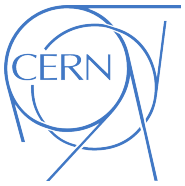


Watering the Seed for Speed

Track Seeding Optimisation on GPU in FTF

A report submitted in fulfilment of the requirements for the

CERN Summer Student Programme

Author

Mathias Moors

Co-Supervisor

Dr. Marco Rimoldi

Co-Supervisor

M.Sc. Aleksandra Poreba

Conseil Européen pour la Recherche Nucléaire (CERN)

Genève, Suisse

Septembre 2024

Abstract

This project focused on optimising track seeding in FastTrackFinder (FTF) on a GPU using Nvidia tools to increase speed without compromising efficiency. The processing time for track seeding within FTF was reduced from 153.45 ± 4.09 ms to 115.97 ± 2.75 ms, while maintaining the same level of accuracy. In addition, a study of multi-processing configurations revealed that performance is primarily influenced by the number of concurrent events rather than the configuration, allowing for the selection of configurations that best suit the needs of other algorithms.

Contents

Chapter 1

The Experimental Setup

1.1 ATLAS

1.1.1 The Large Hadron Collider

The Large Hadron Collider (LHC) is a 27 kilometre collider located at the European Council for Nuclear Research (CERN). The LHC is one of the various accelerators situated at CERN, and is the world's largest and most powerful particle accelerator. The two LHC rings allow the protons to achieve a collision with a centre-of-mass energy of up to 13.6 TeV [1]. This marvel of technology commenced circulating proton beams on 10 September 2008 [2].

Before reaching 6.8 TeV per beam, the protons enter the LHC with an energy of 450 GeV after undergoing acceleration in the CERN accelerator complex. Starting with negative hydrogen accelerated by Linac 4, the electrons are stripped off and the proton passes through the Proton Synchrotron Booster (PSB), the Proton Synchrotron (PS), and the Super Proton Synchrotron (SPS) before concluding their journey in the LHC rings, as illustrated in Figure 1.1. Part of these rings were previously used as a final accelerator, such as the SPS, which was responsible for the discovery of the W and Z bosons.

Lead ions can also be accelerated and follow a similar path, starting at Linac 3 and entering the Low Energy Ion Ring (LEIR) before following the same path as the protons. However, they achieve a smaller centre of mass, 5.36 TeV in November 2023 [3].

The LHC collides protons at four distinct interaction points, where experiments are performed. These are ALICE, CMS, LHCb and ATLAS. Additionally, many other experiments are conducted around the LHC ring, utilising the results of the collisions away from the interaction point.

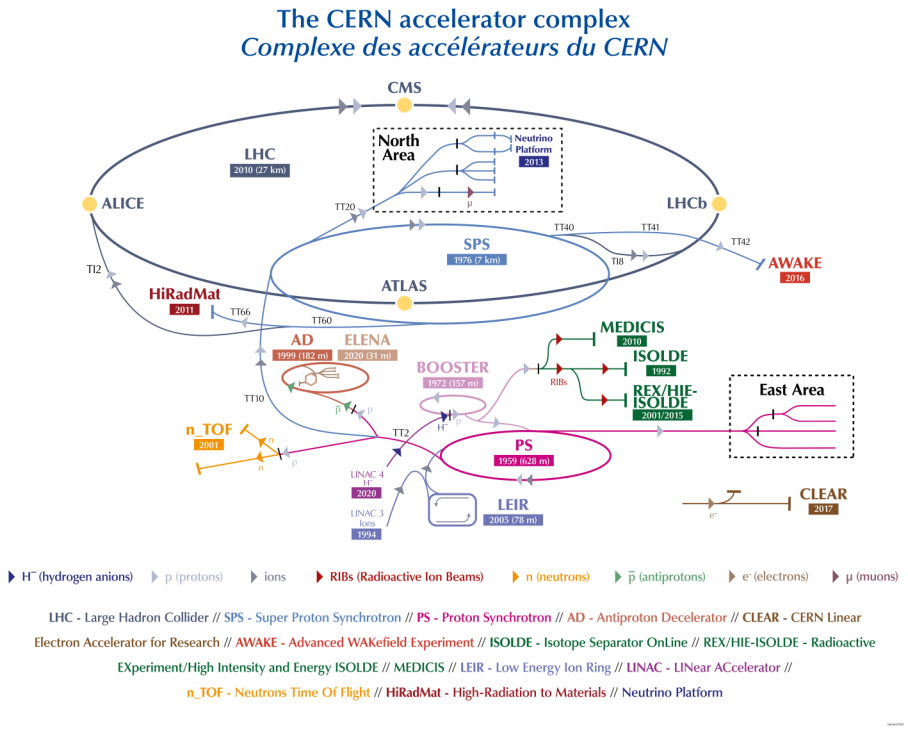


Figure 1.1: The CERN accelerator complex in 2022 [4].

1.1.2 ATLAS Collaboration

ATLAS is one of the two general-purpose experiments alongside CMS. It is the largest detector ever constructed for a particle collider, with dimensions of 46 metres in length and 25 metres in diameter, and a total weight of 7,000 tonnes.

The ATLAS collaboration is now composed of over 5,500 members and almost 3,000 scientific authors from 182 institutions across 42 countries [5].

1.1.3 Sub-Detectors

The ATLAS experiment is constituted of multiple sub-detectors, as illustrated in Figure 1.2. The Inner Detector (ID), which is primarily focused on the reconstruction of the tracks coming from the particles collision and will be the focus of this project. The second subsystem is the calorimeter system, which is further divided into the electromagnetic Liquid Argon calorimeter (LAr), and the tile hadronic calorimeter. Their purpose is to measure the energy of the particles. The Muon system occupies the largest space in the detector and is used to detect and determine properties of muons. Additionally, the magnets, which bend particles

and allow for the determination of their momentum, are divided into the solenoid and toroidal magnet.

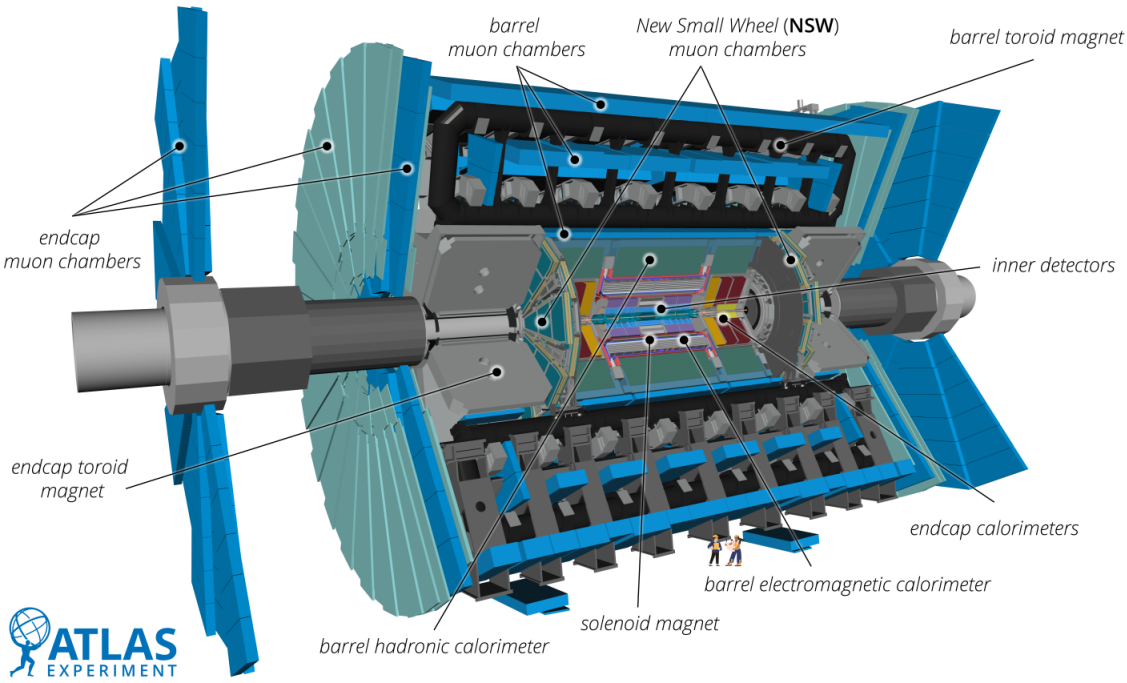


Figure 1.2: Cut-away view of the Run 3 configuration of the ATLAS detector, indicating the locations of the larger detector sub-systems [6].

1.1.4 Inner Detector

The tracker is the sub-detector of interest in this study, and the only one that will be described in detail. As mentioned above, the ID is the primary tracking device, responsible for reconstructing the trajectories of all charged particles. Thanks to the 2 tesla (T) magnetic field generated by the solenoid magnet, the ID has an excellent momentum resolution and the ability to measure the primary and secondary vertices for particles within the pseudorapidity range $|\eta| < 2.5$. The current ID consists of three main components, shown in Figure 1.3. The Pixel Detector, the Semiconductor Tracker (SCT) and the Transition Radiation Tracker (TRT) [7].

The Pixel Detector comprises a single pixel Insertable B-Layer (IBL) detector, added during the Long Shutdown 1 (LS1), which reduced the distance from the beam axis from 5 cm to 3.3 cm. Additionally, there are three pixel outer layers situated between 33.25 mm and 122.5 mm. These layers allow the ID to achieve a precision of 10 μm, utilising 92 million

pixels.

Surrounding the Pixel Detector, the Semiconductor Tracker (SCT) comprises over 4,000 modules with 6 million micro-strips of silicon sensors. These have been designed to detect and reconstruct the trajectories of charged particles produced during collisions. The system's layout has been optimised to ensure that each particle traverses at least four layers of silicon, enabling track measurements with an accuracy of up to 25 µm.

The Transition Radiation Tracker (TRT) is constructed from 300,000 thin-walled drift tubes, each with a 4 mm diameter, filled with a gas mixture and centred by a 30 µm gold-plated tungsten wire, which produces an electric signal when a charged particle passes through the tubes. The barrel section contains 50,000 straws, each 144 cm long, while the end caps each contain 250 straws, measuring 39 cm long, as illustrated in Figure 1.3. This module achieves a precision measurement of 0.17 mm [7, 8].

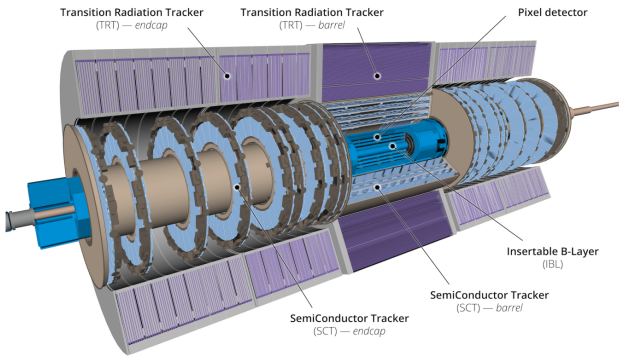


Figure 1.3: Cut-away view of the ATLAS inner detector, which is designed to provide a high-precision reconstruction of charged-particle trajectories [7].

1.2 Trigger And Data Acquisition

The Trigger And Data Acquisition (TDAQ) system is designed to filter and save events for further analysis. It is consisting of two main parts: the Trigger and the Data Acquisition (DAQ) systems. The DAQ system is responsible for collecting data from the read-out of the detectors and converting them into a format suitable for computer processing, storage, and analysis. The DAQ system alone would be sufficient for experiments where proton bunches collide at a low rate. However, the ATLAS experiment has a collision rate of 40 MHz, which requires the full capabilities of the TDAQ system, including the trigger, to manage the data influx. To save all the events, approximately 80 TB s^{-1} of storage would

be required. However, only about 6 GB s^{-1} of storage is available for detector readout and offline computing [7].

Furthermore, the majority of collisions result in events of little interest, involving low-energy interactions. Therefore, it is acceptable to carefully choose a small subset of events from the generated 40 MHz, focusing simply on those that are objects of interest. Defining what constitutes a 'interesting' event can be challenging, as interesting physics often involves high energy, transverse momentum, and more creative solutions such as displaced signatures. Therefore, the trigger system must be both rapid and selective in order to handle 40 million collisions per second without excluding events that could potentially lead to new physics.

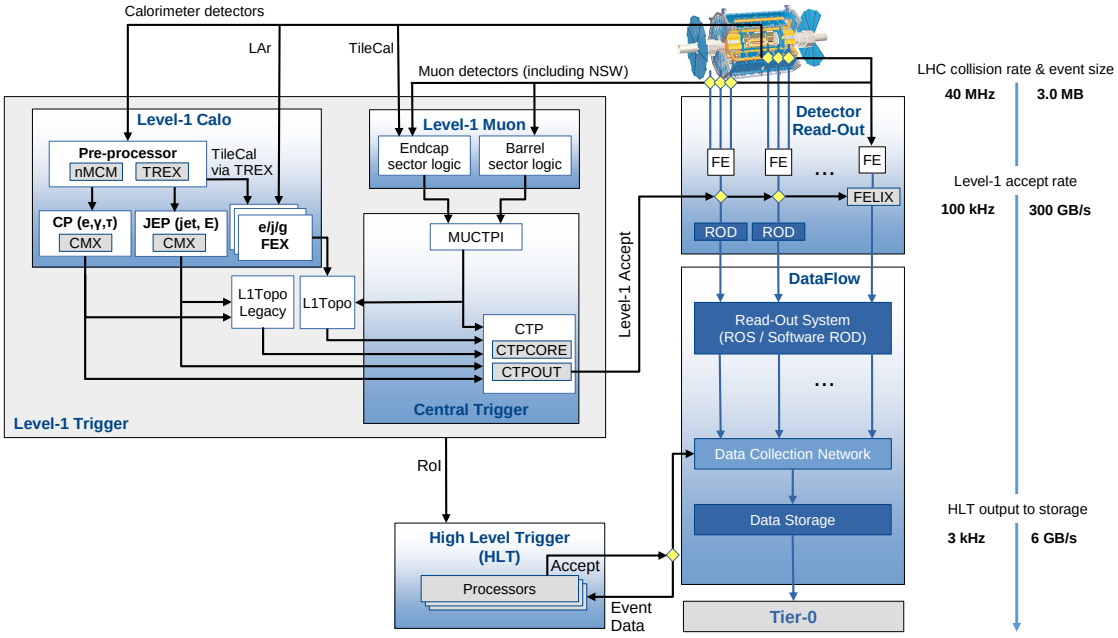


Figure 1.4: Schematic visualisation of the ATLAS TDAQ system during Run 3 [7].

The Trigger system consists of two main components: the Level-1 Trigger (L1) and the High-Level Trigger (HLT), as shown in Figure 1.4. Their purpose is to decrease the event rate from 40 MHz to 100 kHz for the L1 trigger, and then to 3 kHz for the HLT. As a result, out of the 40 million events detected per second by the ATLAS detector, only 3,000 are ultimately selected for further analysis [9]. The L1 trigger is a hardware-based system that operates with a fixed latency to process each bunch crossing uniformly, ensuring synchronisation. In contrast, the HLT processes events within a maximum allowed time frame, which is significantly longer

than the L1 trigger's average processing time. The L1 trigger's latency is approximately 2.5 µs, while for the HLT, the average latency time is about 500 ms [9].

The DAQ system is segmented into two parts: the Detector Read-Out and the Dataflow. The Detector Read-Out gathers information from the various subsystems of the detector, while the Dataflow collects all data from events approved by the L1 trigger. The collected data is then forwarded to the HLT, which accepts and archives it for further analysis, discarding any data that is not approved by the HLT.

1.2.1 High-Level Trigger

The High-Level Trigger (HLT) is a software designed for central processing units (CPUs) with an input rate of 100 kHz and an average processing time of 500 ms. The current HLT system utilises over 55,000 physical cores, specifically equipped with AMD EPYC 7302 processors [9]. During the data taking, the HLT receives Region of Interests (RoIs) from the Central Trigger Processor (CTP) and obtains more detailed event information from the DataFlow system. The HLT enables the processing of more complex information than the basic data received by the L1 trigger. The HLT farm executes advanced trigger algorithms, beginning with fast reconstruction using simplified algorithms, often guided by RoIs. It then proceeds to more accurate reconstruction on the events that pass the first selections. These algorithms are slower, but more closely mirror the offline ones. For instance, the HLT employs data from the Inner Detector to reconstruct tracks and analyse vertices with precision, where the L1 does not handle tracks information. The selected events are processed and saved for further physics analyses.

1.3 High-Luminosity LHC

The LHC has been and continues to be one of the most renowned particle accelerators in the world. Since its first beams running in 2008, the LHC has made a significant contribution to particle physics, including the discovery of one of the most anticipated particles, the Higgs boson, in 2012. However, to advance further and unlock new potential discoveries, the LHC must undergo a major upgrade. The HL-LHC upgrade is designed to achieve an instantaneous luminosity that is 5 to 7.5 times greater than the LHC's nominal value. Figure 1.5 illustrates the projected timeline for the upgrade, which is scheduled to take place between 2026 and 2028 included. Additionally, the figure illustrates the expected increase in total integrated luminosity, which is expected to be an order of magnitude greater than

the total luminosity achieved by the end of Run 3.

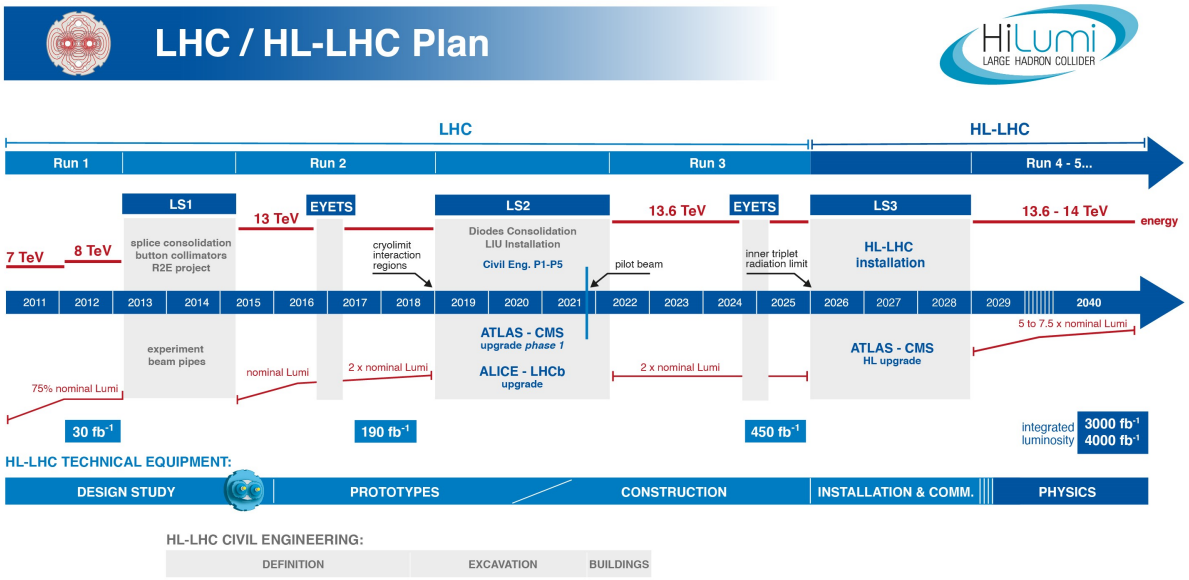


Figure 1.5: Current plan for the LHC/HL-LHC [10].

This potential increase in instantaneous luminosity presents a significant challenge: pile-up, which can be categorised into two types: in-time pile-up and out-of-time pile-up. In-time pile-up occurs when the high density of protons in each bunch results in multiple proton-proton collisions per bunch crossing, rather than a single collision.

During Run 3 of the LHC, there are typically around 60 proton-proton collisions per bunch crossing. After the upgrade, in Run 4, the estimated pile-up is expected to increase to approximately 140 proton-proton collisions per bunch crossing, and to around 200 in Run 5 and beyond [11].

The increased pile-up presents challenges not only for the detector, which must be robust against radiation, but also for the TDAQ and analysis systems, which must handle the greater background noise.

It is important to note that the HL-LHC is not the exclusive contributor to this significant upgrade, it is accompanied by substantial improvements to almost each individual experiment associated with the LHC ring. Scientists from each experiment have invested a considerable amount of time and resources into developing and testing enhancements to prepare the experiments for increased pile-up and to globally improve each subsystem of the

experiment.

1.3.1 ATLAS Phase-II Upgrade

The Long Shutdown 3 (LS3), which is taking place for the HL-LHC installation, is also a decisive period for ATLAS, which will undergo significant changes and upgrades. The ATLAS Detector will undergo several improvements, including a new Inner Tracker (called ITk) and the new High Granularity Timing Detector (HGTD). Additionally, improvements will be made to the muons spectrometer, the electronics for the calorimeter system will be enhanced, and an improved TDAQ system will be implemented [12].

1.3.1.1 Inner Detector Phase II Upgrade

The current ID, made of Pixel Detector, SCT, and TRT, will be completely replaced by a fully silicon-based detector with coverage extending up to a pseudorapidity of 4. The upgraded Inner Tracker will comprise only of silicon pixels and strips, with five pixel layers and five strip layers, providing greater radiation tolerance and higher sensor granularity, as illustrated in Figure 1.6. This significant improvement will enhance track reconstruction, primary and secondary vertex measurements, and allow for more efficient pile-up removal, which is essential in this environment.

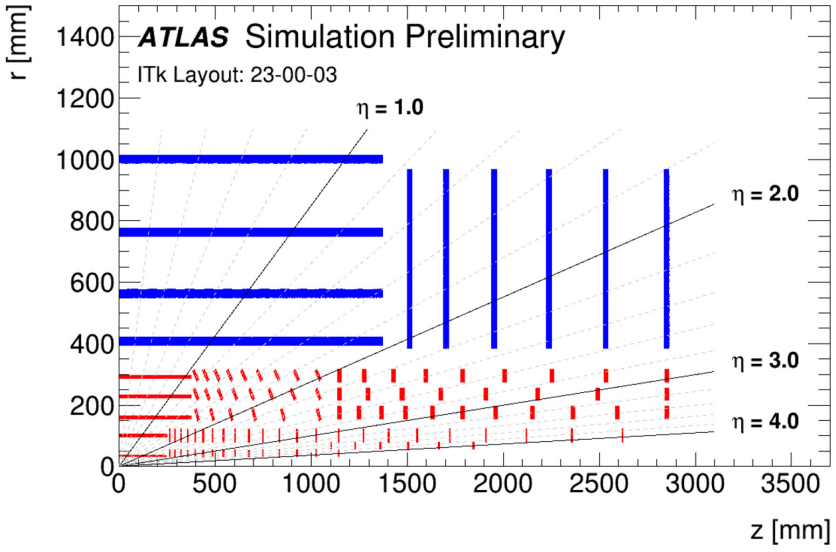


Figure 1.6: A schematic depiction of the ITk layout, with the strip detector shown in blue and the pixel detector in red. The horizontal axis represents the beamline, with zero indicating the nominal interaction point. The vertical axis shows the radius measured from the interaction point [13].

1.3.1.2 TDAQ Phase II Upgrade

The hardware trigger has been renamed from L1 to L0 and will have an extended maximum latency, increasing from 2.5 µs to 12.5 µs **??**. This extension allows more complex algorithms to run on the L0 level, aided by the addition of a new global trigger. The L0 output rate is 1 MHz, almost ten times that of the L1 trigger during Run 3. The HLT has also been renamed the Event Filter (EF), and is supposed to process the events at a rate of 1 MHz to 10 kHz, keeping the processing time per event between about 0.1 s and 1 s.

The main difference between the detector outputs of Run 3 and future runs is the increase in the number of pile-ups, resulting in a higher occupancy of the detector. This increase results in more complex events, and the HLT processing time increases exponentially with the pile-up due to the combinatorial nature of the majority of the algorithm. Furthermore, the time budget remains relatively unchanged.

One way to improve the algorithm without affecting the latency of the difference system is to use as much low-latency hardware as possible, such as a GPU for the event filter. This change in hardware could achieve the same latency with more pile-up and more complex algorithms by parallelising them, avoiding the exponential increase in time.

Chapter 2

Fast Track Finder

2.1 Track Reconstruction In ATLAS

The particle track reconstruction (tracking) is a crucial step in reconstructing the full event, both in the Trigger system, where it helps to decide whether to keep the event, and in the analysis phase. Tracking is also the slowest step in the reconstruction process due to the large input of detector hits and the complexity of the algorithms involved in identifying them. Track reconstruction is essential for determining the momentum of the particle, finding primary and secondary vertices and for particles identification.

In ATLAS, the track reconstruction used in Run 3 consists of two stages: ATLAS Primary Tracking, which works from the inner to the outer detectors, and ATLAS Back-Tracking, which works in the opposite direction, from the outer to the inner detectors. The different steps of these two stages are illustrated in Figure 2.1. Only the ATLAS Primary Tracking will be explained here, as it is the relevant part for this discussion.

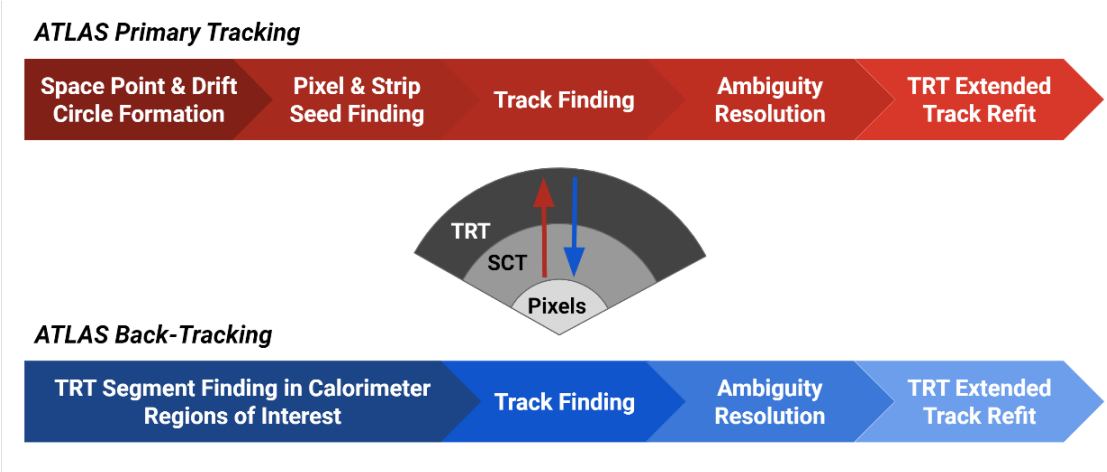


Figure 2.1: ATLAS Track Reconstruction Scheme [14].

2.1.1 ATLAS Primary Tracking

ATLAS Primary Tracking is divided into five steps, beginning with the creation of space points (SPs) from raw detector information and concluding in full track reconstruction.

1. **Space Point & Drift Circle Formation**: The first step in tracking is the creation of SPs, which are derived from hits in the ID (pixels, microstrips, and straw tubes).

2. **Pixel & Strip Seed Finding**: Once the SPs have been created, the next step is to group them into sets of three (triplets) that are expected to be the starting points of tracks (seeds). These triplets can be formed either entirely within the pixel detector or including the SCT.

3. **Track Finding**: In this step, search roads are constructed, which are sets of detector modules likely to contain clusters compatible with the seed. The search road reduces the computational time by restricting the search to a subset of modules. A Kalman filter is used to extend the SPs along the search road, and compatibility with the calorimeter is also checked.

4. **Ambiguity Resolution**: After track finding, ambiguities may arise, such as overlaps between track candidates or combinations of unrelated clusters. An ambiguity solving algorithm is used to assign scores to track candidates based on their quality, removing low quality tracks that share clusters with high quality tracks.

5. **TRT Extended Track Refit**: Finally, an extension into the TRT sub-detector is attempted. A road search is performed in the TRT starting from the position estimation of the track candidate. A Kalman filter is again used to build candidate extensions. Once TRT hits are added to the tracks, a full track refit is performed using a global χ^2 fitter, improving measurements on the track, particularly for momentum and particle identification [14].

2.2 Fast Track Finder

Fast Track Finder (FTF) is an algorithm used to select candidate tracks in the online HLT system. The algorithm consists of three main steps: track seeding, track extrapolation, and the Kalman filter. It has been developed to generate track candidates early in the HLT system. The design philosophy of FTF prioritises efficiency over purity, as track candidates can be refined in later stages [15].

As discussed in Section 1.3.1.2, due to increased pile-up and the constraints of the fixed time budget, modifications to the algorithm are necessary. One possible modification is adapting Track Seeding to a GPU-based algorithm, using the parallelisation capabilities of GPUs to improve performance.

2.3 Graphics Processing Unit

2.3.1 CPU vs GPU

Before discussing the advantages of using a GPU for track seeding, it is important to understand how a GPU functions in comparison to a CPU, and why this shift in device could lead to performance improvements. Figure 2.2 provides an overview of the key differences between the two.

The CPU, on one hand, contains a small number of powerful cores, typically ranging from 2 to 64 in high-end models. This architecture is well-suited for complex sequential processing. The GPU, on the other hand, is composed of thousands of smaller cores, making it ideal for parallel algorithms, as each core can perform a computation simultaneously. If an algorithm is designed to be parallelised, running it on a GPU can result in significantly faster execution compared to a CPU.

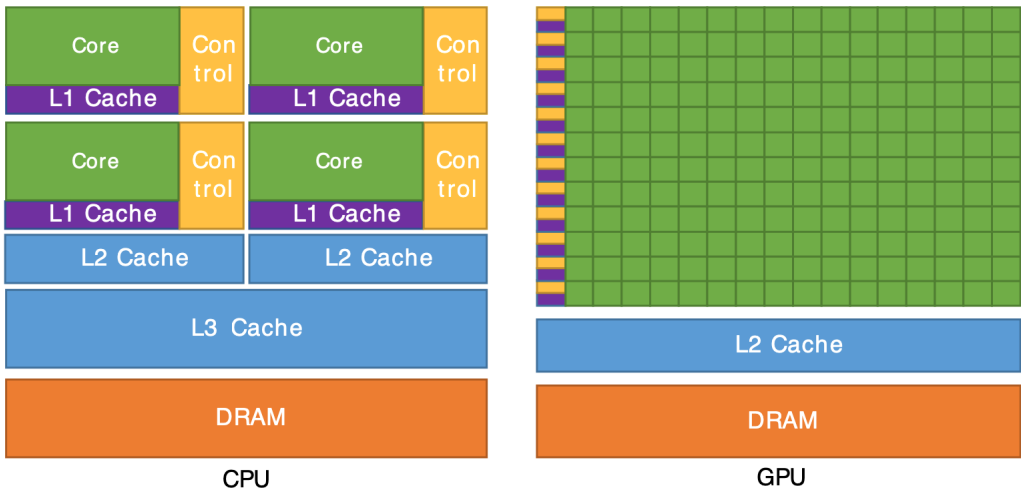


Figure 2.2: Illustration of the CPU/GPU architecture [16].

2.3.2 GPU Architecture

A GPU is composed of multiple Streaming Multiprocessors (SMs), such as the 64 SMs in the NVIDIA RTX A5000, which serve as fundamental building blocks of the GPU. Each SM has its own memory, known as the L1 cache. Inside each SM are threads (1536 in the current example), which represent the smallest unit of execution responsible for a specific task as part of a larger parallel computation. Each thread executes the same kernel but works on different data, enabling highly parallel processing. Threads have allocated memory known as registers, which store temporary variables and intermediate computational results. These threads are grouped into blocks that collaborate on common tasks using shared memory, a very fast memory that allows quick communication between threads.

Additionally, each SM can access the L2 cache, which is shared among all SMs and used for inter SM communication or to retrieve data from higher-level memory. Finally, the GPU memory, which is larger but slower than the L2 cache. Before executing each kernel, the GPU anticipates memory needs to provide as much data as possible to lower-level memory for faster access.

2.4 Track Seeding on GPU

After discussing the advantages of GPUs over CPUs and the GPU architecture, it is now time to explain the different kernels within the FTF track seeding on GPU for ITk. The process consists of four kernels, each representing a different step of the algorithm: doublet counting, doublet making, doublet matching, and triplet confirmation.

2.4.1 Doublet Counting

The first kernel, called 'Doublet Counting', calculates the number of doublets, which are pairs of SPs. The aim is to consider all the possible doublets, using all SPs in the Pixel detector, and then eliminate those that do not meet specific criteria. These criteria are evaluated using cuts, and doublets that do not have the required characteristics to pass the cuts are discarded.

1. **Doublet Coordinate Cut**: This cut ensures that the coordinate of the second space point (either z or r) lies within valid boundaries based on the first space point, discarding invalid doublets.

2. **Doublet Length Cut**: This cut ensures that two SPs are neither too close, which could indicate noise or low-energy tracks, nor too far apart, which could indicate that there is another pixel layer between them, meaning that the same track would be considered by another pair.

3. τ **Cut**: Ensures that the track formed by the doublet lies within a valid range of cotangent angles, which correspond to the cotangent of the polar angle θ. The cotangent angle (τ) must not be too large to ensure the track remains within the detector's sensitive region, $|\eta| < 4$.

4. **Pixel Width Cut**: This cut checks if the width of the pixel cluster in the barrel corresponds to the angle of the doublets, based on the τ angle at which the track crosses the layer.

5. **Z Cut**: Two cuts are applied based on the Z coordinate:

   (a) Z_0 **Cut**: This cut checks whether the z-coordinate of the extrapolated track's point, where the particle track would intersect the beamline, is within the expected range (beamspot). This ensures that the doublet corresponds to a track originating from the collision region.

   (b) **Outer Z Position Cut**: This cut checks the extrapolated z-position of the track at the outer radius of the detector. It ensures that the z-position is still within the detector's bounds, ensuring that the doublet remains within the sensitive region of the detector.

6. **Curvature Cut**: Filters doublets based on their curvature, keeping only particles with transverse momentum (p_T) above a certain threshold.

In this kernel, the algorithm counts the number of inner and outer doublets to allocate the exact amount of memory needed for the next kernels.

2.4.2 Doublet Making

This kernel is structured similarly to the doublet counting kernel, with the same cuts and criteria, but it now has the number of doublets, allowing for efficient memory allocation for the output container. These two kernels are essential for good GPU resource management, especially when processing multiple events simultaneously, ensuring scalability and efficiency.

2.4.3 Doublet Matching

The doublet matching kernel is responsible for combining doublets to form triplets. A triplet is created by matching two valid doublets with a middle SP. The goal of this kernel is to match doublets such that the first SP comes from the first doublet, the middle point from both doublets, and the third point from the second doublet. To ensure that the triplet is physically meaningful and can be interpreted as part of a track, several cuts are applied:

1. **Polar Angle Matching**: The first cut is checking if two doublets can form a track based on the cotangent of their polar angles.

2. **Breaking Angle cut**: The second cut checks if the polar angle of the considered track would be possible given the uncertainties coming from the spacepoint measurement and the multiple scattering effects.

3. **Transverse Momentum and Trajectory Cut**: This combined cut first checks the estimated transverse momentum of the triplet, discarding those below a certain threshold. It then re-checks the cotangent angle difference using the estimated transverse momentum to ensure that the geometry of the triplet is consistent with p_T.

4. **d0 Cut**: This cut removes triplets with large transverse impact parameters, indicating that they are far from the beamline in the transverse plane and unlikely to be part of a primary track.

5. ϕ **Region Cut**: If tracking is restricted to a specific Region of Interest (RoI), this cut ensures that triplets lie within the desired range. If full-scan mode is enabled, no cut is applied.

After these cuts, the quality of the triplets is evaluated based on the d0 parameter, with tracks closer to the interaction point being assigned a higher score. A check for duplicates is then performed. Having duplicates indicates that the seeds are more likely to be a part of the track since multiple part of it were reconstructed. Finally, only the best triplets are kept, and those without duplicates are rejected.

2.4.4 Triplet Confirmation

The final kernel, called 'Triplet Confirmation', aims to validate and confirm that the triplets are consistent with plausible track candidates. The verification only happens for triplets entirely in the endcap region, because tracks in the barrel will only hit few possible layers

and are unlikely to have duplicates.

This verification compares each triplet with all the others. One by one, it checks if the triplet shares two SPs with any other triplet. If it finds another triplet with two shared SPs, further checks are performed, such as a curvature check and a transverse momentum check, to verify if they can be a part of the same track.

In the end, the triplet has to be confirmed by at least two other triplets to be considered as a valid seed.

Once all the triplets have been processed, track seeding is complete and the track seeds are saved for the next tracking step.

Chapter 3

Processing Time Improvements

3.1 Methodology

The Trigger system is one of the most intriguing aspect of ATLAS, as it necessitates a reconciliation between rapidity and efficiency. The central objective of the project was to enhance the timing of the four kernels of the GPU Track Seeding while maintaining their efficiency.

3.1.1 Setup

For the study of the four kernels, the Athena release 25.0.12 was used[1]. The reconstruction was run with Reco_tf command used during this study is the one below:

```
Reco_tf.py --maxEvents=100 --inputRDOFile /cvmfs/atlas-nightlies.cern.ch/
   repo/data/data-art/PhaseIIUpgrade/RDO/ATLAS-P2-RUN4-03-00-00/mc21_14TeV
   .601229.PhPy8EG_A14_ttbar_hdamp258p75_SingleLep.recon.RDO.
   e8481_s4149_r14700/RDO.33629020._000047.pool.root.1 --outputAODFile AOD
   .root --CA --steering doRAWtoALL --preInclude InDetConfig.
   ConfigurationHelpers.OnlyTrackingPreInclude --preExec "flags.Tracking.
   useITkFTF=True;flags.Tracking.doITkFastTracking=True;flags.Trigger.
   InDetTracking.doGPU=True" | tee
```

which incorporate the use of the Fast Track Finder algorithm with the track seeding on GPU in a standalone setup. These tests have been done with 100 simulated $t\bar{t}$ events.

The present study was primarily conducted on the group testbed GPU cluster, which is equipped with an RTX A5000 GPU featuring 64 streaming multiprocessors and 8,192 Nvidia CUDA cores. Further specifications can be found in Appendix A. This GPU was chosen over

[1] Athena is a software used in ATLAS for reconstruction, analysis and simulation [17].

the lxplus-GPU cluster, which uses the slower Tesla T4, and the A100 SXM4 40 GB, due to the high demand on the GPUs and the slower memory transfer speeds associated with its virtualised CPU.

3.1.2 Tools

The timing of these kernels has been monitored using two distinct methods. The first method utilise the Athena provided "trigDumpTimers.py" script, which summarises the timing of the FTF steps based on the monitoring histograms, filled during the job execution. In our case, the time used is `TIME_TRIPLETS`, covering the four kernels and various memory operations before and after them. The second method employs the Nvidia profiling tool, Nvidia Nsight System [18], which offers greater precision than the previous method. This method provides the timing of each kernel, along with information such as the average, median, and standard deviation. Additionally, it enables the visualisation of all operations on the timeline. To achieve this last method, this command has been added to the first part:

```
nsys profile --trace=cuda,nvtx,cublas,cublas-verbose,cusparse,cusparse-
    verbose,mpi,oshmem,ucx,nvvideo,osrt,cudnn,opengl,opengl-annotations,
    openacc,openmp,vulkan,vulkan-annotations --output=profile
```

which will create an nsys-profile file containing all the information discussed above, that can be browsed with Nvidia provided tool.

Finally, to gain the insight into the kernels and identify potential areas for optimisation, it is necessary to employ methods that provide more comprehensive information than simply timing data and the operations performed during the execution of a program. Another Nvidia tool, called Nvidia Nsight Compute [19], is specifically designed to study the kernel. It provides details such as computational throughput, memory throughput, duration, and other key metrics like pipeline utilisation, which measures how efficiently the GPU's resources are used. These tools can be used by replacing the 'nsys' command with this new one:

```
ncu -o profile --set full
```

which will also give us a file with all the information we need about these kernels. An important thing to know is that this tool has made the processing time much slower, as it takes a lot of time to process the kernel to get all this information.

3.2 Initial Performance

At the start of this project, it was important to measure the time of each of the kernels, as well as the total triplet timing, in order to have a time recapitulation.

Stage	Counting	Making	Matching	Confirmation	Kernels	Total
Time (ms)	13.20 ± 2.83	13.25 ± 3.00	41.40 ± 14.33	34.27 ± 15.45	102.80	153.45 ± 4.09

Table 3.1: Timing of the different kernels in milliseconds (ms), where the kernel times are measured using NVIDIA Compute and the total time, including memory transfer, is measured using the Athena script `trigDumpTimers.py`. The operations are performed on the RTX A5000.

It can be seen in Table 3.7 that the major part of the time is spent on the last two kernels as well as for the memory related operations. These kernels were examined for potential improvements.

3.3 Floating-Point Computation

By examining the kernel performance using NVIDIA Compute tools, it was observed that a significant percentage of pipeline utilisation is spent on floating point 64 (FP64) operations, where each variable in the calculation uses 64 bits, specifically, about 75% of the computations for doublet counting, 71% for making, 8% for matching and less than 1% for confirmation. While this may seem harmless, double precision operations are slower and not always necessary. On the NVIDIA RTX A5000, the ratio of single-precision (FP32) to double-precision (FP64) performance is approximately 64:1, meaning that the GPU can perform 64 single-precision operations in the time it takes to complete just one double-precision operation. In this context, the use of double precision operations is unnecessary because the kernels do not perform tasks that require high precision. By modifying the kernels to ensure that all operations and variables use a maximum of 32-bit floating point, kernel execution times are significantly reduced while maintaining the same level of computational accuracy.

This modification represents an improvement of 15.48 ms, which is 15% on the total kernels time and of 16.12 ms, which is 10% on the total triplets time, without affecting the efficiency of the seeding.

More details on the modification made to achieve this result can be found in the Appendix B.

Stage	Counting	Making	Matching	Confirmation	Kernels	Total
Double Time (ms)	13.20 ± 2.83	13.25 ± 3.00	41.40 ± 14.33	34.27 ± 15.45	102.37	153.45 ± 4.09
Float Time (ms)	7.35 ± 1.54	7.98 ± 1.85	37.10 ± 13.10	34.46 ± 15.59	86.89	137.33 ± 3.73

Table 3.2: Timing of the different kernels in milliseconds (ms), where the kernel times are measured using NVIDIA Compute and the total time, including memory transfer, is measured using the Athena script `trigDumpTimers.py`. The operations are performed using float operations on the RTX A5000.

3.4 Analysis of Cut Selections

After implementing the initial improvements, a detailed study was conducted to assess the impact of each cut within the kernels. Both the doublet counting and doublet matching processes were examined closely. Since the cuts applied during doublet counting and doublet matching are identical, only one set of cuts was evaluated in depth.

In this study, the `atomicAdd` function was used to count the number of doublets and triplets after each cut, as well as to measure the time taken for each step. The use of `atomicAdd` allows each block of threads to add its contribution independently, without interfering with other blocks [20].

3.4.1 Doublet Counting

The cuts applied during doublet counting have been previously discussed in Section 2.4.1. In table 3.3 below is a summary of the time analysis for each selection criterion during the doublet counting process:

Selection	Rejection (%)	Time (s)	Time (Normalised) (ns)
Doublet Coordinate Cut	36.55	11.30	84
Doublet Length	29.55	21.62	229
τ Cut	16.56	5.26	67
Pixel Width Cut	2.39	1.65	21
Z_0 Cut	59.61	2.24	72
Outer Z Position Cut	0.00	1.15	37
Curvature Cut	86.01	1.09	250

Table 3.3: Doublet Counting Time Analysis, showing rejection rates, time in seconds, and normalised time in nanoseconds, conducted on the RTX A5000 GPU.

The timing in seconds is higher than the kernel timing due to the methodology used here.

By using `atomicAdd`, each block adds its own timing, and even though the blocks run in parallel, they accumulate their individual times. This total time is normalised by the number of events contributing to the timing, giving the time per event. In this kernel, the two most time-consuming cuts are the Doublet Length Cut and the Curvature Cut. It is believed that the Curvature Cut could be optimised by altering how it is performed, using a simpler calculation than the current one. The ratio between $d\phi$ and dr could provide sufficient curvature precision to achieve the same rejection accuracy as the current method. This change was not made in this study.

3.4.2 Doublet Matching

The following table 3.4 presents the time analysis for the various cuts applied during the doublet matching process:

Selection	**Rejection (%)**	**Time (ms)**	**Time (Normalised) (ns)**
Polar Angle Matching	76.43	829	57
Breaking Angle Cut	89.17	98	62
Transverse Momentum (pT)	11.04	213	153
Trajectory Cut	44.49	45	58
d_0 Cut	1.59	30	40
ϕ Region Cut	0.00	7	10
Quality & Duplicates	85.54	181 000	1700

Table 3.4: Doublet Matching Time Analysis, showing rejection rates, time in seconds, and normalised time in nanoseconds, conducted on the RTX A5000 GPU.

During the doublet matching process, confirming triplets by checking for duplicates was by far the most time-consuming part of the kernel, significantly contributing to the overall execution time. The first two cuts, Polar Angle Matching and Breaking Angle Cut, are major contributors to the rejection rate, alongside the Quality & Duplicates Cut, which is a special case and only applies after all other cuts.

The order of the cuts is important: for instance, the Breaking Angle Cut depends on the outcome of the Polar Angle Matching, so they cannot be reversed, even though the second cut has a better rejection rate than the first. The same applies to the relationship between the Transverse Momentum Cut and the Trajectory Cut. After the d_0 Cut, which takes relatively little time, the ϕ Cut shows no effect here since it's a full scan. At the end, the Quality & Duplicates Cut consumes a considerable amount of time compared to the others, and if any part of the kernel is to be improved, this is the one to focus on.

3.5 Threading Configuration

After this initial improvement and the selection study, a closer analysis was conducted on the two kernels that consumed the most time: doublet matching and triplet confirmation. While looking for possible improvements, it was found that most of the time the different parts of the GPU were idle, meaning they were waiting for another part of the block. It can happen when a part of a block is waiting for some other threads before starting the next step of the kernel. This appears if the calculations between the threads are not equal, or it could be due to the wrong threads or blocks architecture.

A possible improvement to the doublet matching kernel is to change the way the two kernels work by going from 1D to 2D. This modification could bring a huge improvement, since the longest step of the doublet matching would be perfect for 2D, since it is comparing tracks between themselves. During the experiments with the algorithm conversion, it has been seen that just by changing the number of threads within a block could be a huge improvement. The results of the experiments with different threads configurations are in the tables 3.7 and 3.6.

Blocks/Threads Per Blocks	**128**	**256**	**512**	**1024**
512	-	22.66 ± 7.60	-	-
1024	22.17 ± 7.50	21.50 ± 7.30	21.02 ± 7.09	36.89 ± 13.05
2048	22.12 ± 7.53	20.63 ± 6.95	20.44 ± 6.88	-
4096	-	20.43 ± 6.91	-	-

Table 3.5: Timing of the different kernels in milliseconds (ms), where the kernel times are measured using NVIDIA Compute and the total time, including memory transfer, is measured using the Athena script `trigDumpTimers.py`. The operations are performed using float operations and modified thread configurations on the RTX A5000.

Blocks/Threads Per Blocks	**128**	**256**	**512**	**1024**
512	-	30.61 ± 13.24	-	-
1024	27.25 ± 11.60	26.58 ± 11.14	30.20 ± 13.45	34.29 ± 15.46
2048	27.27 ± 11.56	26.07 ± 10.99	29.72 ± 13.16	-
4096	-	26.27 ± 11.11	-	-

Table 3.6: Timing of the triplet confirmation kernels in ms, including the total time with memory transfer, using float operations and modified thread configuration on RTX A5000.

The first major conclusion that can be drawn is the significant difference in performance when using 1024 threads per block compared to other configurations. This setup is almost twice as slow as the fastest one. While this was the kernel configuration used, it does not align well with the GPU utilised in this performance test. The GPU used has a maximum number of 1536 threads per SM. When the number of threads per block is set to 1024, the GPU can only process one and a half blocks at a time. Inside a block, execution occurs in parallel, and synchronisation between threads is sometimes required. This means that the half-processed block must wait for its remaining portion to catch up before continuing, which may explain the significant performance drop with 1024 threads per block. On the other hand, if the number of threads per block is too small, the CPU must launch more blocks, potentially wasting time due to overhead.

The second conclusion relates to the number of blocks. Increasing the number of blocks for the doublet matching kernel improves performance, which may seem counterintuitive given that the total number of threads for the GPU is $1536 \times 64 = 98,304$, which is smaller than $256 \times 512 = 131,072$. However, experiments show that using configurations such as 256×2048 or 256×4096 yields better kernel times. The reason is that the over-utilised GPY performs better, ensuring that all its threads are occupied at any given time.

The information on how the code was modified for this study can be found in the Appendix C.

3.6 Final Results

After applying those two improvements, it is useful to observe how the kernel performance has improved. The details are shown in the table below:

Stage	Counting	Making	Matching	Confirmation	Kernels	Total
Double Time (ms)	13.20 ± 2.83	13.25 ± 3.00	41.40 ± 14.33	34.27 ± 15.45	102.37	153.45 ± 4.09
Float Time (ms)	7.35 ± 1.54	7.98 ± 1.85	37.10 ± 13.10	34.46 ± 15.59	86.89	137.33 ± 3.73
Threads Time (ms)	7.35 ± 1.54	7.98 ± 1.85	20.43 ± 6.91	26.07 ± 10.99	61.83	115.97 ± 2.75

Table 3.7: Timing of the different kernels in milliseconds (ms), where the kernel times are measured using NVIDIA Compute and the total time, including memory transfer, is measured using the Athena script `trigDumpTimers.py`. This summary table includes total time for double and float operations, and threading, using the RTX A5000.

As a result, the total time for the four kernels decreased from 102.37 ms to 61.83 ms,

representing an improvement of 40.54 ms, or 39.60%. Additionally, by including the kernel execution time as well as the memory operations before and between kernels, the total improvement amounts to 37.48 ms, or 24.42%, going from 153.45 ms to 115.97 ms.

Of course, these two changes did not affect the memory management before and after the kernels, which explains why the difference in time between the total kernel time and the total triplet time remains approximately the same. However, this allows us to estimate the time the algorithm spends on such operations, which is about 50 ms, a significant amount, especially considering the kernel timings have been improved. Out of the total 115.97 ms, only 53% of the time is spent on the kernels themselves. A more in-depth study of the memory transfers outside the kernels may be necessary to reduce the contribution of memory management and kernel launch operations during track seeding.

Chapter 4

Multi-Processing

4.1 Methodology

The second task of this project was to study the scaling of track seeding. In other words, the aim was to analyse how the execution time of the kernels, and the track seeding process as a whole, changes as a function of the number of events processed in parallel. However, before rushing into the scaling analysis, it is important to understand how events are distributed within the HLT system.

4.1.1 Framework Overview

To better understand how events are distributed, the diagram in Figure 4.1 illustrates the four main steps involved in the process.

The first step is the High-Level Trigger Supervisor (HLTSV), which is responsible for distributing RoIs from the L1 trigger [21]. The HLTSV coordinates the flow of data to the various processing units in the system.

The second step is the mother process, the HLT Multi-Processing Unit (HLTMPPU). This process receives events from the HLTSV and forks into multiple child processes. Each of these child processes operates independently but inherits the configuration from the mother process, in this case, the Athena Multi-Threading (AthenaMT) framework. These child processes share the number of the event slots to define how many events can be processed in parallel by one fork. Additionally, each child process is assigned a number of CPU threads that are used to execute operations within the available event slots.

Parallelisation is essential to improve the speed of event processing in the HLT. Modern GPUs have enormous computing power and memory capacity, allowing them to handle

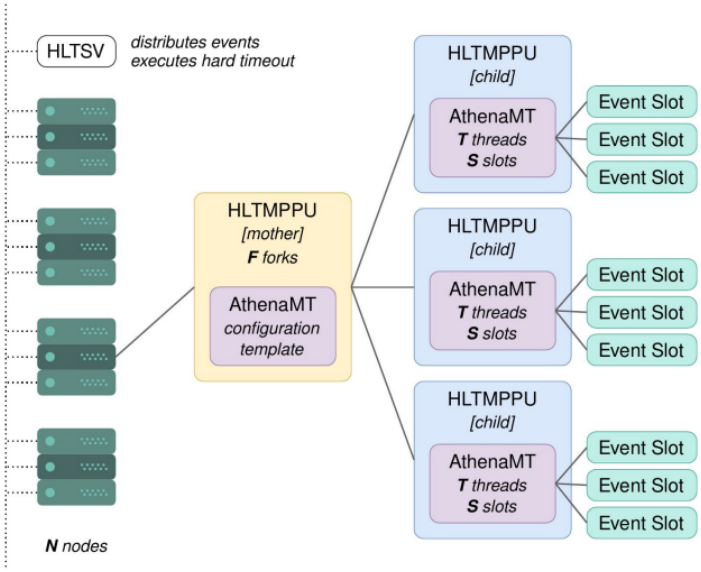


Figure 4.1: Event distribution with forks, threads, and slots in the HLT system.

multiple processes simultaneously. To fully utilise the power of the GPU, events are split across multiple processors and threads, significantly reducing the overall processing time.

The combination of multi-processing and multi-threading ensures that the system operates at full capacity, handling a high throughput of events and enabling the HLT to achieve higher processing rates.

4.1.2 Setup

The setup for this study was noticeably different from the previous one, though the Athena version used remained 25.0.12. Instead of a standalone job, FTF is run as a part of the offline event reconstruction. The setup can be replicated with a following command:

```
athena.py --imf --nprocs=x --threads=y --concurrent-events=z --evtMax=100
   --filesInput=/cvmfs/atlas-nightlies.cern.ch/repo/data/data-art/
   PhaseIIUpgrade/RDO/ATLAS-P2-RUN4-03-00-01/mc21_14TeV.601229.
   PhPy8EG_A14_ttbar_hdamp258p75_SingleLep.recon.RDO.
   e8514_s4345_r15583_tid39626672_00/RDO.39626672._001121.pool.root.1
   TriggerJobOpts/runHLT.py Trigger.doRuntimeNaviVal=True ITk.doTruth=
   False Tracking.doTruth=False IOVDb.GlobalTag="OFLCOND-MC21-SDR-RUN4-02"
    Exec.FPE=1 Trigger.InDetTracking.doGPU=True Detector.GeometryITk=True
   > athena.log 2>&1
```

The tests were performed with a $t\bar{t}$ sample containing 100 events. In this case, the number

of processors, threads and concurrent events were modified and examined. Different GPUs were used in this test: the RTX A5000 and the Tesla T4. The `trigDumpTimers.py` script was used as before, but this time `nsys` was not used. The RTX A5000 was used exclusively, meaning that no one else was using the GPUs when the tests were run. For the Tesla T4, which is a GPU in the lxplus-gpu cluster, the GPU was shared between several users, making it difficult to assess performance without this shared use.

4.2 Results

As previously stated, this study explores the behaviour of three parameters: forks (the number of child processes), threads (the number of CPU threads per child), and slots (the number of concurrent events processed per process). To determine the total number of simultaneous event processes, the calculation is simple: multiply the number of child processes by the number of slots. In this project, configurations processing between 2 to 16 simultaneous events were examined with the aim of maximising application throughput, finding the configuration that provides the highest events per second, or at least identifying the type of configuration that performs best.

4.2.1 Application Throughput

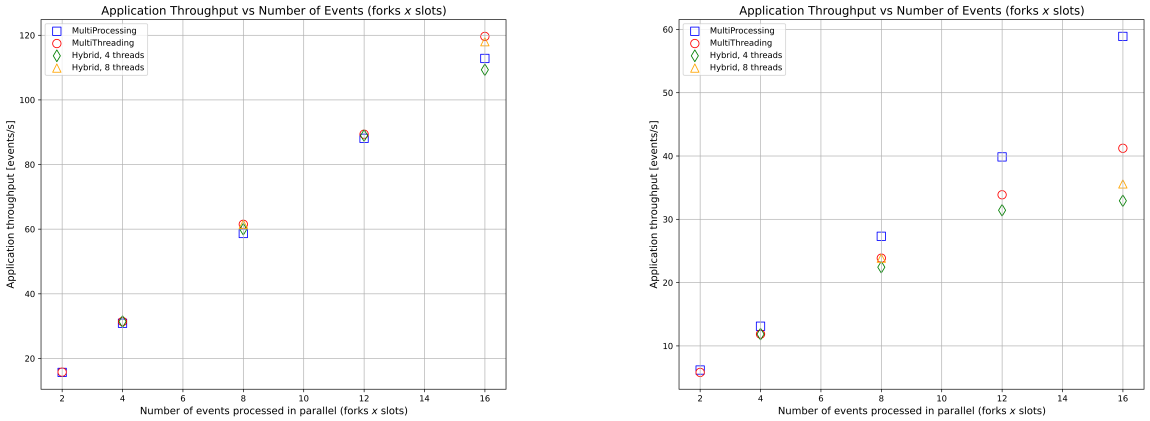
This study tested four different types of configurations:

1. **Multi-Processing**: Uses only child processes, with one thread and one slot per process.
2. **Multi-Threading**: Uses only threads and slots, with a single child process. Here, the number of threads equals the number of slots.
3. **Hybrid, 4 Threads**: The number of threads and slots is fixed at 4, while the number of child processes varies.
4. **Hybrid, 8 Threads**: Similar to the previous configuration, but with 8 threads and 8 slots.

The particularity of these configurations is that the number of threads is always equal to the number of slots. This was done to optimise resource usage by ensuring that each event has a dedicated thread to launch the kernel and manage its operations. This avoids situations where events are waiting for instructions and prevents the allocation of more threads than

there are events, which, while potentially beneficial for certain operations, would increase resource consumption unnecessarily.

The results of these configurations on the two GPUs are shown in the plots below:



(a) RTX A5000 (b) Tesla T4

Figure 4.2: Measuring the throughput with different configurations

The behaviour of the two GPUs is not identical. On one hand, for the RTX A5000, the graph indicates that the application throughput primarily depends on the number of events processed in parallel, rather than on the specific configuration used, as shown in Figure 4.2a. This finding is advantageous, as it allows flexibility in choosing configurations for other algorithms that may behave differently based on the setup. Moreover, it is notable that the application throughput scales linearly with the number of simultaneous events processed, at least up to 16 parallel events.

For the Tesla T4, it can be observed that the multi-processing configuration achieves the highest application throughput starting from 8 simultaneous events, as illustrated in Figure 4.2b. The gap between the multi-processing configuration and the others increases as the number of simultaneous events grows, and it is the only configuration that maintains a linear scaling. The other configurations exhibit a negative second derivative in their scaling. Interestingly, the multi-threading configuration comes in second, while the hybrid configuration performs the worst. It is important to note that the application throughput of the Tesla T4 is at least half that of the RTX A5000.

The conclusions might differ if the study were extended to 32 events, particularly for the RTX A5000. However, this was not conducted here due to several issues encountered with

the GPU when saturated with a high number of parallel events.

4.2.2 Memory Consumption & Usage

4.2.2.1 RTX A5000

Why does the configuration not affect the results? In the case of multi-processing, each event is assigned to a separate process, meaning each event has its own memory allocation since each child process operates in its own memory space. This increases overall memory usage. However, in this study, memory consumption was not an issue due to the relatively low number of concurrent events.

In contrast, multi-threading consumes significantly less memory because the events share memory within a single process. However, multi-threading is not faster because the events being processed simultaneously do not need to share information with one another which would create additional overhead of communication between processes. Similarly, it is not slower either, because the number of simultaneous events is not large enough to saturate the shared memory within a child process.

For the hybrid configurations, the results follow the same linear trend as multi-threading and multi-processing. This makes sense, as these hybrid configurations are a combination of the two methods. In certain cases, the hybrid solution might be preferable because it can use fewer processes than pure multi-processing (if memory saturation becomes a concern) and fewer threads than pure multi-threading (if the shared memory of a single process risks being overwhelmed). However, in this case, neither of these issues appear to be a limiting factor, which explains why all four configurations ultimately achieve the same application throughput.

These explanations could have been verified by checking the GPU's memory usage and consumption. Unfortunately, this was not possible in this study. However, these explanations have been demonstrated for other algorithms [22]. Using the Athena provided performance monitor only the information about the CPU is provided, not the GPU, so only during initialisation, configuration and finalisation, not during execution in our case since the CPU is not used in this case. The use of Nvidia Compute was also considered, but it only measures the memory and GPU usage of a single kernel, not the overall GPU metrics during the process. In conclusion, our memory hypothesis remains unconfirmed, although it is consistent with the findings of other studies.

4.2.2.2 Tesla T4

For the second GPU, the results are more challenging to interpret. One potential factor that could have altered our conclusions is the variation in GPU utilisation depending on the time of day. This means that the GPU load could have fluctuated during our different tests, even though they were performed consecutively. However, assuming that the GPU load remained constant during the experiments, the results can be analysed accordingly.

The fact that the multi-processing configuration delivers the fastest application throughput is not surprising. This configuration assigns one processor per event, and as long as the memory is sufficient and not fully utilised, multi-processing should be at least as fast as the other configurations.

For the multi-threading configuration, the application throughput is expected to be lower. The maximum memory available to a single child process is exhausted faster than the total memory, which explains why multi-threading does not show a linear increase in throughput. This is primarily due to memory limitations and the computational constraints of each child process.

The most surprising result is that the hybrid configuration performs worse than multi-threading. In the study by R. Bielski [22], the hybrid configuration falls between the two, which is logical as it avoids the memory limitations of multi-threading but cannot match the speed of multi-processing. No clear explanation for this discrepancy is currently available, although GPU utilisation may be a contributing factor, albeit a speculative one.

In conclusion, the behaviour of the Tesla T4 is more difficult to explain and includes some unexpected results. However, it allows us to observe a different performance pattern compared to the more predictable behaviour of the RTX A5000.

Chapter 5

Summary & Outlook

This project has successfully improved the execution time of track seeding, one of the three steps of the FTF. From converting all operations to single-precision floating-point, as double precision was unnecessary, to optimising the thread configuration within the kernels, these changes result in significant improvements. The kernel time improved from 153.45 ± 4.09 ms to 115.97 ± 2.75 ms, resulting in a performance increase of almost 25%.

The second task was to study the different multi-processing configurations, and it was observed that the configuration did not seem to play a significant role, at least up to 16 concurrent events. This can be explained by the fact that, at this point, none of the memory resources were saturated. This discovery is advantageous because, with at least 16 events processed in parallel, the configuration can be chosen based on other algorithms.

Of course, this project did not cover every line of code for optimisation and focused on potential major improvements. As seen with the two main optimisations made, there is likely still room for further kernel optimisation. For example, finding a way to calculate the curvature cut, in the doublet counting and making, more efficiently could be beneficial, as this cut is time-consuming. Implementing a 2D thread configuration for the doublet matching and triplet confirmation kernels could also be an interesting way to further improve kernel performance. It would also be worth looking at memory transfers and all the processes that occur before and after the kernel runs, as the kernel only accounts for 53% of the total runtime after these optimisations.

In terms of configurations, extending the study to 32 parallel events could provide important information. Additionally, monitoring the GPU memory and computational resources during the process would offer more insight. It may also be worth investigating the effects of changing the number of threads independently of the number of slots to see whether this approach performs better or worse than keeping them equal.

In conclusion, this project has yielded promising results that can serve as a foundation for further investigation into optimising the performance and scalability of these kernels.

Chapter A

GPUs Characteristics

	Tesla T4 (Lxplus-GPU)	RTX A5000 (TestBed)	A100 SXM4 40GB (Condor)
CUDA cores	2560	8192	6912
Boost Clock Speed	1590 MHz	1695 MHz	1410 MHz
Number of transistors	13,600 million	28,300 million	54,200 million
Power consumption	70 Watt	230 Watt	400 Watt
Memory type	GDDR6	GDDR6	HBM2e
Maximum RAM	16 GB	24 GB	40 GB
Memory bandwidth	320.0 GB/s	768.0 GB/s	1,555 GB/s
Number of SMs	40	64	108
Max Threads per SM	1024	1536	1536
Total Number of Threads	40,960	98,304	165,888

Table A.1: Comparison of GPU specifications between Tesla T4 (Lxplus-GPU), RTX A5000 (TestBed), and A100 SXM4 40GB (Condor) [23–25].

Chapter B

Floating Point

Many modifications were made to eliminate double-precision calculations. The first step was to identify where double-precision variables were being initialised and modify them to use float instead. However, this was not enough, and the percentage of double-precision operations remained high, with only a slight reduction.

The second modification involved adjusting the operations themselves. When performing operations like multiplication or division between two numbers, the compiler tends to assume that the operation is in double precision. An example is the following line of code from the doublet matching kernel:

`float R2inv = 1.0f/(dx_out*dx_out+dy_out*dy_out);`.

Without the `f` suffix to specify that `1.0` is a float, the operation would have been performed in double precision, even though the denominator was declared as a float.

The final set of modifications involved replacing certain functions that default to double precision. Functions such as `cos` or `fabs` typically handle double-precision numbers. These functions were replaced with their floating-point equivalents, such as `cosf` and `fabsf`, which explicitly use float.

With these three corrections, it was possible to achieve 0% double-precision calculations.

These changes were merged into Athena on 31 July 2024. The merge can be accessed at the following link: Athena Merge Request.

Additionally, they can also be found in my GitLab repository with the 25.0.12 Athena version. You can access the repository at the following link: GitLab Repository, where the track seeding on GPU for FTF is located.

Chapter C

Threading Configuration

This type of study can be performed by adjusting the number of blocks and the number of threads per block, which are the two parameters used to configure the doublet matching and triplet confirmation kernels.
These parameters are defined in the file `SeedMakingDataStructures_ITK.h`. During the study, it was observed that the current code does not allow more than 1024 blocks to be used. Additionally, exceeding 1024 threads per block causes the code to crash, as most GPUs, including the one used in this study, have a maximum thread-per-block limit of 1024.

This is due to confusion between these two parameters in the current code. Specifically, before using `NUM_TRIPLET_BLOCKS_ITk` in the thread configuration, this value is compared to the maximum number of threads per block on the GPU using the following command:

```
if(deviceProp.maxThreadsPerBlock < p->m_gpuParams.m_nNUM_TRIPLET_BLOCKS);
```

If the number of blocks exceeds the maximum, it is set to the GPU's limit. This check occurs in the `TrigITkModuleCuda.cu` file. However, this check should be applied to `NUM_TRIPLET_THREADS_ITk` instead. The necessary changes were made to ensure that the number of threads per block is appropriately limited, while the number of blocks can vary freely.

With this issue resolved, conducting the study becomes straightforward, requiring only a modification of the parameters and rerunning the code with different configurations.

This modification has not yet been submitted for a merge request. It could also be useful to consider addressing the potential issue of having a non-integer number of blocks per SM, though this would depend on the specific GPU architecture. In many cases, if the blocks are sufficiently small, having non-integer blocks per SM has a minimal impact on computational efficiency. This adjustment could be considered in future work.

The modifications can be found at the following link: GitLab Repository.

Bibliography

[1] CERN. The accelerator complex. https://home.cern/science/accelerators/accelerator-complex, 2024. Accessed: 2024-04-29.

[2] Lyndon Evans and Lyn Evans. *The Large Hadron Collider: a marvel of technology; 2nd ed.* Physics (EPFL Press). EPFL Press, Lausanne, 2018. On the cover : Including the discovery of the higgs boson.

[3] ALICE Collaboration. First lead-ion collisions at the LHC at record energy, November 2023. Accessed: 2024-05-28.

[4] Esma Mobs. The CERN accelerator complex in 2019. Complexe des accélérateurs du CERN en 2019. 2019. General Photo.

[5] ATLAS Experiment. About. https://atlas.cern/about, 2024. Accessed: 2024-04-29.

[6] Riccardo Maria Bianchi and ATLAS Collaboration. ATLAS experiment schematic illustration. General Photo, 2022.

[7] Georges et al. Aad. The ATLAS Experiment at the CERN Large Hadron Collider: A Description of the Detector Configuration for Run 3. Technical report, CERN, Geneva, 2023.

[8] ATLAS Collaboration. ATLAS Experiment at CERN. https://atlas.cern/, 2024. Accessed: 2024-08-20.

[9] G. et al. Aad. The atlas trigger system for lhc run 3 and trigger performance in 2022. *Journal of Instrumentation*, 19(06):P06029, June 2024.

[10] High-Luminosity LHC Project. The high-luminosity lhc project, 2024. Accessed: 2024-05-09.

[11] ATLAS Software and Computing HL-LHC Roadmap. Technical report, CERN, Geneva, 2022.

[12] Claire Antel. Upgrade and Physics Plans for ATLAS in LHC Run 4. 2021.

[13] ATLAS Collaboration. Expected tracking and related performance with the updated ATLAS Inner Tracker layout at the High-Luminosity LHC. Technical report, CERN, Geneva, 2021. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-PHYS-PUB-2021-024.

[14] ATLAS Tracking CP Group. TRT Seeded Backtracking - Tracking Tutorial. https://atlassoftwaredocs.web.cern.ch/internal-links/tracking-tutorial/trtseeded/, 2024. [Accessed: September 8, 2024].

[15] Andrius Vaitkus and ATLAS Collaboration. Fast track seed selection for track following in the Inner Detector Trigger track reconstruction: Proceedings. Technical report, CERN, Geneva, 2023.

[16] Cornell Virtual Workshop. Gpu characteristics - design, 2024. Accessed: 2024-09-05.

[17] ATLAS Collaboration. Athena, May 2021.

[18] NVIDIA Corporation. Nvidia nsight systems, 2024. Accessed: 2024-09-26.

[19] NVIDIA Corporation. Nvidia nsight compute, 2024. Accessed: 2024-09-26.

[20] QNX Software Systems Ltd. atomic_add() function. https://www.qnx.com/developers/docs/8.0/com.qnx.doc.neutrino.lib_ref/topic/a/atomic_add.html, 2024. Accessed: 2024-10-03.

[21] ATLAS Collaboration. ATLAS DAQ Approved Plots. https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsDAQ, 2024. Accessed: 2024-09-11.

[22] Rafal Bielski. Trigger software performance scaling. Technical report, CERN, Geneva, 2023.

[23] TechPowerUp. Nvidia tesla t4 specifications. https://www.techpowerup.com/gpu-specs/tesla-t4.c3316, 2023. Accessed: 2024-09-12.

[24] TechPowerUp. Nvidia rtx a5000 specifications. https://www.techpowerup.com/gpu-specs/rtx-a5000.c3748, 2023. Accessed: 2024-09-12.

[25] TechPowerUp. Nvidia a100 sxm4 40 gb specifications. https://www.techpowerup.com/gpu-specs/a100-sxm4-40-gb.c3506, 2023. Accessed: 2024-09-12.