The algorithm keeps searching for a better tracklet according to the configured hit candidates value.

Compass is parallelized over the VELO tracks, where each thread processes the tracklet search for each track. When processing the VELO tracks a similar filtering mechanism is applied as when searching for the window ranges explained in subsection V-A. It differs in the conditions to save a valid track, looking for the track to be within UT acceptance, not backwards and to have at least one valid window range. Only the size of the window range is checked to be different from -1 to indicate a window range with at least one valid hit.

We also take advantage of the GPU shared memory to cache the window ranges, as these are accessed during the tracklet search. A shared memory array of size $num\_threads \times num\_panels \times size\_window\_range$ is used to accommodate all the window ranges in a block. As in the search window ranges kernel, we store the window ranges using a signed 16-bit type to save in memory. When processing a valid track, the window ranges for that track are copied to its correct position relative to the block size into shared memory, where only the pointers to the shared memory array are used afterwards. We found this to be faster in all the configurations and GPUs we tested.

When a final tracklet is selected as the best one, the found hits are stored and associated to its VELO track as a VELO+UT track. Alongside the hits, the charge of the particle, calculated from the momentum of the track from the χ^2 fit, and the index of the track within the event are stored, obtained by atomic addition of the track number for this event.

C. CPU IMPLEMENTATION

We implement a CPU version of *Compass* tracking to compare its computing performance against our baseline GPU implementation. To port the algorithm part of the structure of the algorithm is modified. The GPU specific optimizations are removed, which cannot be exploited in a non-GPU architecture. On the baseline GPU version we minimize thread divergence and store various structures into shared memory, whereas the impact of branches is minimized by design in a CPU architecture compared to a GPU architecture [37], [38]. We consider the impact of using shared memory and caching the window ranges in the ported version to be better managed by the large caches found in a modern CPU, compared to the ones in the GPUs. The computation of searching window ranges and tracklet finding is not split into separated kernels, where the window searches are calculated for every VELO track in-place before doing the tracklet search. We do so to benefit from cache locality, as the just calculated window ranges will be used by the tracklet search algorithm.

VI. EXPERIMENTAL EVALUATION

This section covers the performance and physics efficiency evaluation of our proposed algorithms. We have conducted multiple micro-benchmarks using different configurations for both the number of sectors and the number of candidates.

TABLE 2. GPU and CPU hardware employed for the evaluation. Two high-end consumer graphics cards (GeForce GTX 1080Ti and GeForce RTX 2080Ti), two server-grade cards (Tesla T4 and Tesla V100), and an Intel Xeon CPU are compared. We show the number of cores of each processor, where for the GPUs we count the CUDA cores only (no RT cores or Tensor cores are used in the benchmarks). We take the MSRP (manufacturer suggested retail price) for each hardware unit used here. The price for a single Intel Xeon CPU is shown, whereas for the benchmarks a dual socket server with two Intel Xeon CPUs is used. This is reflected in the price performance figure.

Unit	# cores	Max freq. (GHz)	Cache (MiB - L2)	DRAM (GiB)	TDP (W)	MSRP ($)
GeForce GTX 1080 Ti	3,584	1.67	2.75	10.92 GDDR5	250	699
GeForce RTX 2080 Ti	4,352	1.54	6	10.92 GDDR5	250	1,199
Tesla T4	2,560	1.59	6	16 GDDR6	70	2,350
Tesla V100 V100	5,120	1.37	6	16 HBM2	250	8,899
Intel Xeon E5-2678W v3	20	3.50	25 (L3)	64 DDR4	160	2,145

A. EXPERIMENTAL SETUP

Four GPUs and a x86-64 CPU were used for the benchmarks. Two consumer-grade GPUs of different generations and two server-grade GPUs are employed. A dual socket server-grade CPU is used for the *Compass* tracking CPU implementation. The specifics of the hardware are detailed in Table 2.

The software relies on CUDA 10.0 and `gcc` 7.3.0 under the `-O3` optimization flag. The following compilation flags were used: `-use_fast_math -expt-relaxed-constexpr` and `-maxrregcount=63`. The use of those flags were beneficial for the overall execution time of our algorithm [39].

All the benchmarks use the same sets of Monte Carlo simulated events, generated using the LHCb simulation framework. Two different testbeds of events are evaluated: the *minbias* set for throughput performance and the *BsPhiPhi* to check reconstruction efficiency. The *minbias* (minimum bias) set is a realistic simulation of the current expected physics, where data rate and therefore computing performance obtained with it match the realistically expected one. The *BsPhiPhi* set contains more tracks from the rare decay $B_s \rightarrow \phi\phi$. This allows to determine the track reconstruction efficiency for these physically interesting decays with higher statistical significance. It is important to highlight that the same reconstruction efficiency can be achieved in both testbeds. However, we would need more *minbias* samples to obtain the same number of tracks from the rare $B_s \rightarrow \phi\phi$ decay. Each set contains 1,000 events. For the throughput measurements, we iterate 40 times over the *minbias* events to get a sustained throughput. Both server grade GPUs are set to ECC (Error-Correcting Code) memory disabled. The evaluation metrics shown in this paper correspond with the average value of 10 consecutive executions.

B. COMPASS TRACKING PHYSICS PERFORMANCE AND THROUGHPUT

The computing performance of the algorithm is measured in terms of throughput of events per second. Different configurations of the algorithm are evaluated, taking measurements

TABLE 3. Comparison between searching in 1, 3 or 5 sector groups, and using 1 to 16 hit candidates. Two type of tracks are compared: long tracks and VELO+UT tracks. For each type of track, the track reconstruction efficiency and track clone rate achieved are presented. The obtained fake rate for each case is also shown.

Number of sectors	Number of candidates	Long tracks		VELO+UT tracks		Fake rate
		reco. efficiency	clone rate	reco. efficiency	clone rate	
1 sector	1	71.91%	0.36%	61.88%	0.32%	7.73%
	2	76.53%	0.36%	69.99%	0.32%	7.78%
	4	79.09%	0.31%	74.31%	0.32%	7.70%
	8	80.36%	0.34%	76.58%	0.35%	7.61%
	16	80.52%	0.34%	77.04%	0.35%	7.52%
3 sectors	1	84.70%	0.39%	66.87%	0.32%	7.64%
	2	90.07%	0.38%	75.61%	0.33%	7.62%
	4	93.31%	0.35%	80.32%	0.32%	7.52%
	8	94.72%	0.36%	82.66%	0.35%	7.43%
	16	94.94%	0.36%	83.19%	0.35%	7.33%
5 sectors	1	85.23%	0.39%	67.10%	0.31%	7.70%
	2	90.65%	0.38%	75.84%	0.32%	7.67%
	4	93.89%	0.35%	80.52%	0.32%	7.56%
	8	95.27%	0.36%	82.87%	0.35%	7.47%
	16	95.49%	0.36%	83.40%	0.35%	7.38%

when looking into 1, 3, and 5 sectors and different number of hit candidates for 1 to 16 when looking for a better tracklet.

The obtained physics efficiency is shown in Table 3 for the long and VELO+UT tracks. We focus on the long tracks, as these are the preferred ones for analysis. Long tracks carry more information about the momentum resolution. We also analyze the VELO+UT tracks, as these are constructed with the two main inputs of the *Compass* algorithm, VELO tracks and UT hits [40]. Note how for the 3 sector cases, when searching for more hit candidates, the physics efficiency improves. The biggest improvements are achieved in track reconstruction efficiency, where the clone rate increases by less than 0.1% in all cases. Note how the reconstruction efficiency gains flattens when using more hit candidates. While the number of hit candidates is increased exponentially, the track reconstruction efficiency gains do not follow the same increase pattern, but the opposite. This behavior matches our expectations, as in most of the cases, the best tracklet is found in the first set of hit candidates, and therefore, the subsequent ones do not yield a better hit tracklet as often. Calculating the subsequent tracklets has an impact on the throughput performance even if no better tracklet is found, where the physics performance does not improve. The fake rate decreases when using more sectors and candidates, with differences in the range of 1% across the whole scope of benchmarks. Note how the impact of both changing the sectors and candidates has little effect on the clone and fake rates, whereas it has a big impact in the reconstruction efficiency rate.

The reconstruction efficiency achieved when searching in one sector does not reach 90% for long tracks nor 80% for VELO+UT tracks for any number of hit candidates. These reconstruction efficiency does not meet the requirements for the LHCb UT reconstruction, and therefore, we discard the one sector configuration in the following analysis.

In Figure 9, we plot the differences in throughput between all the configurations, using 3 and 5 sectors, and from 1 to 16 candidates. Note how searching for more candidates decreases the throughput, as it needs to iterate over more hits in a $O(n^3)$ algorithm to find a better hit tracklet. The performance degrades more when using more candidates, contrary to what we observed with the physics performance, where the gains were very small by doubling the number of candidates when using the bigger number of candidates. When searching for more hit candidates, the hit tracklet needs to be constructed, and their χ^2 calculated, even if for most of the cases the last calculated hit tracklet does not improve over the previous one.

We highlight the difference in performance between the four evaluated GPUs devices. The 1080Ti and Tesla T4 have a comparable performance despite of the difference in terms of number of cores. We attribute the comparable performance between the two cards to the bigger cache size encountered in the Tesla T4 and its faster GDDR6 memory. The difference in thermal design power (TDP) is very significant, where the 1080Ti consumes 3× more compared to the Tesla T4 to deliver a comparable throughput. The difference in performance between the 1080Ti / T4 compared to the 2080Ti is bigger than the difference found between the 2080Ti and the Tesla V100, with closer comparable performance when using 5 sectors compared to 3. Tesla V100 outperforms the rest of the GPUs due to its High Bandwith Memory (HBM) and increased number of cores, having double the number of cores compared to the T4, 15% more compared to the 2080Ti, and 30% more compared to the 1080Ti as show in Table 2. One generation difference for the high-end consumer cards yields double the throughput for the 1080Ti compared to the 2080Ti for our algorithm.

Note the difference in performance for comparable physics efficiency on different results. We observe a comparable physics efficiency in the long tracks between the 5 sectors - 8 candidates case, and the 3 sectors - 16 candidates case. Taking the Tesla V100 as reference example, a difference in performance of roughly 15% (500k vs 585k) is observed, whereas the difference in physics efficiency is below 1%. The throughput differences change between the tested hardware

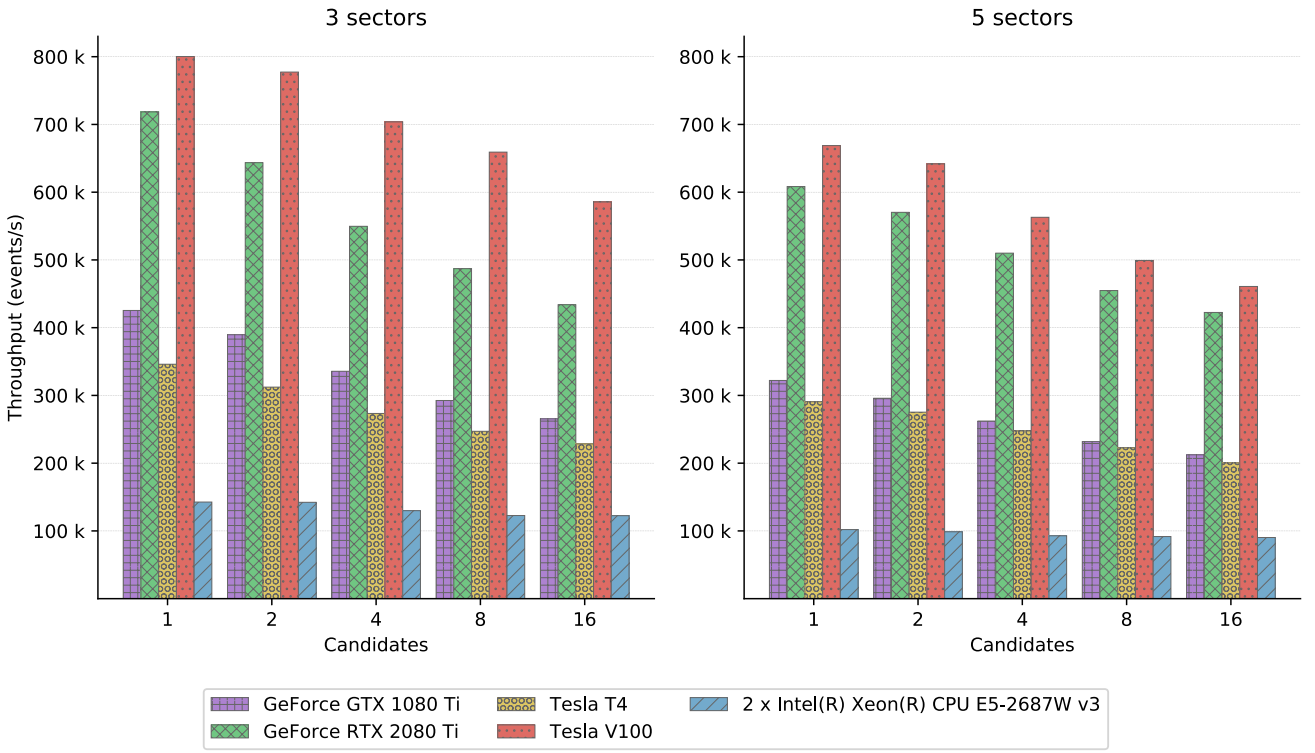

FIGURE 9. 3 vs 5 sectors *Compass* tracking comparison. Throughput comparison between the two consumer grade GPUs, two server grade GPUs and a dual socket Intel Xeon CPU, comparing with 1 to 16 number of hit candidates. The throughput shown here corresponds to running the *Compass* algorithm. In the figure in the left we plot the throughput when looking for hits in 3 sectors. In the right figure, we depict the throughput when looking for hits in 5 sectors, adding an extra neighbor sector on each side with respect to the 3 sectors case.

for different number of candidates and sectors. Not that for comparable physics performance, the 5 sectors version performs better in throughput.

We port our *Compass* tracking algorithm so that it runs on architectures other than the GPUs, to perform a cross-architecture tracking performance comparison. The CPU version differentiate from the GPU version in the implemented optimizations but computes the same algorithm and uses the same data layout and access patterns, as explained in V-C. OpenMP is used to parallelize over the events and tracks, following the same parallelization scheme as in the GPU version. We ensure all cores are used in both the CPU and GPU versions for the comparison. Note how the parallelization differs in the SIMD approach of the GPUs compared to the multi-threaded version of the CPUs, where the CPU version relies on the improvements made by the compiler due to the SoA data layout to exploit the SIMD capabilities of the CPU. The performance difference between a dual socket Intel Xeon CPU and the 1080Ti GPU and Telsa T4 is found to be up to 3× faster for the GPUs, up to 6× faster for the 2080Ti, and more than 6× faster for the V100. Note how the CPU version of the algorithm degrades less its performance compared to the GPUs when increasing both the number of sectors and candidates. We attribute this to the better branch prediction in the CPU, and the impact of divergent threads on the GPU, where the GPU runtime performance is affected more by the increased number of branches, and the work imbalance keeps warps active with low occupation, due to the increased number of candidates and sectors.

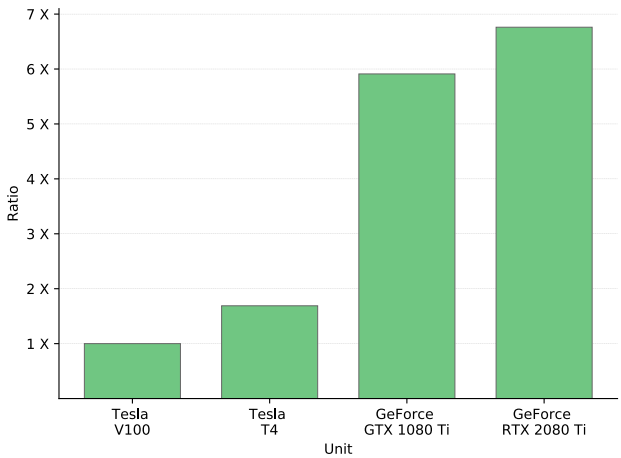

FIGURE 10. Price performance ratio for *Compass* in GPU. All prices are factored to MSRP price indicated in Table 2. We compare the price performance of the 5 sectors case, for the best physics efficiency case with 16 candidates.

In Figure 10, we plot the price performance ratio for the different target GPUs. This figure shows the case for best physics performance with 5 sectors, using 16 hit candidates. It is normalized to the Tesla V100 and compares the other analyzed hardware accelerators in terms of achieved speedup in

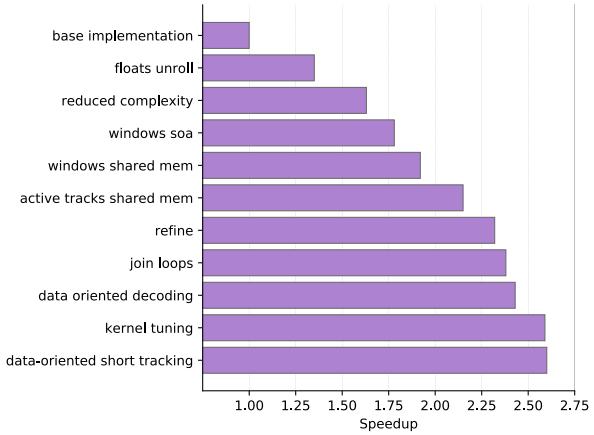

FIGURE 11. Incremental optimizations speedup. Speedup achieved after applying different optimizations to the baseline code. A maximum speedup of 2.6× is achieved in the final version, compared to the baseline implementation. Various small optimizations and changes are grouped into steps.

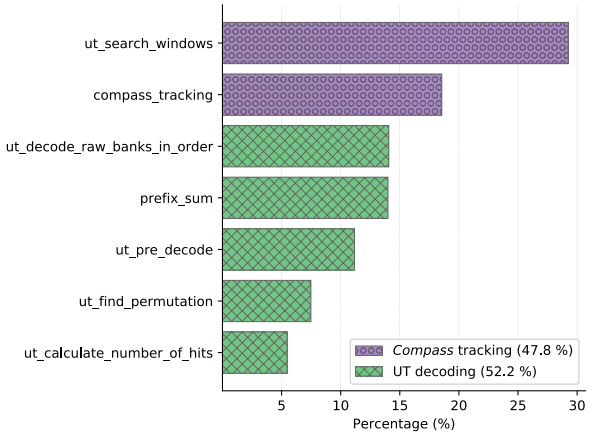

FIGURE 12. Kernels time contribution. Runtime distribution of all the kernels used to compute the decoding and *Compass* algorithm. The best physics efficiency case is used here, with 5 sectors and 16 candidates for the NVIDIA 2080Ti case.

terms of price/performance. Note how the price performance achieved for all the evaluated hardware is given for its MSRP[3] with the prices shown in Table 2. Tesla V100 performs the worse in all the tested GPUs for its price performance, while it achieves the best throughput. Note the comparable price performance between the server grade Tesla GPUs compared to the consumer GPUs, where the consumer GPUs perform around 5× better than the server grade ones despite their differences in throughput. We note a 1.7× speedup between the Tesla V100 and the Tesla T4, and a 1.15× speedup between the 1080Ti and the 2080Ti, being the consumer grade GPUs close in price performance despite the 2080Ti doubling the 1080Ti in throughput. The achieved price performance speedup between the Tesla V100 and the 1080Ti is 5.9×, and 6.7× for the 2080Ti. The 2080Ti obtains the best price performance due to the achieved high throughput and low unit price. The 2080Ti delivers a throughput close to the Tesla V100 with significant less price due the lack of some server-grade characteristics such as HBM or ECC memory.

C. UT DECODING AND TRACKING PERFORMANCE

In Figure 11, we show the speedup achieved for various iterations of optimizations, compared to the initial GPU implementation. Various small improvements and optimizations are grouped into the 11 steps presented in Figure 11. We refer to the first working version that implements the main ideas of the algorithm as *baseline implementation* and apply various optimization on top of it to achieve the final 2.6× speedup. For *floats unroll*, we get the biggest improvement of 35%. We first applied various small modifications to the algorithm, mainly changing all the floating point variables to single precision ones, unrolling some loops manually, and by giving compiler hints with the use of `#pragma`. We note how the change from double to single precision does not affect the physics efficiency. We *reduced the complexity* of window range search by splitting the algorithm in various kernels and re-writing the tracklet finding to be simpler to process when searching in more than one sector, to get a 28% improvement. We improved the window ranges storage to be *windows SoA* to get an extra 15%, and configured it to store only one hit and the size of the window, sorting them to be efficient for our access pattern. We copied the *windows to shared memory* to cache them and improve the access pattern when searching the tracklet. The speedup achieved by filtering the tracks in the shared memory array is 23%, shown in *active tracks shared mem*. When calculating the window ranges, we *refine* the window by checking the hits in both extremes, instead of calculating all the window range validity in the tracking algorithm. We further reduced the complexity of the tracklet finding by *joining the loops* and reducing thread divergence, where we got to 2.37×. We grouped various small optimization to the raw bank decoding, making the data types smaller, aligned and more efficient to be a *data oriented decoding*. We improved an extra 16% by *tuning the kernel* parameters of all the kernels in the decoding and *Compass*, changing to multi-dimension kernels and changing how the kernels are parallelized. Finally, we reduced the memory footprint and made the copies faster by reducing further the data types, by storing types in signed 16-bit instead of 32-bits structures to get the final overall speedup of 2.6×.

Figure 12 depicts the runtime distribution of both kernels used to perform the decoding and the kernels of the *Compass* tracking algorithm. We show the distribution for the best physics case, 5 sectors - 16 candidates, where we encountered similar runtime distributions when using different configurations and different GPUs. Note how *Compass* tracking runtime is dominated by the window searching algorithm compared to the tracklet finding. The refining of window ranges was moved from the tracklet finding to the window range search, increasing the time contribution of the kernel while improving the overall throughput. Note how the

[3]The prices shown in this paper are collected from those recommend by NVIDIA and Intel web site or Amazon.com otherwise.

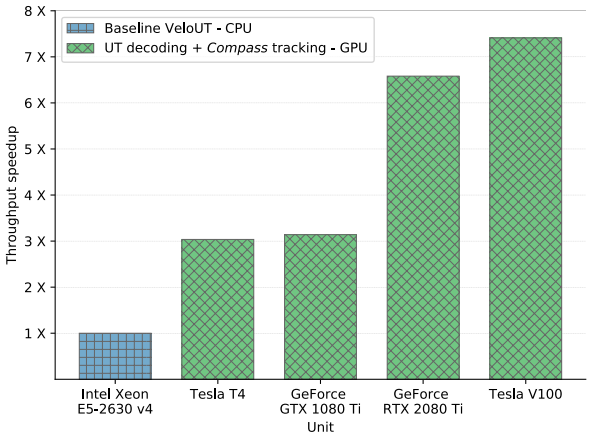

FIGURE 13. Basline LHCb vs GPU decoding + *Compass* tracking throughput speedup comparison. Throughput speedup of the full UT chain of kernels, including the decoding and *Compass* tracking, compared to the baseline LHCb CPU implementation as stated in Section 13. We compare the LHCb baseline (blue) with the *Compass* over different GPUs (green).

complete decoding of the UT hits accounts for more than half the time needed to compute the whole UT sequence.

Finally the complete implementation explained in this paper is shown, with the decoding and tracking in GPU compared to the equivalent algorithms found in LHCb baseline implementation. We acknowledge that the results compared here have changed and improved since the publication of these numbers in [41] used for the comparison, where more recent results are not found or published. We set comparable conditions as those found in [41], where we apply the same Global Event Cut, which filters a selection of events, at the beginning of the chain, thus reducing the amount of processing the tracking algorithms need to do. We add data preparation kernels after the full UT chain is processed, in the form of a prefix sum and consolidation steps to leave the tracks in coalesced memory for the algorithms using UT tracks as input. The LHCb baseline implementation uses a Intel Xeon E5-2630 v4,[4] which delivers a top throughput of 12,400 events per second for the full sequence.5 Combining the time contributions of the UT decoding and tracking for peak throughput yields the results shown in Figure 13. We compare these results to the full UT decoding and *Compass* tracking presented in this paper. The throughput speedup shown corresponds to our *Compass* implementation using the configuration for 5 sectors and 8 candidates. Both the Tesla T4 and 1080Ti achieve roughly a 3× speedup, where the latter performs slightly better than the T4. The 2080Ti achieves a speedup of 6.5× and the Tesla V100 achieves the best speedup at 7.4×. We acknowledge that the physics results obtained in both implementations are comparable, but yield different results due to the different algorithms used.

VII. CONCLUSIONS

We have presented a new algorithm, *Compass*, designed for parallel GPU architectures with focus to perform efficiently on GPUs. We designed our algorithm so that it maximizes throughput processing on GPUs by being data-oriented, minimizing branching, reducing the memory footprint of the algorithm and taking advantage of the architectural characteristics of GPUs.

We presented a SIMD parallel UT raw data decodification algorithm, data-oriented and optimized for GPUs. We demonstrated a new hit organization that stores hits in SoA, in a parallel and coalesced manner, where we sorted groups of hits into regions for fast decoding. We benefit from the new hit organization to search efficiently for sector regions, defining window ranges that indicate where compatible hits are found. We stored the windows efficiently for parallel architectures.

We designed *Compass* to be configurable in both number of sectors to search for, and number of hit clusters to test for a tracklet. We showed the physics efficiency results when searching in one sector, proving it to yield too low reconstruction efficiency rate to be considered for performance benchmarks. We compared the performance for searching in three and five sectors, and tested with different number of hit candidates. We validated our algorithms with Monte Carlo simulated data to verify the physics performance of the results, getting comparable physics performance.

We developed a CPU tracking implementation and analyzed our algorithm in different parallel architectures, focusing on GPU architectures and comparing them against the parallel CPU implementation of the same algorithm. We showed the differences in performance across the analyzed hardware. We conclude that a physics performance close to 95% in track reconstruction is achieved with various configurations of the algorithm, where a configuration using 5 sectors and 8 hit candidates yields a throughput of 231k events per second in the 1080 Ti, 222k in the Tesla T4, 454k in the 2080 Ti, 499k in the Tesla V100 and 92k in the dual socket Intel Xeon CPU, for the *Compass* tracking. The 5% of tracks that were not reconstructed correctly do not satisfy the assumptions and selections made in this algorithm. These are not due to computational precision, as has been verified switching from single to double precision obtaining the same results.

We consider this configuration to be the best trade-off for this algorithm considering the achieved physics efficiency and the performance. We compare with the baseline LHCb results for the full UT decoding and tracking, where our GPU implementation delivers up to 7.4× more throughput with the Tesla V100, and 6.5× when comparing with 2080Ti.

We plan to evaluate the possibilities of implementing further optimizations to the algorithm by exploiting various hardware capabilities of NVIDIA GPUs, such as the usage of Tensor and Ray Tracing cores. For the CPU implementation, vectorisation opportunities could be explored to further optimize the CPU implementation of the algorithm.

ACKNOWLEDGMENTS

The authors would like to thank Vladimir Gligorov and Florian Reiss for the fruitful discussions, Alberto Ottimo for

[4]This CPU differs from the one used for our benchmarks.

early discussions on the UT decoding. We would also like to thank the LHCb computing and simulation teams for their support and for producing the simulated LHCb samples used to develop and benchmark our algorithm, and the ARCOS UC3M Group for their support.

REFERENCES

[1] L. Canetti, M. Drewes, and M. Shaposhnikov, "Matter and antimatter in the universe," *New J. Phys.*, vol. 14, no. 9, Sep. 2012, Art. no. 095012.

[2] LHCb Collaboration, "Framework TDR for the LHCb upgrade: Technical design report," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2012-007. LHCb-TDR-12, Apr. 2012. [Online]. Available: http://cds.cern.ch/record/1443882

[3] LHCb Collaboration, "LHCb trigger and online upgrade technical design report," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2014-016. LHCB-TDR-016, May 2014. [Online]. Available: https://cds.cern.ch/record/1701361

[4] LHCb Collaboration, "Expression of Interest for a Phase-II LHCb Upgrade: Opportunities in flavour physics, and beyond, in the HL-LHC era," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2017-003, Feb. 2017. [Online]. Available: http://cds.cern.ch/record/2244311

[5] J. Zhao, Z. A. Liu, W. Gong, P. Cao, W. Kuehn, T. Gessler, and B. Spruck, "New version of high performance Compute Node for PANDA Streaming DAQ system," Jul. 2018, *arXiv:1806.09128*. [Online]. Available: https://arxiv.org/abs/1806.09128

[6] M. Vogelgesang, L. Rota, L. E. A. Perez, M. Caselle, S. Chilingaryan, and A. Kopmann, "High-throughput data acquisition and processing for real-time X-ray imaging," *Proc. SPIE*, vol. 9967, 2016, Art. no. 996715.

[7] D. vom Bruch, "Online data reduction using track and vertex reconstruction on GPUs for the Mu3e experiment," in *EPJ Web Conf.*, vol. 150, Aug. 2017, p. 00013.

[8] J. Nieto, D. Sanz, P. Guillén, S. Esquembri, G. de Arcas, M. Ruiz, J. Vega, and R. Castro, "High performance image acquisition and processing architecture for fast plant system controllers based on FPGA and GPU," *Fusion Eng. Des.*, vol. 112, pp. 957–960, Nov. 2016.

[9] D. Rogora, M. Papalini, K. Khazaei, A. Margara, A. Carzaniga, and G. Cugola, "High-throughput subset matching on commodity GPU-based systems," in *Proc. 12th Eur. Conf. Comput. Syst.*, New York, NY, USA, 2017, pp. 513–526. doi: 10.1145/3064176.3064190.

[10] Z. Chen, J. Xu, J. Tang, K. Kwiat, C. Kamhoua, and C. Wang, "GPU-accelerated high-throughput online stream data processing," *IEEE Trans. Big Data*, vol. 4, no. 2, pp. 191–202, Jun. 2018.

[11] C.-L. Hsieh, L. Vespa, and N. Weng, "A high-throughput DPI engine on GPU via algorithm/implementation co-optimization," *J. Parallel Distrib. Comput.*, vol. 88, pp. 46–56, Feb. 2016.

[12] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," *ACM SIGPLAN Notices*, vol. 51, no. 8, p. 11, 2016. doi: 10.1145/3016078.2851145.

[13] D. H. C. Pérez, N. Neufeld, and A. R. Nuñez, "A fast local algorithm for track reconstruction on parallel architectures," in *Proc. IPDPS Workshops*, 2019.

[14] D. Rohr, S. Gorbunov, and V. Lindenstruth, "GPU-accelerated track reconstruction in the ALICE high level trigger," *J. Phys. Conf. Ser.*, vol. 898, Dec. 2017, Art. no. 032030.

[15] P. Sen and V. Singhal, "Event selection for MUCH of CBM experiment using GPU computing," in *Proc. Annu. IEEE India Conf. (INDICON)*, Dec. 2015, pp. 1–5.

[16] D. Funke, T. Hauth, V. Innocente, G. Quast, P. Sanders, and D. Schieferdecker, "Parallel track reconstruction in CMS using the cellular automaton approach," *J. Phys., Conf. Ser.*, vol. 513, no. 5, 2014, Art. no. 052010.

[17] R. Wilton, T. Budavari, B. Langmead, S. J. Wheelan, S. L. Salzberg, and A. S. Szalay, "Arioc: High-throughput read alignment with GPU-accelerated exploration of the seed-and-extend search space," *PeerJ*, vol. 3, p. e808, Mar. 2015.

[18] R. Wilton, A. S. Szalay, X. Li, and A. P. Feinberg, "Arioc: GPU-accelerated alignment of short bisulfite-treated reads," *Bioinformatics*, vol. 34, no. 15, pp. 2673–2675, 2018. [Online]. Available: http://oup.prod.sis.lan/bioinformatics/article-pdf/34/15/2673/25230719/bty167.pdf

[19] S. Pawar, A. Stanam, and Y. Zhu, "Evaluating the computing efficiencies (specificity and sensitivity) of graphics processing unit (GPU)-accelerated DNA sequence alignment tools against central processing unit (CPU) alignment tool," *J. Bioinf. Sequence Anal.*, vol. 9, no. 2, pp. 10–14, 2018.

[20] S. Samsi, B. Helfer, J. Kepner, A. Reuther, and D. O. Ricke, "A linear algebra approach to fast DNA mixture analysis using GPUs," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2017, pp. 1–6.

[21] N. Cadenelli and Z. Jakšić, J. Polo, and D. Carrera, "Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads," *Future Gener. Comput. Syst.*, vol. 94, pp. 148–159, May 2019.

[22] M. D. Cranmer, B. R. Barsdell, D. C. Price, J. Dowell, H. Garsden, V. Dike, T. Eftekhari, A. M. Hegedus, J. Malins, K. S. Obenberger, F. Schinzel, K. Stovall, G. B. Taylor, and L. J. Greenhill, "Bifrost: A Python/C++ framework for high-throughput stream processing in astronomy," *J. Astronomical Instrum.*, vol. 6, no. 04, 2017, Art. no. 1750007.

[23] A. Magro, "A real-time, GPU-based, non-imaging back-end for radio telescopes," 2014, *arXiv:1401.8258*. [Online]. Available: https://arxiv.org/abs/1401.8258

[24] X. Hu and Y. Zhao, "Gridding algorithm in ARL based on GPU parallelization," in *Proc. 6th ACM/ACIS Int. Conf. Appl. Comput. Inf. Technol.*, 2018, pp. 13–18.

[25] A. Recnik, K. Bandura, N. Denman, A. D. Hincks, G. Hinshaw, P. Klages, U.-L. Pen, and K. Vanderlinde, "An efficient real-time data pipeline for the CHIME Pathfinder radio telescope X-engine," in *Proc. IEEE 26th Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2015, pp. 57–61.

[26] T. Maceina, P. Bettini, G. Manduchi, and M. Passarotto, "Fast and efficient algorithms for computational electromagnetics on GPU architecture," *IEEE Trans. Nucl. Sci.*, vol. 64, no. 7, pp. 1983–1987, Jul. 2017.

[27] W. Han, H. S. Chwa, H. Bae, H. Kim, and I. Shin, "GPU-SAM: Leveraging multi-GPU split-and-merge execution for system-wide real-time support," *J. Syst. Softw.*, vol. 117, pp. 1–14, Jul. 2016.

[28] J. G. Blas, M. Abella, F. Isaila, J. Carretero, and M. Desco, "Surfing the optimization space of a multiple-GPU parallel implementation of a X-ray tomography reconstruction algorithm," *J. Syst. Softw.*, vol. 95, pp. 166–175, Sep. 2014.

[29] LHCb Collaboration, "LHCb tracker upgrade technical design report," CERN, Geneva, Switzerland, Tech. Rep. CERN-LHCC-2014-001. LHCB-TDR-015, Feb. 2014. [Online]. Available: http://cds.cern.ch/record/1647400

[30] J. Van Tilburg and M. Merk, "Track simulation and reconstruction in LHCb," Fac. Sci., Ph.D. dissertation, Vrije Univ. Amsterdam, Amsterdam, The Netherlands, 2005. [Online]. Available: http://cds.cern.ch/record/885750

[31] M. T. Schiller, "Track reconstruction and prompt K_s^o production at the LHCb experiment," Combined Faculties Natural Sci. Math., Ph.D. dissertation, Heidelberg Univ., Tiffin, OH, USA, 2011.

[32] M. Chouaki, "Analysis of low-momentum upstream tracking for the LHCb upgrade," École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, Tech. Rep., Jun. 2016. [Online]. Available: https://lphe.epfl.ch/oschneid/cours/2015-2016/rapports_TP4/TP4b_Mourad_Chouaki_UpstreamTracking_jun2016.pdf

[33] E. E. Bowen, U. Straumann, N. Serra, O. Steinkamp, and B. Storaci, "Upstream tracking and the decay $B^0 \rightarrow K^+\pi^-\mu^+\mu^-$ at the LHCb Experiment," Ph.D. dissertation, Univ. Zurich, Zürich, Switzerland, Oct. 2016. [Online]. Available: http://cds.cern.ch/record/2261918

[34] E. Bowen and B. Storaci, "VeloUT tracking for the LHCb upgrade," CERN, Geneva, Switzerland, Tech. Rep. LHCb-PUB-2013-023. CERN-LHCb-PUB-2013-023. LHCb-INT-2013-056, Apr. 2014. [Online]. Available: http://cds.cern.ch/record/1635665

[35] E. E. Bowen, B. Storaci, and M. Tresch, "VeloTT tracking for LHCb Run II," CERN, Geneva, Switzerland, Tech. Rep. LHCb-PUB-2015-024. CERN-LHCb-PUB-2015-024. LHCb-INT-2014-040, Apr. 2016. [Online]. Available: http://cds.cern.ch/record/2105078

[36] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA," *GPU GEMS*, vol. 3, no. 39, pp. 851–876, 2007.

[37] H. Lin, C.-L. Wang, and H. Liu, "On-GPU thread-data remapping for branch divergence reduction," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 3, p. 39, 2018.

[38] Y. Djenouri, A. Bendjoudi, Z. Habbas, M. Mehdi, and D. Djenouri, "Reducing thread divergence in GPU-based bees swarm optimization applied to association rule mining," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 9, p. e3836, 2017.

[39] NVIDIA Corporation. (2019). *CUDA Toolkit Documentation*. [Online]. Available: https://docs.nvidia.com/cuda/
[40] LHCb Collaboration, "Measurement of the track reconstruction efficiency at LHCb," *J. Instrum.*, vol. 10, no. 2, 2015, Art. no. P02007.
[41] M. De Cian, A. Dziurda, V. Gligorov, C. Hasse, W. Hulsbergen, T. E. Latham, S. Ponce, R. Quagliani, H. F. Schreiner, S. B. Stemmle, J. Van Tilburg, M. J. Zdybal, and J. M. Williams, "Status of HLT1 sequence and path towards 30 MHz," CERN, Geneva, Switzerland, Tech. Rep. LHCb-PUB-2018-003. CERN-LHCb-PUB-2018-003, Mar. 2018. [Online]. Available: http://cds.cern.ch/record/2309972

PLACIDO FERNANDEZ DECLARA received the degree in computer science and engineering, in 2013, and the M.Sc.Eng. degree in software engineering from the University Carlos III of Madrid. After that, he joined Thales for a year, developing validation software for rail signal systems. He then joined the Flight Dynamics and Operations Department, GMV Space and Defense, developing their software products. He joined CERN as a doctoral student in the LHCb experiment, in 2016, where he conducts his research in HPC for particle tracking algorithms.

DANIEL HUGO CÁMPORA PÉREZ received the bachelor's and master's degrees from the University of Sevilla. He is currently pursuing the Ph.D. degree with the University of Sevilla and CERN. He has been with CERN, since 2010, where he previously worked as a Technical Student with the ATLAS Network Administration Team and as a Fellow at the LHCb Online Team. His main focus is in the optimization of high-throughput real-time processes in physics reconstruction with parallel architectures.

JAVIER GARCIA-BLAS received the Ph.D. degree in computer science from the University Carlos III, in 2010. He has been an Associate Professor with the University Carlos III of Madrid, since 2019. He has cooperated in several projects with researchers from various high-performance research institutions such as HLRS (funded by HPC-Europe program), DKRZ, and Argonne National Laboratory. He is currently involved in various projects in topics such as parallel I/O, cloud computing, heterogeneous computing, and accelerators for high-performance platforms. He has been involved in six research projects funded by the European Union (such as Repara, Rephrase and IC1035 Cost action NESUS). He currently counts with more than 80 international publications in journal and conference papers.

DOROTHEA VOM BRUCH received the B.Sc. degree from Humboldt University, Berlin, Germany, the M.Sc. degree from the University of British Columbia, Vancouver, Canada, and the Ph.D. degree in physics from Heidelberg University, Germany. She is currently a Postdoctoral Researcher with LPNHE, CNRS/IN2P3, Paris. Having participated in particle physics experiments with TRIUMF, Canada, the Paul Scherrer Institute, Switzerland, and currently LHCb, CERN, her main focus is on high-throughput real-time data selection using GPUs.

J. DANIEL GARCÍA is currently an Associate Professor in computer architecture with the Universidad Carlos III de Madrid, Spain. After working in the industry for several years, he joined the University. Since 2008, he has been the Spanish Head of delegation in the ISO C++ Standards Committee, where he actively participated in the development of the C++11, C++14, and C++17 standards. He has carried out many publicly funded research projects and technology transfer contracts. His current research interests focus on programming models for applications improvement. In particular, his aim is improving both the performance and maintainability of applications.

NIKO NEUFELD received the Ph.D. degree in engineering physics from the University of Technology, Vienna, Austria. He has also studied computer science and mathematical logic. After some work in detector physics, he switched to high-throughput computing and high-speed data acquisition systems. He has been working for the University of Lausanne and the École Polytechnique Fédérale de Lausanne, Switzerland, and is a CERN Staff Scientist, since 2005.