Technische Universität München

Fakultät für Informatik

Informatik 5 – Fachgebiet Hardware-nahe Algorithmik und Software für Höchstleistungsrechnen

Analysis and Optimization of the Offline Software of the ATLAS Experiment at CERN

Robert Johannes Langenberg

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Bernd Brügge, PhD
Prüfer der Dissertation:	1.	Prof. Dr. Michael Georg Bader
	2.	Prof. Dr. Siegfried Bethke

Die Dissertation wurde am 03.07.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.10.2017 angenommen.

ABSTRACT

The simulation and reconstruction of high energy physics experiments at the Large Hadron Collider (LHC) use huge customized software suites that were developed during the last twenty years. The reconstruction software of the ATLAS experiment, in particular, is challenged by the increasing amount and complexity of the data that is processed because it uses combinatorial algorithms. Extrapolations at the end of Run-1 of the LHC indicated the need of a factor 3 improvement in event processing rates in order to stay within the available resources for the workloads expected during Run-2.

This thesis develops a systematic approach to analyze and optimize the ATLAS reconstruction software for modern hardware architectures. First, this thesis analyzes limitations of the software and how to improve it by using intrusive and non-intrusive techniques. This includes an analysis of the data flow graph and detailed performance measurements using performance counters and control flow and function level profilers. Different approaches and possible improvements are evaluated. An analysis of a new low complexity track reconstruction algorithm is presented.

No full analysis of the grown structure of the ATLAS reconstruction software had existed so far. The performance analysis identified where optimization efforts would have the largest impact. The analysis concludes that the tracking, already being the computationally most expensive part, would become more dominant with future data. Consequently, the optimization efforts concentrated on the tracking, combining algorithmic and structural changes as well as replacing outdated libraries. A detailed description of the implemented changes and their impact is presented. All changes combined lead to a speedup of a factor of 4.5 for the reconstruction. Both code quality and the process to ensure code quality have been improved. For the presented parallelization study a data flow graph was created and combined with module's runtimes. The critical path was found to consume 95% of the reconstruction time, showing that there is little potential performance improvement through of inter-module-parallelism. The analysis concludes that for the tracking most parallelism must be inside a majority of the modules to have a significant effect. Currently, efforts are focused on achieving thread-safety of all modules to start introducing parallelism, while recent efforts reduced the overhead of processing multiple events concurrently on multicore machines. The new transform-based tracking method presented is parallelizable within an event and can be applied to preselect tracks to reduce the complexity for the classic slow reconstruction. It runs up to 17.5 times faster than the comparable algorithm currently in use.

CONTENTS

List of Tables

List of Figures

List of Appendices

1 INTRODUCTION

High-energy particle physics experiments are conducted to study the properties of fundamental particles generated in collisions at particle colliders. The term “high-energy” indicates the very high energies prevailing at the collision process of initial particles that imitate conditions moments after the creation of the universe. During a collision, this energy is available to be converted into other particles with potentially much higher mass. This process is described by the theory of particle interaction. It describes how each physical process occurs with a certain probability. Physicists are interested in observing rare processes, e.g. the production of Higgs bosons and their decay in proton-proton collisions at the LHC. Some processes have a very low probability such that billions of collisions have to be analyzed in order to find a single occurrence. To accumulate sufficient statistics, a large number of signatures consistent with the desired process has to be collected, such that petabytes of data have to be analyzed.

The Large Hadron Collider (LHC) [1] is the largest machine constructed by mankind producing rare particles at higher energies and at a higher rate than ever before. It is designed to accelerate protons up to energies that correspond to more than 14,000 times their invariant mass and to generate collision events. Collision events, during each of which multiple protons can collide, have a minimum temporal distance of only 25ns between two events. This suggests 40 million collision events per second, but because of gaps the number is closer to 30million per second. During each of these collision events, the LHC design foresaw on average 23 simultaneous proton-proton collisions. This number has already even been surpassed. Although having only one collision per event would simplify the subsequent analysis, the increased complexity is accepted in order to increase the rate of producing rare physics processes. Storing and processing all events would correspond to storing more than one hundred terabytes of data per second, which is not feasible. This is why a so-called trigger system selects only certain events for detailed analysis that follow distinct signatures, such as having specific particles.

The ATLAS experiment [2] is one of the four main experiments at the LHC. The detector consists of several sub-detectors, designed to measure different particle properties. Closest to the center is the Inner Detector (ID), which consists of three different sub-detectors. Surrounding the ID is a strong solenoid magnet. The ID is designed to measure the origins and trajectories of charged particles. The two detectors closest to the center work with sensors similar to those in a digital camera. They allow to precisely measure the locations where charged particles pass through and other information. This “camera” has more than 90 million readout channels and takes 40 million pictures per second, up to

1000 of which are selected by the ATLAS trigger for analysis. The outermost detector of the ID works with a different technique, which has a lower spatial resolution but takes a high number of measurements per particle. Outside of the magnet are the calorimeters, which are designed to stop most particles to measure their energy. The outermost detector system is the Muon Spectrometer, which was designed to measure muons, a type of particle not stopped by the calorimeters. The Muon Spectrometer can measure the trajectory of charged particles (muons) independently of the ID and the calorimeters because it has a separate magnet system, the ATLAS Toroid Magnet System. The ATLAS detector is hermetically surrounding the collision area to measure as many of the produced particles as possible.

In order to make accurate measurements, the detector has to be aligned to the precision of a few micrometers and calibrated accordingly. To analyze the measured particles, the events are then reconstructed to interpret the measurements, such that it is possible to determine which physics processes occurred. The simultaneous collisions of multiple protons cause tens of thousands of position measurements in the ID, such that complex pattern recognition algorithms are required to reconstruct particle tracks. This process is called track reconstruction or tracking. During event reconstruction, many other physics objects are reconstructed from the recorded information in the ID and other detectors such as charged particle trajectories, the particle production vertex or energy clusters. Tracking is also performed in the muon spectrometers, while the calorimeters are used to measure other particle properties by analyzing their decay.

The track reconstruction in the ID is the most complex and computationally most expensive of all reconstruction problems, the optimization of which is the main focus of this thesis. Reconstruction requires solving complex combinatorial problems to find the most probable set of measurements that stem from one particle. To find probable tracks, possible combinations of location measurements created by traversing particles have to be evaluated. Therefore, the complexity of processing the ID data is directly dependent on the number of particles per event.

For the event reconstruction ATLAS utilizes a software framework, Gaudi-Athena [3]. It provides the backbone of a huge custom software suite. To confirm the feasibility of the experiment in simulations, parts of this software suite were written more than a decade ago, even before the detector construction was approved. Software design choices made at that time still influence development today. Over the years, the software has grown to 6 million lines of code in thousands of packages [4].

The evolving physics requirements made continuous modification and optimization of the software necessary. An example for a huge change of requirements emerged with the upgrade period between 2012 and 2015 with changes to the detectors and to the LHC. This period is known as the Long Shutdown 1 or LS1. The changes to the detector and the LHC during this time foresaw a doubled frequency of events and an average of 40 simultaneous collisions per event. The trigger updates during LS1 are designed to increase the number of events selected by the trigger by a factor of 2.5 to up to 1kHz. Including other factors this is an enormous increase of data to be processed while at the same time the complexity of the reconstruction of each event increases significantly.

Future major upgrades planned for 2023, the so-called High-Luminosity-LHC, is foreseen to result in up to 200 simultaneous proton-proton collisions per event and to record events at rates of 5kHz or more after trigger selection. The scale of the problem is visualized by Figure 1, showing a reconstructed High-Luminosity LHC event and the corresponding measurements. The LS1 was a window of opportunity to address the increased demands for 2015 by improving the software to meet the new requirements. Much effort has been put in optimizing the code and algorithms, which is part of this thesis.

The ATLAS software is run on a set of computing sites distributed around the world, which jointly form the largest scientific grid-computing network in the world, the Worldwide LHC Computing Grid (WLCG) [5]. These sites belong to universities and collaborating laboratories, which also manage them. This means these sites have diverse hardware

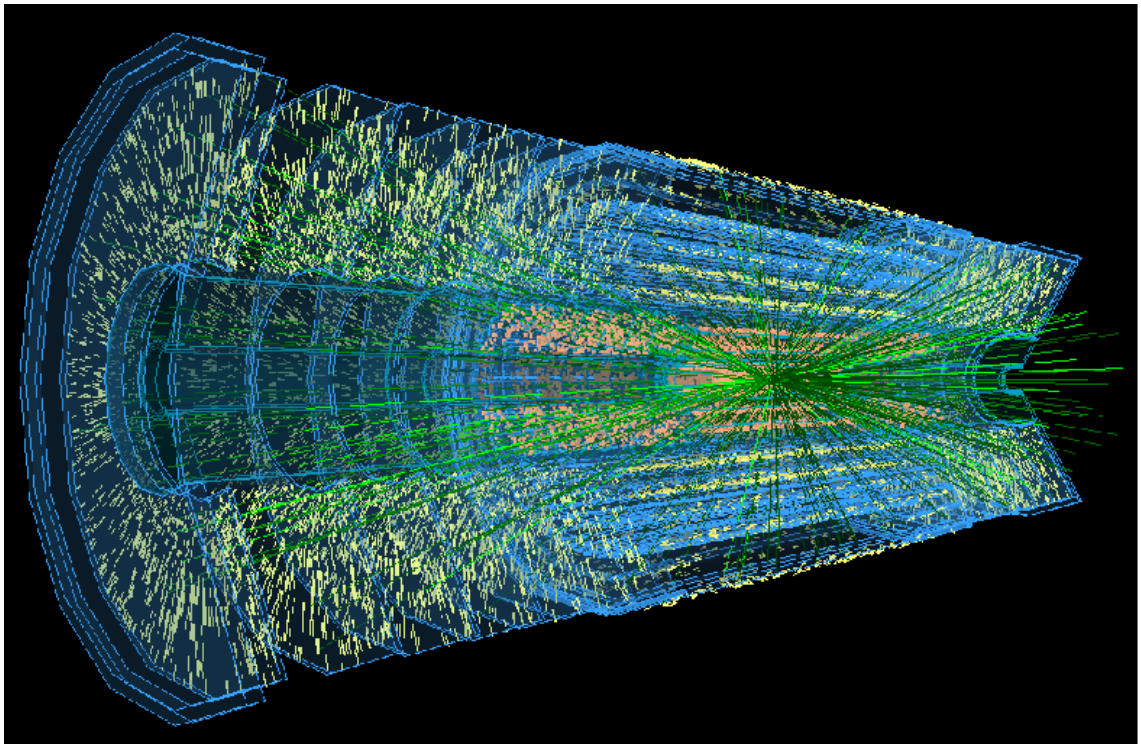


Figure 1: Event with 140 simultaneous collisions in an upgraded ATLAS detector geometry. The detector elements are outlined in blue and the location measurements are in orange and yellow. The reconstructed particle tracks are shown in green. [6]

that can change anytime. Yet, large amounts of data have to be processed as efficiently as possible, requiring attention to performance of all parts of the software, both in terms of physics and computational performance. To achieve the required computational performance, code maintenance needs to address new hardware architectures, changing software environments and evolving physics requirements. Such conditions pose a problem particularly for long-lived projects that grew historically, because optimizations exploiting new hardware features often require profound changes in the software. High-energy physics software is different to most other projects that require computing power of this scale. These experiments have different sub-detectors with different purposes, such that many different problems have to be solved, distributing the time spent over millions of lines of code with few hot spots. Writing such software requires a profound understanding of the involved physics processes, which is why it is typically written almost entirely by physicists rather than software engineers.

Significantly optimizing the ATLAS ID reconstruction requires a thorough analysis of the framework, the algorithm interplay, the algorithms themselves and their performance on the processing hardware. For all considered optimizations has to be kept in mind that they impact both performance and maintainability. Maintainability is especially important in the ATLAS collaboration as the skills of the developers differ significantly and knowledge of the software gets lost due to the quick turnover of the developers which are often students.

For decades, the need for faster software was addressable by increasing the computing power by buying newer machines, whose clock speed was increasing following Moore's Law. These machines were able to run the same code much faster than the previous generation, sometimes even without requiring recompilation [7]. In 2006 clock speed development stalled at around 3 GHz. The reason is the superlinear increase in power leakage with higher clock speeds, an issue commonly referred to as the power wall. The

number of transistors per processor area continues to grow following Moore's law [8]. With sequential execution of most basic operations already at the minimum of one instruction per clock cycle through pipelining, however, individual operations do not profit from the increase in transistors. To utilize the additional transistors, CPU vendors started to introduce more parallelism, manifested in wide CPU registers, parallel ports, hardware threads and, since 2001, more cores. The difference to improvements in basic operations or in clock speed is that software does not indistinctly profit from these developments. Instead, it depends on the software to make use of the new features or to allow automated usage of these features. Leaving these resources lying idle, however would lead to wasting potential of several orders of magnitude. Exploiting largely independent parallel resources such as multi core CPUs by running multiple instances of the current ATLAS software will not remain feasible for future architectures: Memory per core in future architectures will not be as abundant as on current systems [9]. Maintaining the current ratio for the increasing number of cores would lead to unacceptable acquisition and electricity costs. As a consequence, to be able to utilize these resources, either memory usage per instance must decrease substantially or parallelizations without huge additional memory cost must be applied.

1.1 Research Questions

The ATLAS software has grown to millions of lines of code over the years of its development. The code runs on hundreds of thousands of computers worldwide causing millions of Swiss Francs of cost. Cost for computation will exceed the available resources without improvement of the software, because the problem solved by the software predictably shifts towards higher complexity and size with the performed and planned updates to the experiment. This makes it a classical optimization problem. Due to its sheer size and the untypical distribution of load over many parts of the code, focusing on a single point cannot yield the required gains. Another constraint is that the quality of the results may improve, but not reduce with any changes to the software. These constraints show that on the one hand, it is not sufficient to optimize few hot spots, while on the other caution is necessary when applying optimizations that may affect the results. Only understanding the software enables developers to tackle these challenges and meeting the goals.

When modifying the software design, the suitability for current and future hardware should be taken into account while at the same time future efforts should come at a lower cost. Large-scale changes to the software need to be organized in a way they can be performed by the individual groups without expert knowledge. All efforts to meet the short-term requirements had a strict deadline, which was set by the restart of the LHC after LS1. For long-term projects, the applicability of certain technologies such as different parallelization techniques to the problems to be solved for reconstruction would have to be analyzed, requiring both an analysis of the technologies as of the current structure of the code. I identified three individual steps required to achieve these goals, which are addressed by the contributions outlined in 1.2. These steps are:

1. **What are the main shortfalls of the current software with respect to CPU performance?**
   To be able to improve a piece of software, it has to be analysed for both wasted and unused resources. Understanding reasons for how the software works requires significant effort but is necessary to expose its limitations and to restructure and reformulate problems. At the same time, algorithmic changes may also affect physics performance, which must not be reduced even by a fraction of one percent. Analyzing the interplay, on the one hand, requires code analysis and expert knowledge distributed over many different people. On the other hand, it requires

benchmarking software behavior and the evaluation and interpretation of such benchmarks.

2. **Which improvements can be implemented before the end of LS1?**
   First and foremost, attention is to the performance of the software in the near future. Pressing issues are different workloads, which shift hot spots and can create problems that did not appear with previous workloads. Expected hardware developments, the computing budget in the near future and predicted workloads were used to calculate that the reconstruction software was required to be sped up by a factor of three to deal with the new workloads [10]. Therefore, estimating gains of a project and the time its execution takes using the previous analysis is necessary for timely completion of the projects as well as achieving the set goals.
3. **Which long-term improvements are most promising?**
   The ATLAS experiment is going to run until the end of the LHC lifetime 2030, and while requirements will change, the type of changes and their impact can be foreseen. The software has to deal with increasingly complex events with a constant computing budget and a changing and increasingly parallel hardware landscape. In order to stay within the computing budget, utilization of parallel resources on all levels is necessary. Pinpointing possible promising projects, analysing required effort and possible impact could serve as a starting point for further developments that go outside of the scope of this thesis.

1.2 Contributions

The following approaches and new methods are presented in this thesis, which are designed to improve the ATLAS software such that it can fulfil the requirements of the near term and long term future:

1. A general overview of the state of the ATLAS reconstruction software includes a detailed performance analysis [11], [12]. It demonstrates the problem of developing the complex software of ATLAS from an organizational point of view. I highlight the challenges arising with organizing hundreds of people, including many students with differing skill levels, working on different areas. This includes as well a discussion of the parallelizability of the ATLAS software. The bottlenecks within the current approach preventing the introduction of parallelism on different levels are analysed in detail. Concrete measures are presented to counter shortfalls in near future scenarios.
2. The measures in which I participated that were implemented during LS1 to be able to deal with the workloads of the next phase of the LHC data taking campaign are described in detail. Non-intrusive as well as intrusive optimizations are applied, following the analysis results. I discuss immediate and future impact of each optimization with respect to performance and maintainability.
3. I analyze and improve a near-linear complexity vertexing and tracking algorithm, which presents an outlook to how the software can be prepared for future challenges. This algorithm tries to avoid the most expensive operation of the currently used ID tracking algorithms. Based on a trigger algorithm, I heavily modified it to achieve the best possible physics and CPU performance. Due to its much lower computational complexity and because it is largely based on bit operations, it is orders of magnitude faster than the currently used tracking algorithm. The modifications make parallelizations possible, which, depending on the configuration, would allow to divide the problem into hundreds of independent sub problems. The impact of the detector geometry's differences to the simplified assumptions used in the algorithm are analyzed in detail and corrective measures are imple-

mented. A fully working prototype has been implemented and serves as an example to show under which conditions the algorithm is applicable.

1.3 Structure of the Thesis

The remainder of the thesis is structured as follows:

Chapter 2 – Research Environment: The environment in which the work presented in this thesis is embedded is given in this chapter. It starts with an introduction to CERN and the purpose and technical challenges of the Large Hadron Collider and ATLAS together with other LHC experiments. A separate subsection is dedicated to the Inner Detector. The ID is the innermost part of the ATLAS experiment and a main focus of this thesis. Continuing with the computing environment and the available resources, the remainder of this chapter explains the general functionality of the main software and algorithms for the ID reconstruction.

Chapter 3 – Software Development and Computing in ATLAS: The chapter motivates the challenges and the problem description in detail. This comprises the complexity of previous and expected data and resource consumption of the software before the start of the thesis. A detailed analysis of the interaction of the different modules demonstrates the complexity and number of interacting modules. The challenges of an organization with contributions from many people with different backgrounds and in separate working units developing these modules are indicated as well. A section is dedicated to the hardware evolution and the different available hardware types on the market and available to ATLAS. The requirement to parallelize the single threaded software to exploit future hardware resources is explained. Lastly, the development of programming languages, external libraries and other surrounding software since the start of development on Athena is discussed with respect to their importance for ATLAS.

Chapter 4 – Performance Analysis: A detailed analysis shows the state of the ATLAS reconstruction before and during the implementation of different optimizations. The analysis breaks up the contributors to CPU usage in domains and modules and shows their dependencies to give an insight into the complexity in order to find hot spots for optimization. Subsequently, the different types of possible optimizations are explained. They are grouped into three conceptually different domains: Software environment, hardware environment and own software. Applicable optimizations from the own software and software environment domain are analyzed. Examining the dependency chain of the ID software clearly shows that running algorithms in parallel cannot lead to huge gains.

Chapter 5 – Software Integration: Using information from the performance analysis, changes were introduced into the software. The software integration chapter presents the implementation of these changes in the software domains until the end of the long shutdown. The expected gains for different ways to implement a change are assessed, the process of implementation described and a prediction of the expected impact given.

Chapter 6 – Algorithmic Improvements of Tracking: The currently used tracking methods have a very high complexity, use slow operations and cannot be parallelized trivially. In this chapter, alternative algorithms are presented that try to tackle these problems. The detailed description of applicable algorithmic improvements is followed by a feasibility study to assess the physics performance and the possible areas of application. Further improvements to the algorithm are discussed and physics performance results of

an implementation presented. The parallelizability and throughput of different parallelization methods of different tracking approaches is discussed.

Chapter 7 – Conclusions: A summary of the achieved goals of analyzing and optimizing the ATLAS software. The impact of the major findings are presented alongside an outlook to future projects.

2 RESEARCH ENVIRONMENT

This chapter is dedicated to describing the goals and the purposes of the organizations and projects in which the research presented in this thesis was embedded. It provides the technical context describing software and resources available. It should equip the reader with the necessary background for the subsequent chapters.

This chapter is structured as follows: CERN is introduced in Section 2.1 to explain the overall goals of this organization. CERN famously operates the LHC, which will be introduced in 2.2. The amount of data generated by the LHC and how much of the data was recorded and processed by the ATLAS experiments is shown in Section 2.3. One of the four main experiments at the LHC and the organization in which this thesis is embedded is the ATLAS experiment which will be described alongside its detector components in Section 2.4. Section 2.5 presents ATLAS' inner tracking subsystem, the Inner Detector. Algorithms to process data from the ID are very time intensive, which is why most of the work presented in this thesis is primarily aimed at the ID. The type of data taken at ATLAS and how it is processed is presented in Section 2.6. Section 2.7 explains an especially costly processing step of data, the track reconstruction in the ID and its algorithmic solution, which this thesis focuses on. Section 2.8 explains the main software suites used at ATLAS. Section 2.9 goes into details of the ATLAS software framework Athena running all software developed in the context of this thesis. The computing infrastructure is introduced in Section 2.10 and gives details of the distributed computing resources available to ATLAS.

2.1 CERN

CERN was founded as the "Conseil Européen pour la Recherche Nucléaire" on 29 September 1954 for strictly peaceful fundamental research in physics by twelve western European countries [13]. It was created to re-establish world-class research in Europe, which had suffered during World War 2. Since then many nations have joined CERN to collaborate even during political tensions, and CERN has become one of the largest institutions for basic and applied research worldwide. Around 10,000 people work for CERN or CERN experiments on site, and thousands more in institutes worldwide. These people come from over 70 countries and 120 different nationalities [14], making CERN a gathering point for different cultures. It is based on the border of France and Switzerland close to Geneva and features facilities in both countries. The site was chosen for political stability

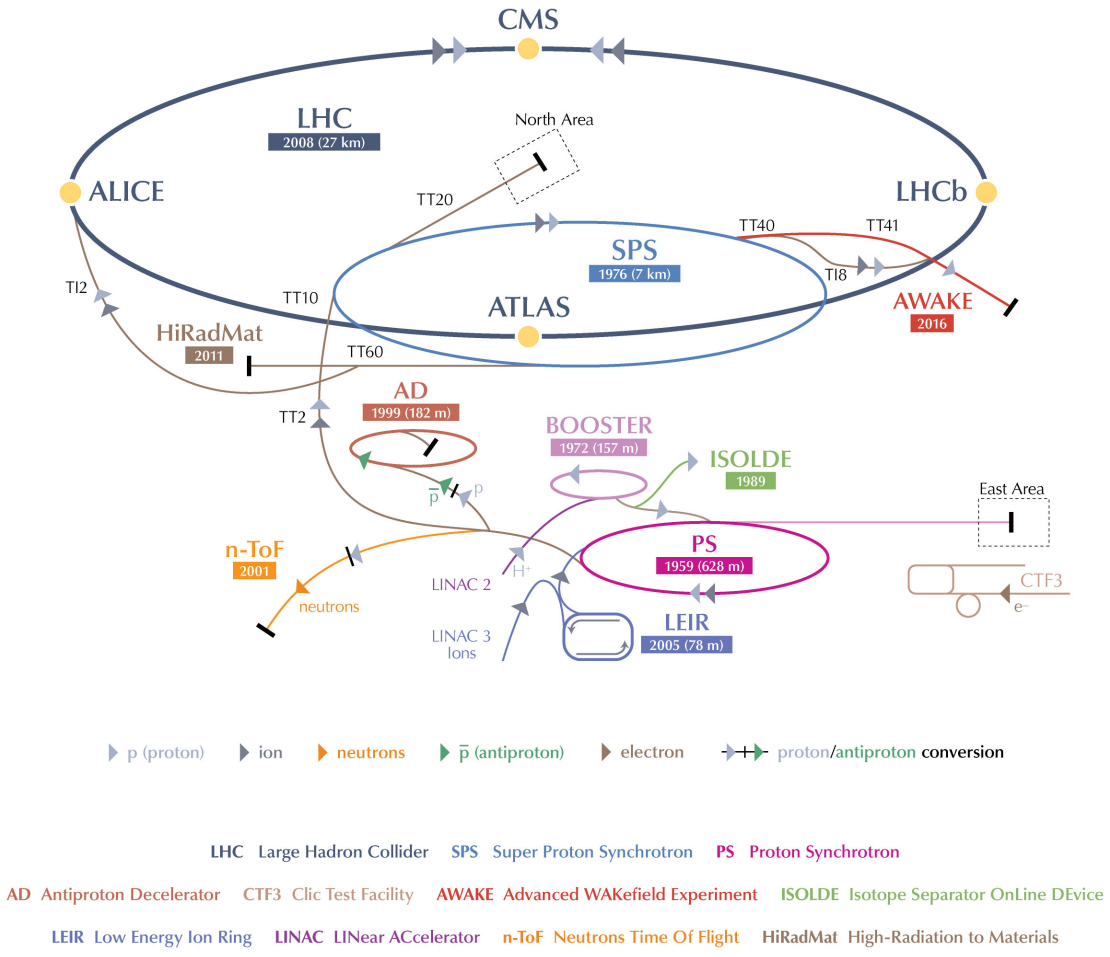


Figure 2: The accelerator complex at CERN. The acceleration of protons starts in the Linear Accelerator 2 and reaches the LHC with intermediate steps in circular accelerators of increasing size: Booster, PS and SPS. The four main experiments are marked on the LHC ring. Other accelerators are for different particles and/or different experiments. Graphic from [98].

and its central location within Europe. Many discoveries and major technological innovations have been made, both in the field of physics and outside of physics. Famously, the World Wide Web (www) has been invented here as a means to share information between researchers. Several medical applications such as cancer treatments are based on technologies developed at CERN. Since the very beginning CERN has been at the forefront of research in the area of computing.

CERN hosts the Large Hadron Collider (LHC), currently the world's largest particle accelerator that provides data to four main experiments ATLAS, CMS, LHCb and Alice [15], [16], [17]. In 2012, ATLAS and CMS announced the discovery of the Higgs boson [18], [19], a fundamental particle these two experiments were able to measure with unprecedented accuracy.

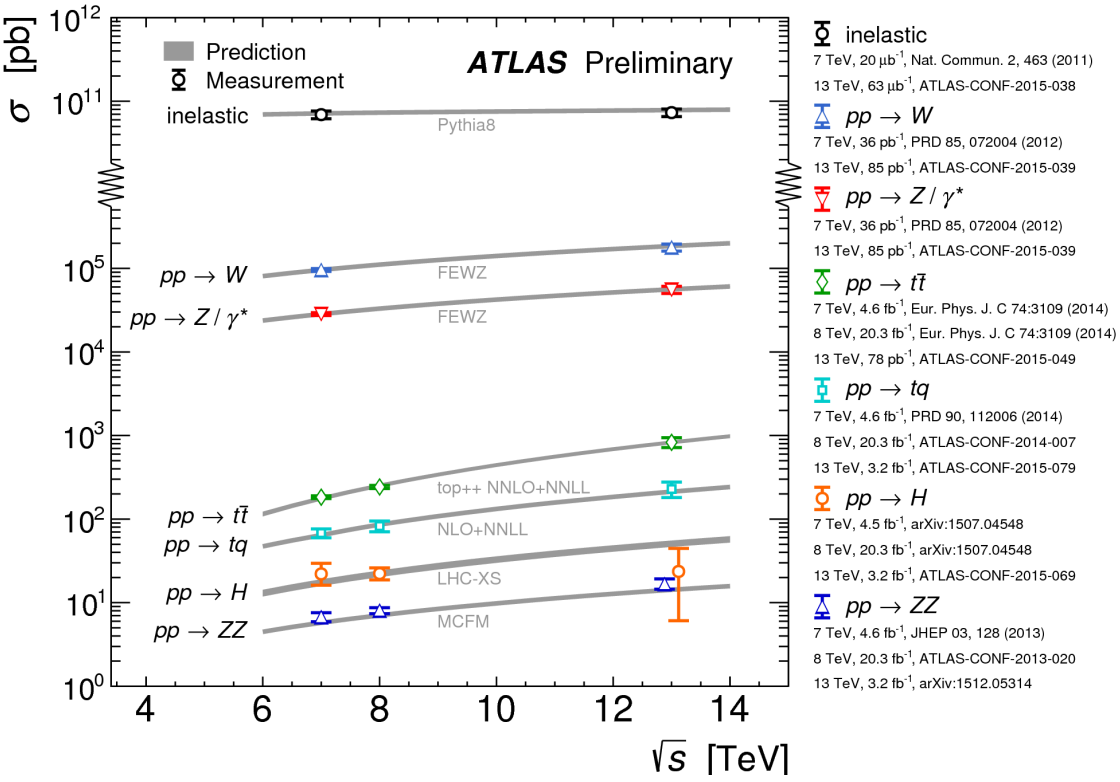


Figure 3: Cross section of certain physics processes for LHC center-of-mass energies before and after LS1. The cross section of a process corresponds to the probability of this process to occur. The process marked "inelastic" corresponds to the probability of a proton-proton collision. As the graphic shows this probability is almost 6 orders of magnitude higher than the next most probable shown process, and about 10 orders of magnitude more probable than the least probable ZZ process. This means that on average ten billion collisions have to take place for one ZZ process to occur, and to create reliable statistics many are needed. Graphic from [99].

2.2 The Large Hadron Collider

The LHC is the biggest particle accelerator in the world, designed to reach collision energies of 14 TeV (Tera electron Volt) for proton-proton collisions and up to 40 million collision events per second [20]. The LHC has been commissioned in 2008, replacing the previous accelerator in the same tunnel, the Large Electron Positron collider (LEP), which had been in operation between 1989 and 2000. Between 2008 and 2012 the LHC has been operating with energies of up to 8 TeV during the first data-taking period referred to as Run 1. After the first long shutdown (LS1) between 2012 and 2015, collisions with a center-of-mass-energy of 13 TeV marked the start of Run 2. To accelerate protons to such energies, they pass through a chain of accelerators, see Figure 2, where each accelerator increases the energy to a higher level than the previous one. Within the LHC, protons are accelerated in two counter-rotating particle beams along the 27km circumference ring-shaped collider situated in Switzerland and France. The two beams are made to cross in four points at the location of each of the LHC's four main experiments. When the beams cross, particles from one beam interact with particles from the other beam. This is called a collision event.

In each of the collision events, two so called bunches each consisting of around 10^{11} protons travelling in opposite directions pass through each other. During one such bunch

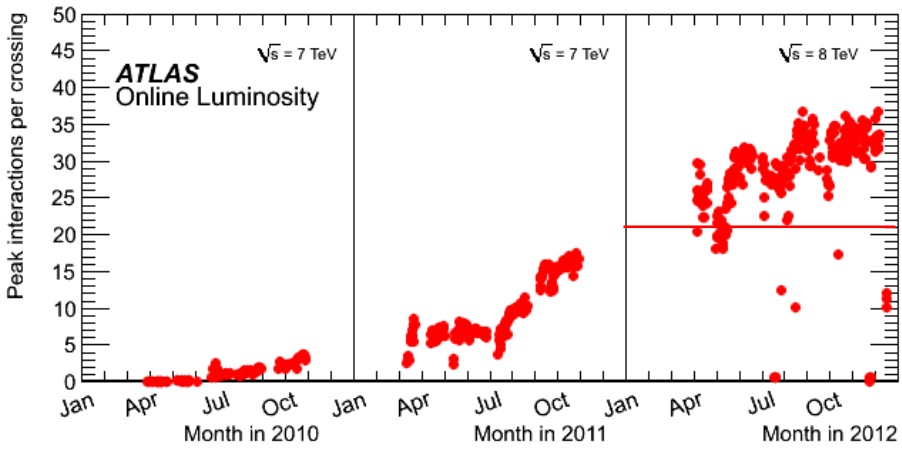


Figure 4: Peak number of proton-proton collisions per event. Maximum reached in 2012 is 37. The red line indicates the average of 20.7 pileup collisions in 2012. [21]

crossing, a number of protons from one bunch collide with protons from the other bunch.

The large number of collisions is necessary because the desired physics to be observed occurs only with a very low probability, see Figure 3. The so–called scattering formalism can be used to describe such processes. To understand the formalism, it is necessary to know that in a proton-proton collision, not entire protons but the constituents of the protons, the so called partons, collide. The parton distribution functions describe the probability to find partons inside the protons of a particular energy during a collision. The probability for a given (scattering) process to occur is then computed using Feynman diagrams. The convolution of the parton distribution functions and the Feynman diagrams returns the probability of a process to occur. High energies are interesting from a physics point of view because they are required to produce certain heavy, possibly unknown particles predicted by theories extending the Standard Model. The recent discovery of the Higgs boson serves as an example. While the probability for two partons with sufficient energy to produce a Higgs is small, the probability is much smaller to find these two needed partons with a sufficient energy.

Due to the extremely short lifetime of the Higgs boson, only its decay products can be measured. Of the many possible decays of a Higgs boson, only a few are actively looked for because they occur with a sufficient probability and can be detected. As a result, although trillions of events have been produced, only a few hundred events showing the signature of a measured Higgs particle decay have been found.

2.3 LHC Performance in Run 1

At the end of Run 1 in the beginning of 2013, the LHC was shut down after more than 3 years of data taking. The total integrated luminosity delivered to the ATLAS experiment during Run 1 was approximately 27 inverse femtobarn, which corresponds to about $2.7 \cdot 10^{15}$ proton-proton interactions, which took place in trillions of events. In 2012, the last year of Run 1, about 80.5% of the data of Run 1 was taken [22]. With the average pileup of 20.7 inelastic collisions per event for 2012 this corresponds to around 10^{14} events [23], [24]. The trigger selected around 2 billion of these events in 2012, which were fully reconstructed, all with software version 17.2. The software fulfilled the requirements of the computing infrastructure, reconstructing events reaching up to 37 pileup interactions in 2012, see Figure 4.

The excellent performance of the LHC comes at a price because the additional simultaneous collisions need to be reconstructed, which strongly affects the runtime of reconstruction, as shown in Section 3.1. The reconstruction of each event is broken down into

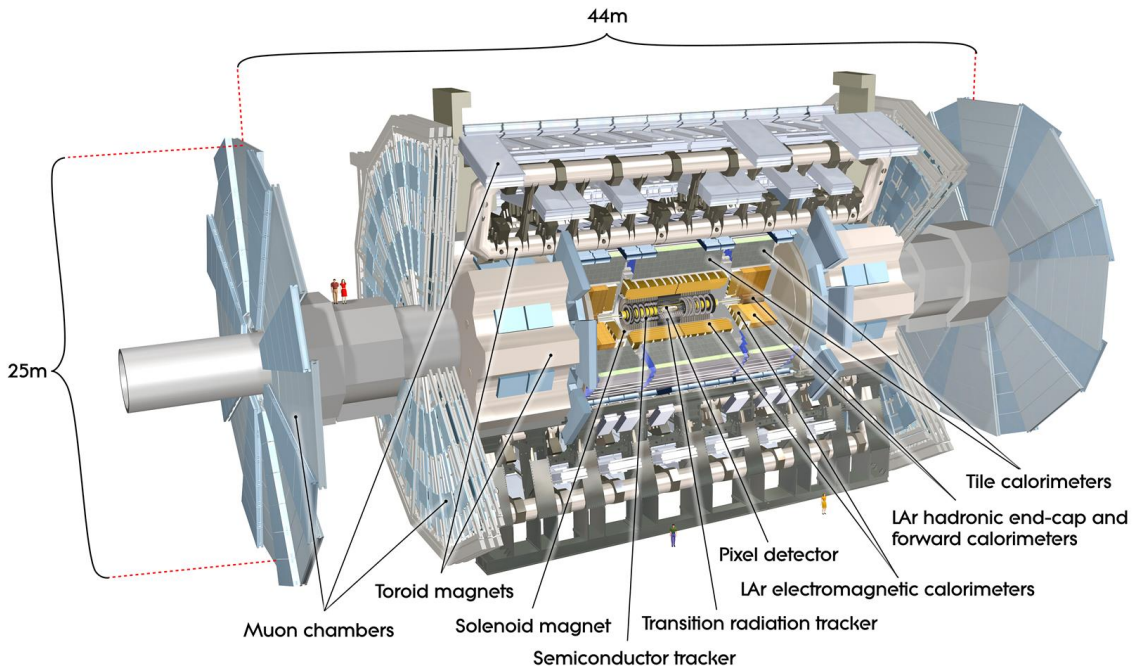


Figure 5: The ATLAS detector and its subsystems [2].

reconstruction within the different subdetector systems of the ATLAS detector, which are presented in the following Section.

2.4 ATLAS

By analysing particle collisions, physicists want to experimentally confirm existing theories and hope to discover previously unpredicted behaviour, so called new physics. The two beams are made to cross each other at four points where data is taken and recorded for analysis by four experiments: ATLAS, CMS, ALICE and LHCb. Each of these experiments has a detector at the collision area, which is capable of detecting produced particles. ATLAS and CMS are general-purpose detectors designed to detect a range of particles and covering the beam interaction region hermetically. ALICE is designed to work best for the analysis of heavy ion collisions, which are produced by the LHC during dedicated runs over a period of four to six weeks, usually at the end of each year. LHCb is designed to detect the decay products of beauty hadrons to detect deviations from the Standard Model. Though not perfectly suited for all purposes, as of 2015 all experiments record data from both heavy ion and proton-proton collisions.

The ATLAS detector has the largest dimensions of the four major experiments at CERN and consists of many different systems, see Figure 5. The detector encloses the collision area in its center. Along the beams, the detector systems are cylindrical, enclosing each other. At both ends of the cylinders, disc-shaped endcaps of each detector system close the detector volume. This structure is designed to measure as many particles as possible and the different detector subsystems are used to measure different aspects of these particles. Data is measured by different detector subsystems. The innermost detector is the ID, which is a tracking detector designed to measure the origins and trajectories of charged particles close to the beam interaction region. Within the ID, the particles pass through a strong magnetic field, causing a charged particle's path to bend, depending on its charge and momentum. Under the assumption of a homogeneous field and no material interactions, the particles follow the path of a helix. From the curvature, the charge and momentum of the particle can be derived. Calorimeters are wrapped around the solenoid magnet

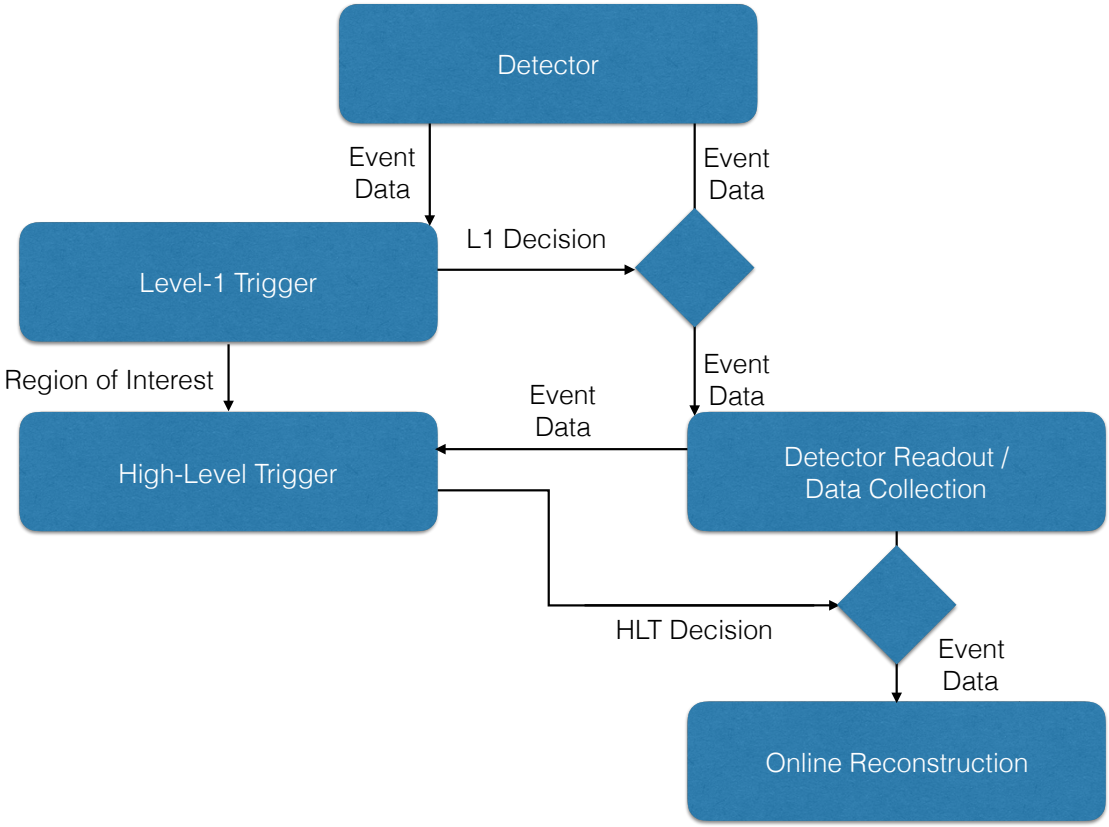


Figure 6: ATLAS Trigger system schematic. The Level-1 (L1) Trigger inside the detector signals the readout system if data should be read out. The High-Level Trigger (HLT) is given a Region of Interest by the L1 Trigger.

enclosing the ID to measure the energies of most charged and neutral particles. Different types of calorimeters are designed to stop different types of particles. They aim to contain the majority of particles within this volume by absorbing all their energy and measuring it in the process. The calorimeters measure a particle's energy for charged and most neutral particles. The outermost detector sub-system is the Muon Spectrometer, which is a tracking device dedicated for the measurements of muons that traverse the calorimeter volume due to their relatively low interaction with the calorimeter material.

ATLAS needs to record interesting events from the overwhelming stream of data. The particles resulting from the collisions pass through various detector surfaces and induce ionization that are detected by the read-out electronics. On board electronics are programmed to react to coincidence certain signatures that indicate a particle of a certain type or energy. This system is called the hardware or Level-1 trigger [25]. It is part of a multi-stage real time triggering system designed to identify events of interest, see Figure 6. If certain conditions are met, this causes the event to be read out from the detector and analysed by the next level trigger system. This High-Level trigger or HLT, before Run 2 divided into Level 2 and Level 3, is implemented in software. It does not run on the full event but just on parts of the data where interesting physics is expected, so-called regions of interest. The processing of these regions is done with algorithms based on algorithms of the offline reconstruction, which denotes the full reconstruction happening after the selection by the trigger system, which does not have real-time constraints. Most work in this thesis is about offline software algorithms, but changes to these algorithms can also be applied to the HLT.

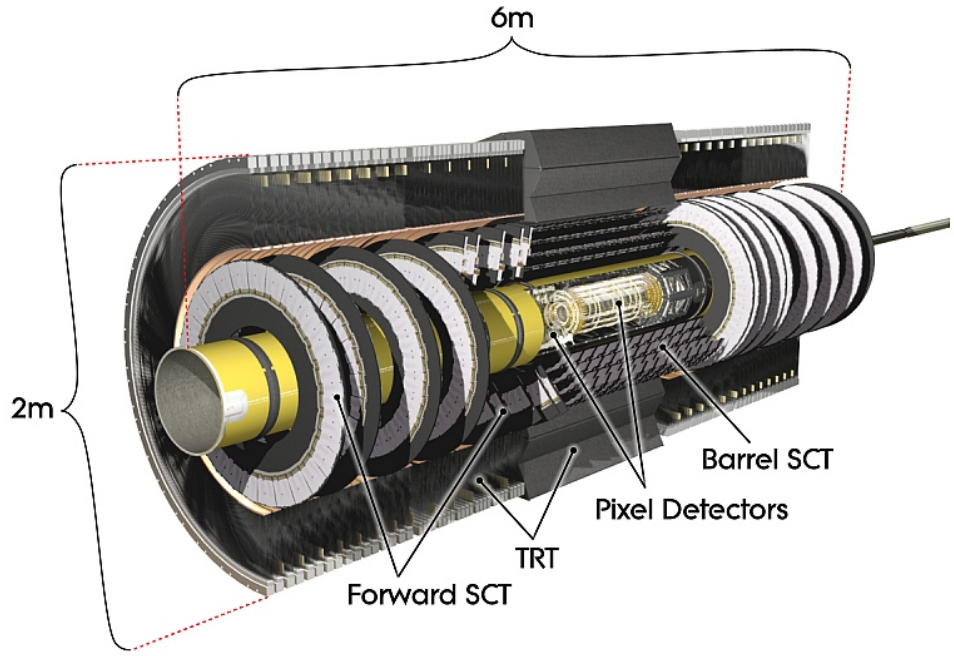


Figure 7: ATLAS Inner Detector cross-section showing the different detector sub systems [2].

2.5 The ATLAS Inner Detector

The Inner Detector [2] is in the focus of many of the algorithms and measures described this thesis. Its detailed description is important to understand the inputs for the algorithms and the effects they have to model. This subsection describes the ID comprising all detector elements in the center of the detector. The ID has a high sensitivity, resolution and number of readout elements for high precision measurements of charged particles close to the beam interaction region. It is located inside a strong solenoid magnet and contains silicon and straw tube detectors. These detectors are sensitive to charged particles traversing them, measuring the deposit of a small fraction of the particle's energy via ionisation in the active detector material. Its elements are constructed in cylindrical layers called barrels around the collision area with a radius as small as 3.325cm as of Run 2 and 80cm length up to 106.8cm radius and 272cm length, for the accurate determination of a particle's path. The cylinder ends are closed with endcap disks to measure particles with a small angle to the beam axis, see Figure 7.

Semiconductor Tracker (SCT) and Pixel Detector are both silicon detectors that are arranged in seven cylindrical (or barrel) layers around the central interaction region and in 12 disk structures in each forward direction. During LS1, an eighth cylindrical layer has been installed on the innermost position, the insertable B Layer or IBL. Each layer consists of flat modules with overlapping areas to prevent particles from passing undetected between two modules, except for the IBL which has no overlap in z. The previously three, now four layers closest to the interaction region are Pixel Detectors, semiconductors with a total of 80.4 million readout channels.

To reduce costs, the SCT layers located at higher radii of up to 52.3cm still use silicon as active material but are strip-shaped with a pitch of 80µm x 12cm. In order to achieve a good resolution in all dimensions, the strips are arranged in two layers per module on top of each other with a small angle with respect to each other, to derive a 2D position on a module from two strips. The trade-off is a reduced resolution in one direction as well as possible ambiguities. Table 1 shows the resolution of the modules from which the

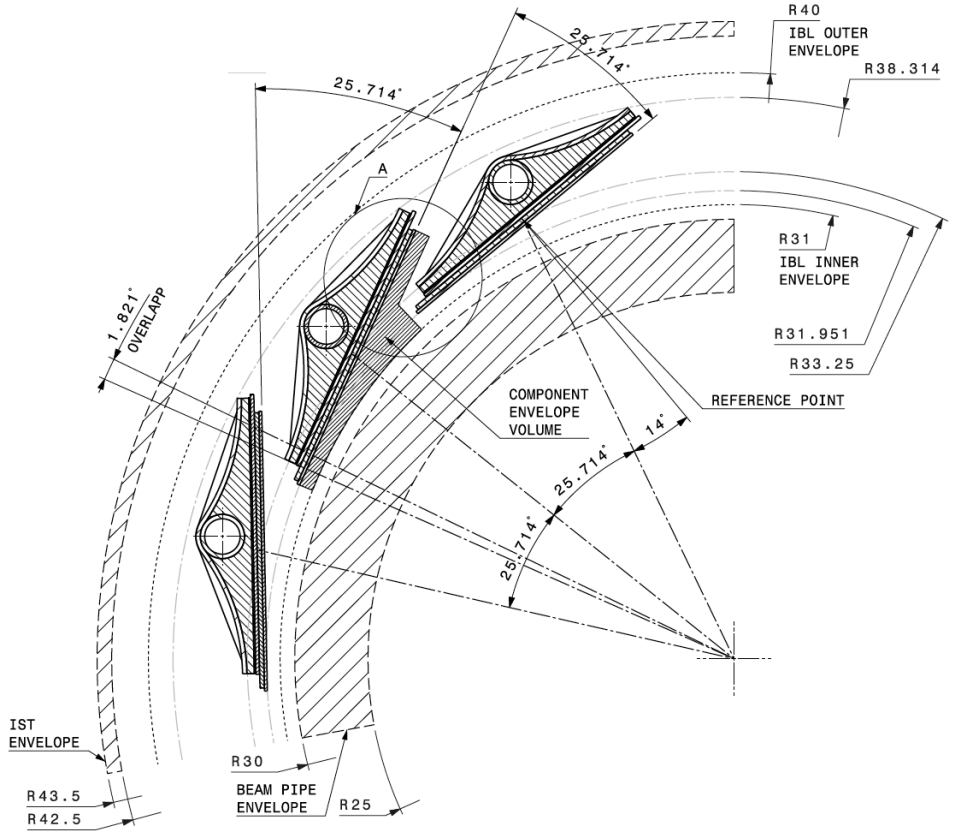


Figure 8: IBL architecture from [26]. Other silicon detectors are arranged similarly around the beamspot at higher radii.

	Min/Max Radius	No. of Barrel Layers	Endcap Layers per Side	No. of Channels	Readout Size	Est. Barrel Resolution φ (x Z) in µm
IBL	3.325cm	1	0	6.02M	50 µm x 250 µm	10 x 50
Pixel Detector	4.7cm/ 12.6cm	3	3	80.4M	50 µm x 400 µm	10 x 100
SCT	29.2cm/ 52.3cm	4	9	6.2M	80 µm x 12cm	16 x 580
TRT	56cm/ 107cm	ca. 36	ca. 40	0.42M	4mm	100-130

Table 1: Inner Detector basic data [2], [26]. In order to make use of the high resolution of the detector modules, the modules' spatial alignment is known with an accuracy of a few micron relative to one another. The approximate number of layers for the TRT gives the average number of straws crossed by a particle originating from the interaction zone. The number of layers is actually up to 73 in the barrel, but more important is that they are aligned such that there are at least 36 measurements per particle.

different detectors are made. Note that in order to establish a 3D measurement the module's position inside the detector has to be used, which is established with slightly lower accuracy than the alignment of the modules relative to each other, which is known up to a few microns. Resolution is in z and φ, with z being the axis along the beam and φ being the angle around the z axis, perpendicular to it. Each module has a side that is closer to z axis,

i.e. has a smaller r coordinate, than the other side, see Figure 8, because of the overlapping arrangement of flat detector modules.

The Transition Radiation Tracker (TRT) surrounds the silicon detectors in both barrel and endcaps and consists of 420,000 straws which can be read out separately. In gaps between the layers of straws, a dielectric material causes particles passing through to emit photons. This effect is used to identify particles, particularly electrons because the number of photons emitted depend on the mass and the momentum of a particle. Each straw contains an ionisable gas and a wire with high voltage in the center. When a particle passes through, the gas is ionised and electrons drift to the central wire. The time taken for the electron charge to deposit in the wire allows measuring a spatial resolution between 100μm and 130μm orthogonal to the straw [27], but neither which side of the wire the particle passed through nor the position along the straw is known. Straws are arranged parallel to the beam pipe in the barrel region and perpendicular in the end-caps. A typical track passes through 36 straws in the TRT in the barrel region. The number of straws crossed in the endcaps varies, depending on the angle of the track with respect to the beam (the so-called θ-angle of a track).

A superconducting solenoid magnet enclosing the ID creates a magnetic field with a strength of 2 Tesla. The field is not completely homogeneous as it slightly changes direction towards the ends of the barrel. Assuming a simplified homogeneous field, a charged particle's path is bent in φ but leaves its direction in the r-z plane unchanged, which leads to a helical path. From this bending, called the track curvatures, both a particle's charge and transverse momentum can be deduced.

2.6 Data to be reconstructed in ATLAS

Tracking detectors are built to localize the intersection of charged particle trajectories with sensitive detector elements. Usually, this is done by measuring the charge induced in either planar silicon sensor (Pixel Detector or SCT) or in the ionization gas of drift tube detectors. These locations (so called hits) for Pixel and SCT detector are shown in green and yellow on the sensitive detector layers in Figure 9 and Figure 10, visualizations of a simulated event as it is read out from the detector. This event has about 40,000 hits in the Inner Detector, which result from approximately 40 proton-proton collisions, which is the expected average during Run 2. Even higher numbers of proton-proton collisions are aimed for in the future, reaching up to 200 collisions per event for the High-Luminosity-LHC [28] in 2023. To deal with these challenges, the detector will be upgraded.

Events are usually triggered by final state signatures. It is possible that multiple signatures are detected in a single event. A signature is usually associated with a high transverse momentum balance. Transverse momentum is defined as the momentum leaving the interaction in an orthogonal direction to the beam. This collision of interest is called signal while all other collisions are referred to as pileup collisions.

Events recorded in 2012 have up to 37 pileup collisions. Higher luminosity has already been reached during Run 2. Increasing luminosity only by increasing the number of pileup collisions causes the reconstruction to become very slow and potentially less accurate due to the high occupancy of the detector. This is why a higher frequency of events is preferred over a higher number of collisions per event by the experiments. The LHC was designed to allow 40 million events per second, corresponding to a so-called bunch spacing of 25ns. During Run 1, a bunch spacing of 50ns was chosen, so that only half the number of bunches were in the LHC, but each with twice the number of protons. This was beneficial because twice the number of protons leads to four times the instantaneous luminosity, i.e. four times the number of proton-proton collisions per bunch crossing. Considering the number of collisions is halved because there are half as many bunches, this corresponds to twice the integrated luminosity. For Run 2 a bunch spacing of 25 nsec was used, yet a

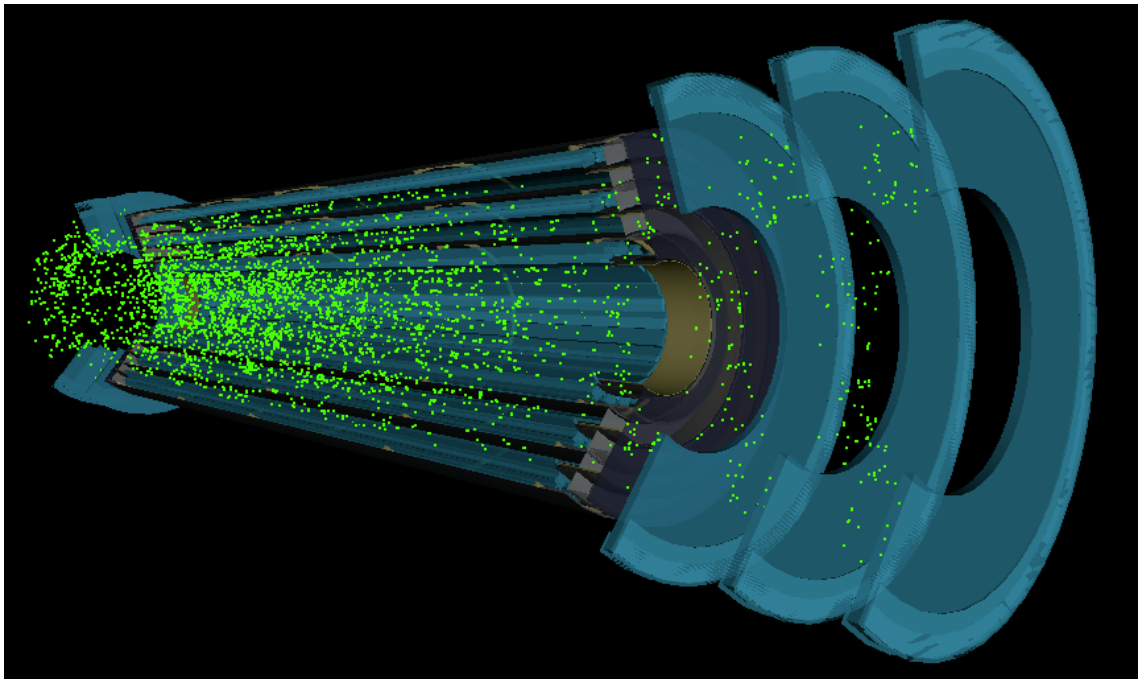


Figure 9: Simulated event with the charge deposits of 40 proton-proton interactions. φ-cut of three Pixel Detector layers and endcaps with hits. Hits not in the φ-cut are occluded by the detector material.

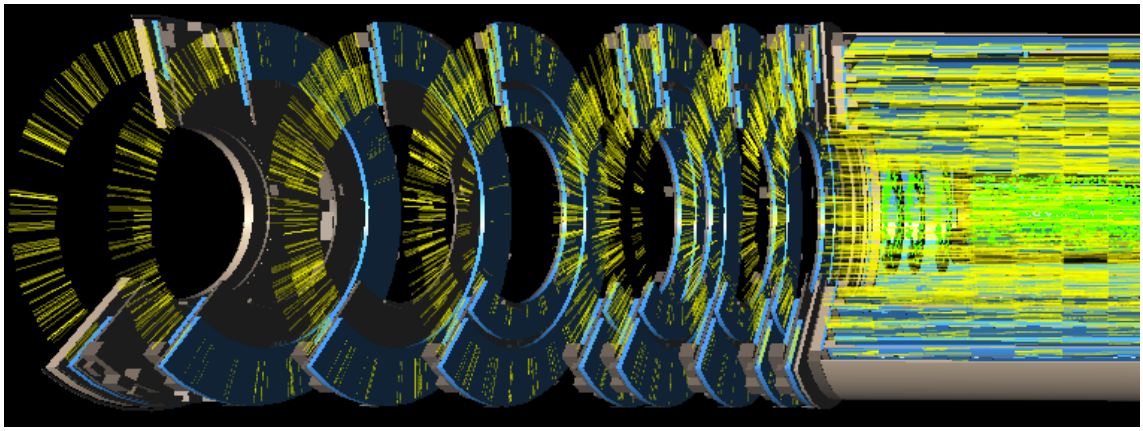


Figure 10: φ-cut of half-length of SCT and Pixel detector with hits in yellow and green. The detector's center is at the right. The green hits are the same as in Figure 9.

higher luminosity was achieved than during Run 1 by changing other machine parameters. The tighter bunch spacing of Run 2 leads to more "out of time pileup", energy deposits from previous events that are read due to the short time difference. Out of the 20 million events per second in Run 1, 400 were recorded. During Run 2, 1000 events per second are recorded, at the same time the events are expected to have a higher average number of collisions than events during Run 1.

2.7 Track Reconstruction

The track reconstruction is the step from raw data from the detector to reconstructed trajectories of particles. As a central part of the ID reconstruction, different parts of this thesis are dedicated to improvements of this step. The reason this thesis puts such a strong emphasis on reconstruction is the near exponential increase in runtime with number of proton-proton collisions in one event, see Figure 11. The average number of

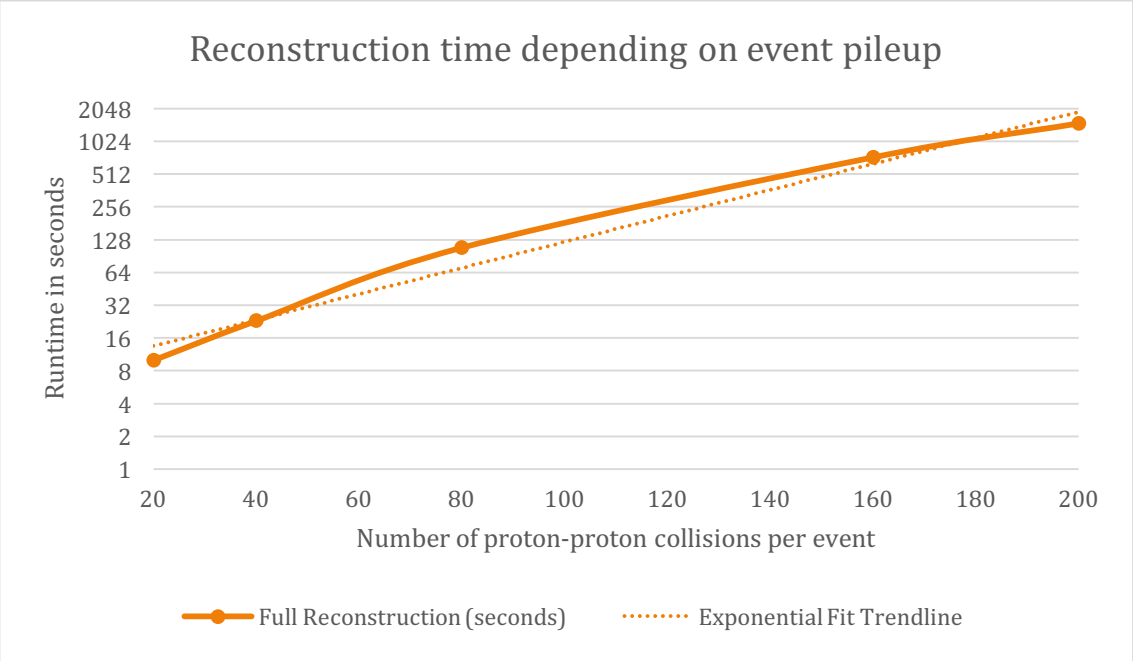


Figure 11: Scaling of reconstruction runtime per event with number of pileup collisions. The y-axis is in log-scale and the curve is fitted to five data points. Data taken from [59].

collisions per event is set to increase to above 40 during Run 2 and up to 200 with future upgrades of the LHC.

Track reconstruction is composed of a chain of algorithms whose internal order is bounded by input/output dependencies. The algorithms used can be categorized into data formation, pattern recognition and vertex reconstruction stage. The order of these steps and their most important constituents are visualized in Figure 12 and Figure 13. During data formation, the measurements from the silicon detectors are grouped in clusters of measurements, which in turn are converted to three-dimensional space points (SP) located on the detector elements. The measurements of the TRT detector are converted to drift circles rather than space points because only the distance to the central wire in a TRT straw can be calculated but not the location along the wire. The SP and the clusters are passed to the track finding [29], [30]. The default strategy has three distinct steps. It starts with the SeedFinder, which tries combinations of SP-triplets likely to have an origin close to the interaction region and returns these triplets as seeds, see the points encircled by continuous lines in Figure 14.

The seeds are passed to a combinatorial Kalman filter [31], [32] to create tracks from clusters spanning all silicon layers. The seeds are used to estimate the initial direction of the particle path through the detector. The Kalman filter uses a Runge-Kutta-Nyström extrapolation engine [33] to predict the path of the particle through the magnetic field and then selects one or more compatible clusters on this subsequent layer or continues without finding any. The Kalman filter selects a compatible cluster using its uncertainty and the probability for multiple scattering. In case it finds a cluster, it uses it to update the error matrix carried over from each surface. In case it finds multiple clusters, the Kalman filter splits up the track into multiple tracks and continues for each as if only one cluster had been found, hence the combinatorial complexity of this algorithm. The result of this processing step is track candidates. Clusters can be allowed to be used by multiple tracks if they exhibit certain properties. For each cluster, it is stored how often it has been used in a track candidate. This information is used in the next processing step.

The next step is to resolve ambiguities of track candidates sharing measurements. Track candidates are rated by different criteria. The rating penalizes a track using the

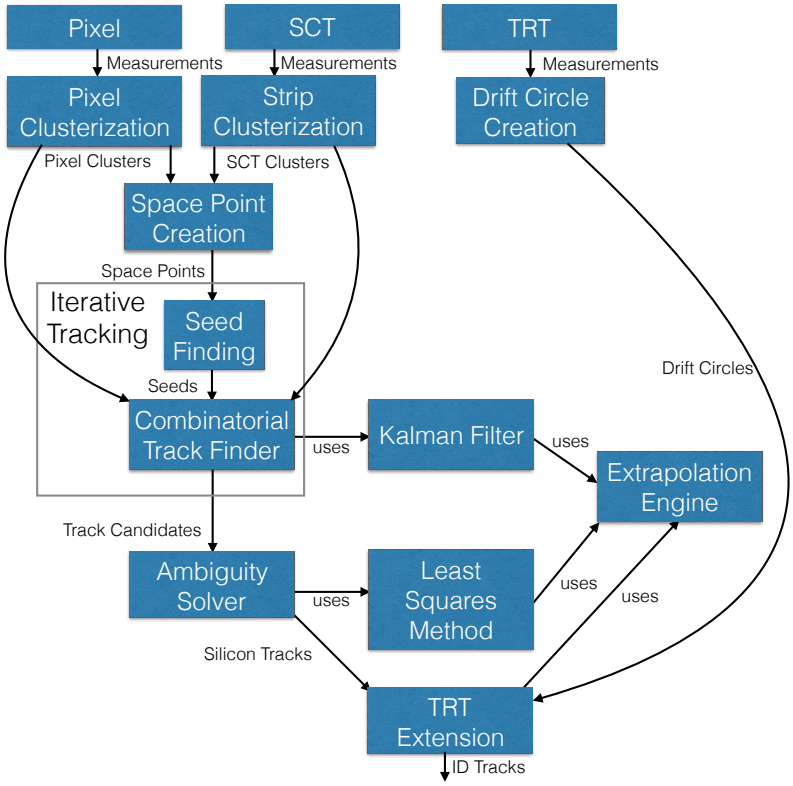


Figure 12: Diagram of the steps from Inner Detector readout to tracks. There are many other highly specific reconstruction algorithms to better reconstruct some physics objects. The combinatorial track finder runs for seed that has been found. A bookkeeping mechanism keeps track of space points/clusters from which a track candidate has already been formed. Smart selection creates seeds first that allow the combinatorial track finder to find likely true track candidates first.

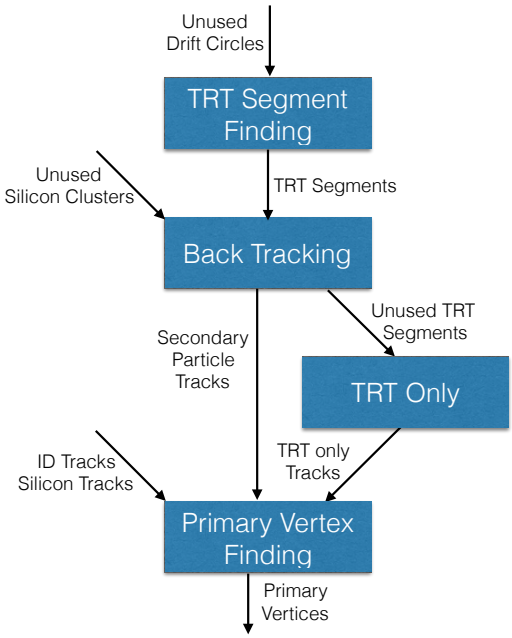


Figure 13: Additional ID reconstruction algorithms. Inputs from Figure 12. "Unused" refers to parts of the data that have not yet been used in a previous algorithm shown in Figure 12.

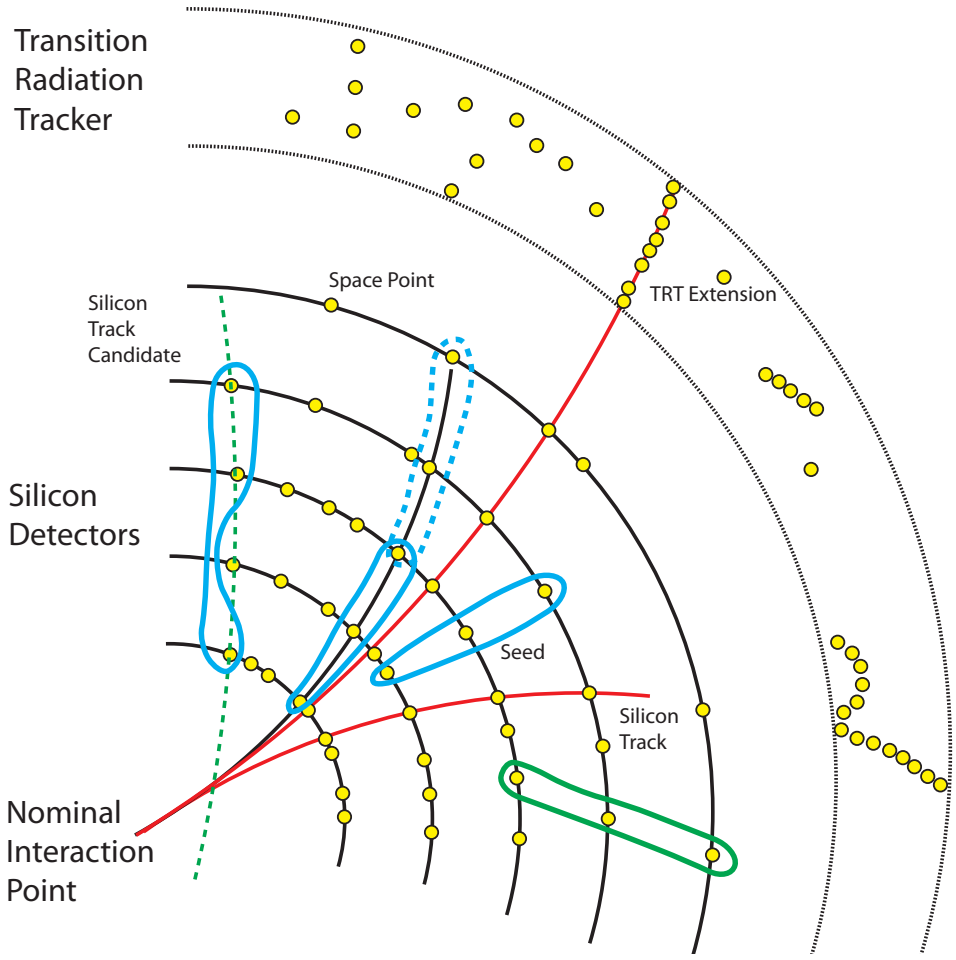


Figure 14: Inner Detector track reconstruction. The graphic depicts the different steps in default ID reconstruction in a simplified model of the Inner Detector. Hits are shown in yellow. Seeds created from hit combinations are encircled by continuous lines. The dashed blue line shows a case where two seeds are created from hits from the same particle. The green continuous line shows a seed rejected because it could not form a track coming from the interaction region. Similarly, the green dashed line corresponds to a track candidate rejected because a track formed with hits compatible with the direction of the seed do not point to the interaction region. The red lines and the black line denote fully reconstructed tracks, with the upper red line showing a track with hits in the TRT. [34]

number of layers the track candidate crosses for which no suitable cluster was found (holes). The second criterion is the quality of the fit defined by the X^2 fitter using the distance of the clusters to the track candidate. The tracks are then sorted by rating. Starting with the highest rating, clusters assigned to track candidates are marked as used. If a track candidate is found to use a cluster already used by a higher rated track (and which has not been allowed for use by multiple tracks), this cluster is removed from the candidate. If the track candidate has enough clusters left, it is then removed from the sorted list and reinserted in the end for re-evaluation. If not enough clusters are left, the track is deleted.

The last step of this stage is the extension into the TRT, connecting the silicon tracks with the drift-circles in the TRT. The TRT measurements are used extend the found silicon tracks by searching for hits in a likely region of the path of a particle, adding them to the track.

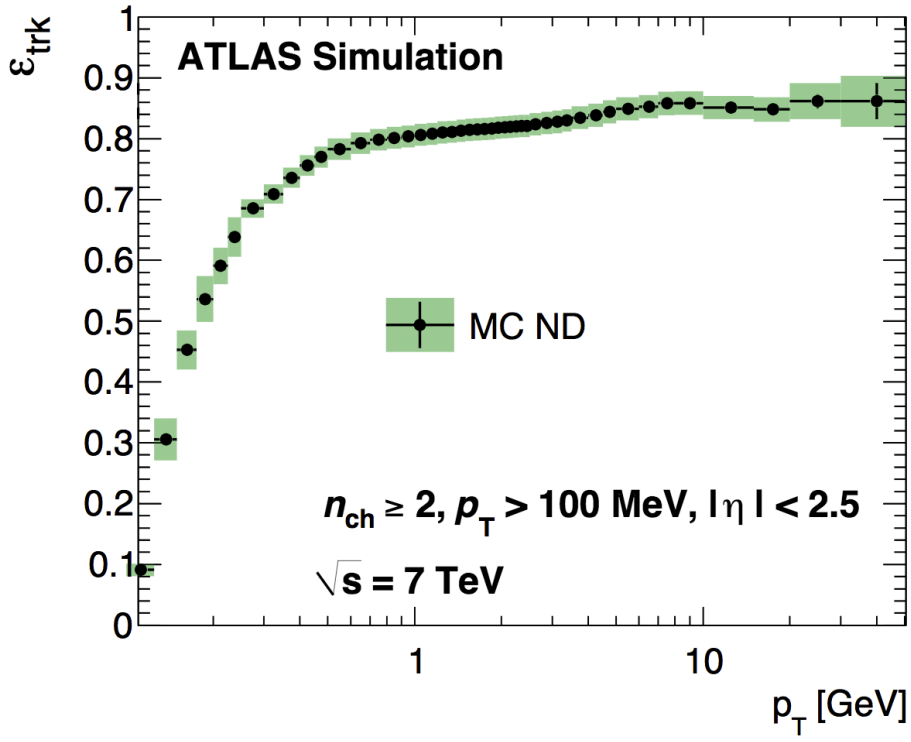


Figure 15: Tracking efficiency as a function of the transverse momentum p_T. ATLAS default reconstruction settings do not attempt to reconstruct tracks with p_T of less than 0.4GeV/c. The maximum azimuthal angle is +-9.4°, corresponding to pseudorapidity |η| = 2.5. [35]

In the next stage, the tracks are extrapolated inward towards the interaction region, defining their point of closest approach as their origin. If two or more tracks have their origin in the same location, this location is defined as a vertex position in a subsequent step.

This collection of algorithms allows for a rudimentary reconstruction but does not yet include more sophisticated methods.

2.7.1 Reconstruction efficiency

For analysis, knowing the energy of a reconstructed physics object is important. There are different methods to measure this energy in ATLAS. Often, the most interesting physics objects decay before they can leave a track in the detector, so only the sometimes multiple decay products can be observed. Each of the decay products holds part of the energy. To reconstruct the energy of the original object accurately, it is therefore necessary to reconstruct the tracks of as many of the decay products as possible. The ID can directly measure only the transverse momentum (p_T) of a particle because the magnetic field bends the particle tracks in transverse direction. Tracks with low p_T are not reconstructed for a number of reasons:

- They usually carry little information about the triggered collision event which is usually associated to a high energetic signature
- The tracks are very costly to reconstruct due to their sheer number, leading to an increased number of possible combinations
- Very low momentum particles suffer very strongly from interactions with detector material and thus the resolution of certain track parameters, such as the p_T, is poor
- The large number of low energy particles makes storing such tracks too expensive

This is why ATLAS reconstructs only particles above a certain threshold, the cut level, usually $p_T > 0.4\text{GeV/c}$. The track reconstruction efficiency is given as the fraction of actual particle trajectories which are reconstructed as tracks. This metric can be used for simulated events where the particles causing a measurement are known. Simulated events are suitable to measure the efficiency of a tested algorithm. The p_T of a particle is calculated using the curvature of its track. Due to the material effects, this is particularly inaccurate for low p_T. This is why the efficiency drop in Figure 15 is not steep at 0.4GeV/c, but is smeared into both higher and lower regions. Particles wrongly classified as having a lower p_T than the threshold will not be reconstructed, and particles with p_T below threshold whose p_T is overestimated are reconstructed.

Reconstruction runtime is mainly affected by the number of measurements. The change from 8 TeV at the end of Run 1 to 13 TeV in Run 2 has only a minor effect on reconstruction runtime because the number of particles per collision only increases within jets for higher collision energies. The higher number of simultaneous proton-proton collisions are responsible for creating a much larger number of additional particles compared to the increased collision energy. The cut levels strongly influence reconstruction complexity, as low-energy particles make up a large fraction of the particles in the detector. Other effects also contribute to the increased runtime. With lower momentum, material effects play a larger role such that the path of lower energy particles may diverge much more, and during reconstruction a higher number of particles have to be considered for continuing a track on subsequent layers. And lastly, reconstructing a larger range of transverse momentum corresponds to a larger range of curvatures. This means a much larger opening angle for possible combinations of measurements leads to an increased amount of combinations, both from actual tracks and from unrelated measurements, significantly increasing the runtime.

2.8 Software used by ATLAS

For operation, ATLAS requires several software suites to run. There are multiple steps requiring different software, the first being the online reconstruction. It is part of the software trigger [36], which uses the online reconstruction results to further filter the events that have been selected by the hardware trigger. Like the hardware trigger it is a real-time system, used to reduce the number of events to an amount that can be processed and archived. Due to the real-time requirements, this software called high-level trigger reduces the required work by using a simplified geometry and reconstructing only regions of interest (RoI), effectively restricting the reconstruction to the area where a signal has been detected. If defined criteria are met that hint towards certain physics events justifying further analysis, the event is then recorded on disk and later archived on tape.

The offline reconstruction does not have the same time constraints and can therefore reconstruct more precisely using more complicated methods and the full detector geometry details. Contrary to several trigger steps that constrain the event reconstruction to RoIs, the offline reconstruction works on data from the full detector. The final result is stored in a data format containing only data required for analysis. This data format has limited information about the original measured data, instead it contains an interpretation of this data in the form of the reconstructed physics objects. In case the original data is required, e.g. because algorithms have been improved or because more accurate information about the detector conditions become available, reconstruction has to be executed again using the original data from the detector.

In order to confirm or reject a theory or to measure the mass of a particle, every physics analysis compares simulated, i.e. expected behaviour with observed behaviour. The simulations predict the detector response to new and already known physics events. The simulation of the detector response to particles is therefore a key tool for particle physics.

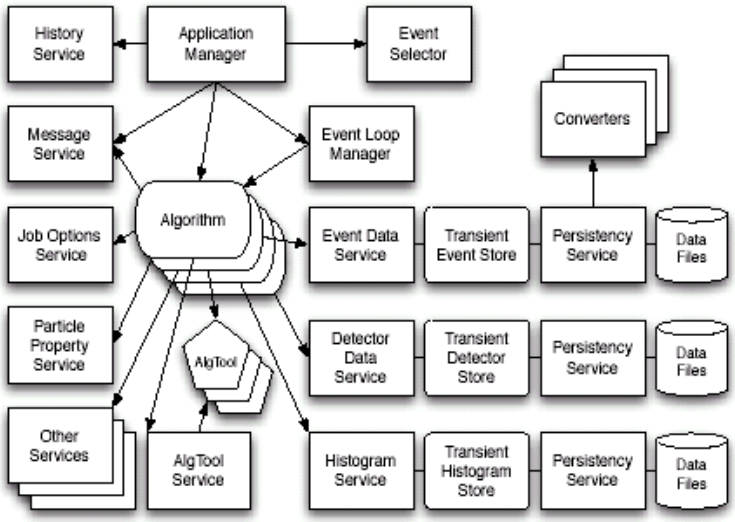


Figure 16: Architecture of the Athena Framework. [3]

To generate simulated data, no event data is needed, but algorithms simulate particles according to different models, most famously the Standard Model. The algorithms then generate the response of the particles to interaction with a model of the detector and the detector response. This output is similar to the event data format so it can be compared with generated events. It is reconstructed in the same way data from the detector is reconstructed. Simulation shares some algorithms with offline reconstruction so that improvement in either may benefit the other.

This thesis will primarily concentrate on reconstruction, although some of the presented work also affects or is applicable to trigger or simulation.

2.9 The Athena Framework

The main processing framework of the ATLAS experiment is called Gaudi-Athena. It is used for reconstruction, simulation and high level trigger, and all code development in the course of this thesis was done in the context of this framework. It is based on the Gaudi framework jointly developed with the LHCb experiment [3]. Development on Athena started in 2003 after the release of Gaudi in May 2000 [37]. Its design reflects different requirements of high-energy physics experiments on software. It is used for event simulation and reconstruction, and to a smaller extent also for analysis. These three steps have different computational profiles. Event simulation is compute bound while analyses typically run over millions of events and are I/O bound. Reconstruction is in between, leaning towards being more CPU bound.

Across all processing steps, the smallest unit that constitutes a processing job is a single collision event, henceforth just “event”. Each event is processed independently from other events. Both reconstruction and simulation process data in many different stages. To easily compose the required components for different stages, the framework is highly modular, allowing mixing experiment-specific modules with generic ones. It loads components through dynamic libraries such that they can be developed and compiled independently of one another and combined as required. The complexity of each of these stages is encapsulated by keeping data in objects separate from the algorithms, such that one stage can be unaware of the complexity of a previous stage. The general architecture is shown in Figure 16. Data processed by the algorithms is accessed through a single interface called StoreGate, which is the only foreseen method for communication between components. Static data, such as the magnetic field, is accessed through different services.

Maintaining the number of common interfaces small allowed to achieve a low coupling between the components. In some cases, there are specialized components which can replace another component. The algorithms need to be configurable without requiring recompilation because the requirements on the configuration of the components varies. This configuration is done via Python files, containing the configuration for one or many components, which can be in turn imported in other Python files. The configurations are so diverse and complex that they make up more than 20% of the ATLAS code [4]. Since only a tiny fraction of this code is executed for each job, the time spent in this python configuration is marginal.

An Athena component can be one of three different types of modules: An Algorithm, Service or Tool. Each type implements its own interface. An Algorithm has to implement the AthAlgorithm interface, requiring it to provide an initialize-, an execute- and a finalize method. The initialize method will be called once before the first event, the execute method will be called once for each event and the finalize method will be called once after the last event. Each Algorithm defines a step in a chain of steps to be executed. An Algorithm can use Tools and Services. Each instance of a Tool is owned by another module and Services are singletons. Both Tools and Services have initialize and finalize functions analogous to Algorithms, but can have arbitrary other public methods that can be called from Algorithms or other Services and Tools. Services are singletons directly owned by the framework. Services normally provide functionality required by many different modules, such as the StoreGate Service [38], which is used to access and create persistent and transient data and is the standard way for algorithms to communicate. Other frequently used Services provide access to random number generators or allow retrieving the magnetic field strength at arbitrary points within the detector. A Tool has to have an owner, which can be another Tool, Service or Algorithm and is by default owned by the Service AlgToolSvc. Different algorithms can share a single Tool instance via the AlgToolSvc, which is in some cases abused to communicate between algorithms.

Athena runs with a single worker thread which executes all Algorithms in the order defined in a Python configuration file, called the “job options”. The algorithms therefore have to be configured in an order that satisfies the sequential dependencies of the job. All configuration of modules is also performed through these job options. An Algorithm e.g. may require a certain type of Tool, the particular implementation of which is specified here, or the default value of a member variable defined as property can be changed. Input and number of events to be run are also defined here. The chain of algorithms is initialized once per job and is executed once per event. They are not expected to have a state, which means a previously processed event shouldn’t influence subsequent events. This is important for parallel execution. It allows events scheduled for reconstruction or simulation jobs to be split up arbitrarily, such that they run in tens of thousands of jobs in parallel with each job processing thousands of events sequentially, without influencing the result.

With the start of Run 2, AthenaMP [39] is used for bulk processing instead of Athena. AthenaMP is a modification of the original Athena framework, which forks multiple Athena processes to process multiple events in parallel on a multicore machine. As multiprocessing is transparent to the modules, no changes to the user code are required.

2.10 Computing Infrastructure

The computing infrastructure comprises all computing resources available to ATLAS. It is where all processing of the data generated by the detector and simulation takes place. Processing starts at the so-called trigger farm, which consists of machines dedicated to the selection of events, and is only available to ATLAS. It is physically located next to the detector and is connected to the readout of the detector by optical links to transport the

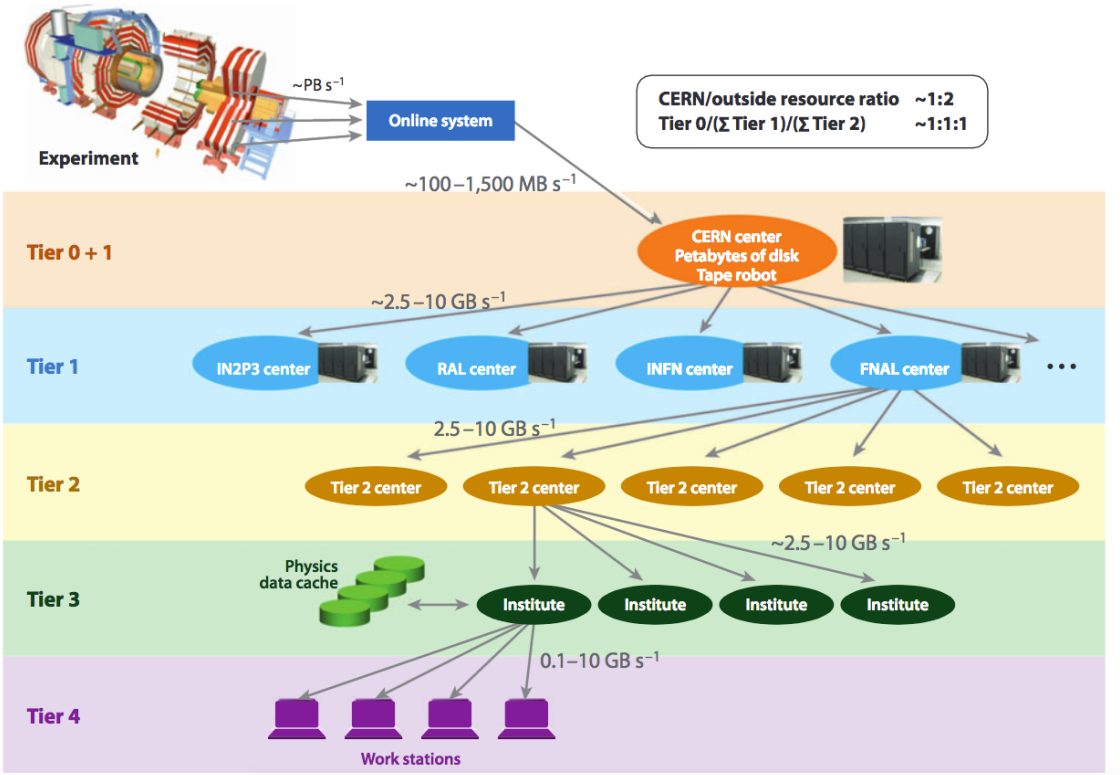


Figure 17: Original MONARC grid tier structure as proposed in 1999 [40]. The different tiers have different roles and are ordered from close integration with CERN or CERN experiments to being loosely integrated. Each experiment has adapted this model for its need. All experiments implemented some less strict form of this model and allow communication crossing more than one tier border.

event data stream arriving at around 15kHz. The trigger farm is not available for other tasks during normal operation, but can be used as an additional computing site during shutdown. It has in the order of 10,000 physical cores from different CPU generations [41].

The other available computing resources are shared among all experiments and are organized in the Worldwide LHC Computing Grid (WLCG), forming the largest scientific computing grid worldwide [5]. Most sites are managed using middleware from the European Middleware Initiative, while sites in the US or northern Europe are managed using other middleware [42]. Apart from being used for simulation and reconstruction, which are coordinated centrally by each experiment, thousands of scientists all over the world access the WLCG every day to perform analyses, accessing huge amounts of data. Four tiers are distinguished, which are hierarchical in that a site distributes data to sites from the same or a higher tier number.

Tier 0 is based directly at CERN and also in Budapest, Hungary. The new Tier 0 site in Budapest has a very fast connection to CERN, with a redundant 100Gbit connection and a low round trip time of under 25ms. Tier 0 is used for first-pass offline reconstruction of the generated events. Due to the requirement that no backlog of events may pile up in the processing, this tier is not part of the batch system, meaning that no user jobs can be directly scheduled on this tier. Tier 0 is the smallest tier but has the largest sites, contributing around 20% of the total computing budget [10].

Tier 1 is composed of the computing sites of mostly national physics laboratories while Tier 2 comprises the computing resources of partner universities. Tier 3 are computing sites which provide computational resources but did not sign a memorandum of understanding defining the service level agreements. Therefore, the size, availability and service quality varies from site to site. Tier 1 and Tier 2 traditionally have distinct responsibilities due to differences in availability and connection, but with improving

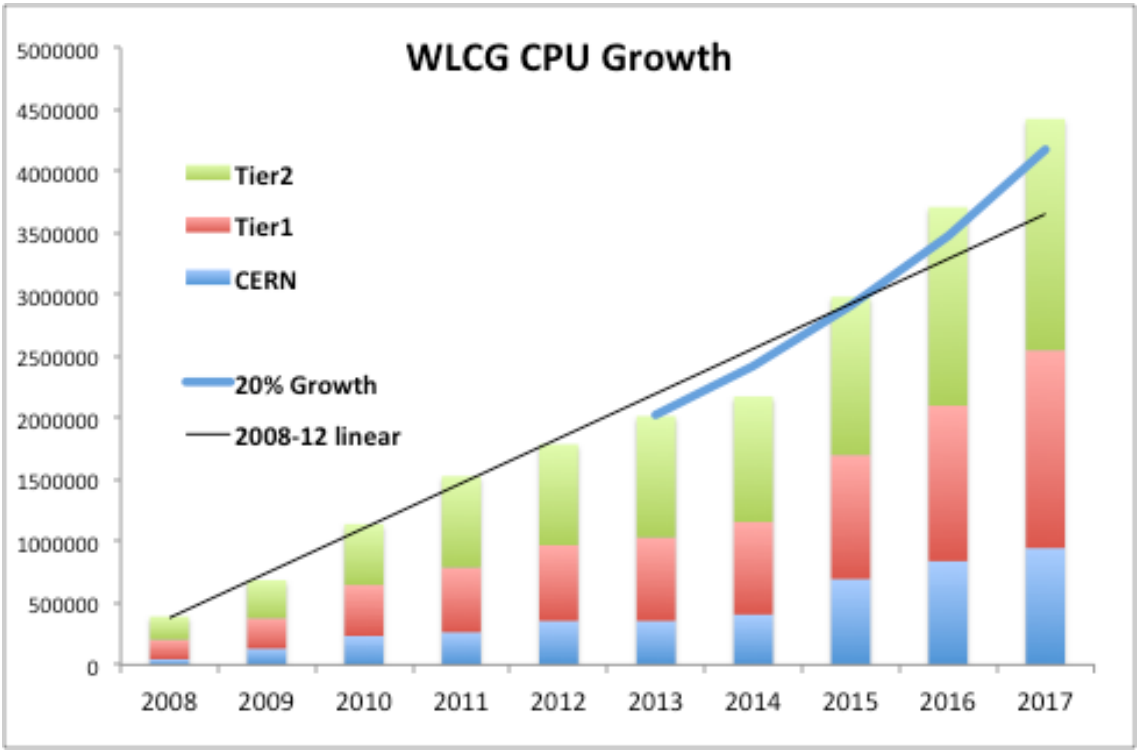


Figure 18: Extrapolated CPU growth assuming flat budget for all Tiers from 2014. The 20% increase is expected with the anticipated technologies. Performance measured in HEPSPEC06. Plot taken from [10].

connection speed and quality this evolved to a model where the boundaries of the roles between Tier 1 and Tier 2 are less strict. The original model can be seen in Figure 17. Tier 0 and Tier 1 provide tape storage to maintain two full backups of all data passing the last level trigger of all experiments. Tier 1 exclusively absorbs shortages in Tier 0 resources ("spillover"). All raw data generated at CERN are archived both at CERN and distributed over all Tier-1 sites. Otherwise, Tier 1 and Tier 2 are used for offline reconstruction, Monte Carlo sample generation and physics analyses. There are 13 Tier-1 sites directly connected with optical fiber to guarantee high bandwidth and availability. Tier-2 sites are connected over the Internet and comprise 160 sites of different sizes. Ongoing tests with opportunistic use of high-performance-computing (HPC) capacity from different collaborations, such as the SuperMUC in Munich, provide another source of computing resources.

Tier 0 being the only Tier managed by CERN consists of approximately 65000 cores. It offers 4GB of memory per core, unlike the other tiers which until before Run 2 only allowed to run jobs occupying up to 2GB per core. Since the start of Run 2, memory restrictions have been loosened. The computing budget is not foreseen to change in the near future. The budget of 41.2 million CHF for 2015 has to cover manpower, energy cost, networking and acquisition of computers and storage capacity [43] for computing resources managed by CERN. Energy cost is a major factor, although the energy to performance ratio has been improving linearly. Performance of these machines is measured in HEPSPEC06, a variant of the SPEC benchmark conceived to mimic typical high-energy physics workloads [44] [45]. Measurements with these benchmarks show that available performance increased steadily, with 20% increase expected in the next years, see Figure 18. Hardware resources in the Tiers consist of Intel and AMD x86 CPUs.

Hardware available on the market is becoming more heterogeneous, low energy ARM processors and GPU based accelerator cards are pushing into the server market with competitive energy to computing power ratios. Although there are ongoing discussions how to adapt to the evolving hardware developments, there are no short-term plans to include different hardware architecture on a large scale. One of the reasons is that it would

require maintaining multiple versions of the software which is intended to run on different hardware or to make sure the same hardware is available on all used computing sites.

3 Software Development and Computing in ATLAS

The concepts and background explained in Chapter 2 give an overview of the problem scale and environment and demonstrate the complexity of the problem. With this information, the problem description given in Chapter 3 is put into a context.

In this chapter, the problems addressed in this thesis are presented and their complexity demonstrated. The state of the software before the start of this thesis and the historical developments and the current development process of the ATLAS software are described. Changing requirements and recent changes in available resources are explained and put into context with the goals for LHC's Run 2.

3.1 Correlation of Beam and Collision Settings and Processing Load

The LHC design peak luminosity was exceeded in 2012, yet available computing resources were able to cope with the increased amount of data (though getting close to their limit). As will be explained in detail in Section 3.2, reconstruction runtime increases rapidly for higher pileup. This makes the processing load strongly dependent on the number of interactions per recorded event. With the number and complexity of events generated during Run 2, the software from the end of Run 1 could not stay within permissible resource limits. The computational load increases for several reasons. Both the peak luminosity and the integrated luminosity increase in Run 2 compared to Run 1. Higher peak luminosity means that the maximum number of instantaneous proton-proton interactions is higher than during Run 1, leading to more measurements per event. This is important for reconstruction on Tier 0, which must fulfill the real-time requirements. The luminosity is at a maximum at the beginning of a data taking run. An example of this is shown in Figure 19, showing the luminosity development over about 13 hours of data taking, from the initial crossing of beams until dumping of the beams to refill the accelerator. Because events in the beginning of a data taking run take much longer to process than events taken later, a backlog of events will accumulate, which can only be reduced later during a data taking run when the events take less time to process. The higher center of mass energy also leads to a higher number of particles produced in each individual

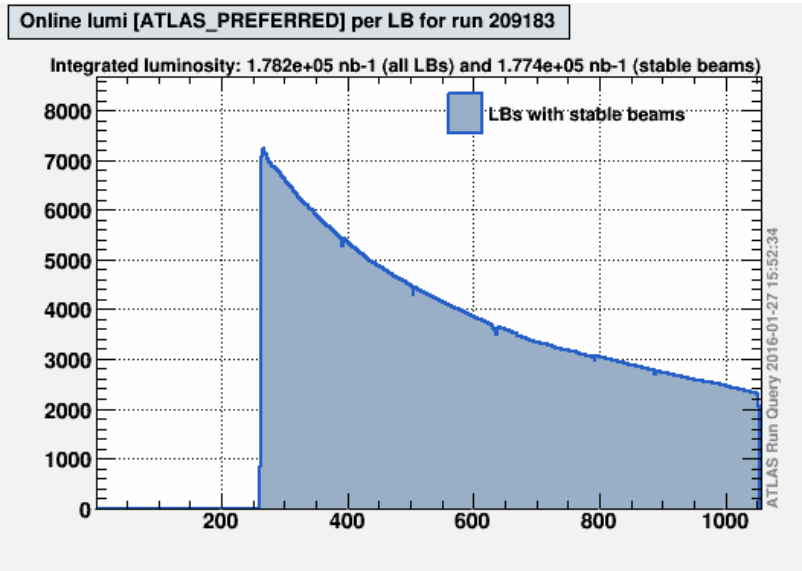


Figure 19: One data taking run with a peak luminosity that corresponds to 33 pileup interactions. Lowest number of pileup interactions is 10 before the beams get dumped. Data is binned in so called lumi-blocks, which usually correspond to 60 seconds. Plot generated with the ATLAS Run Query page.

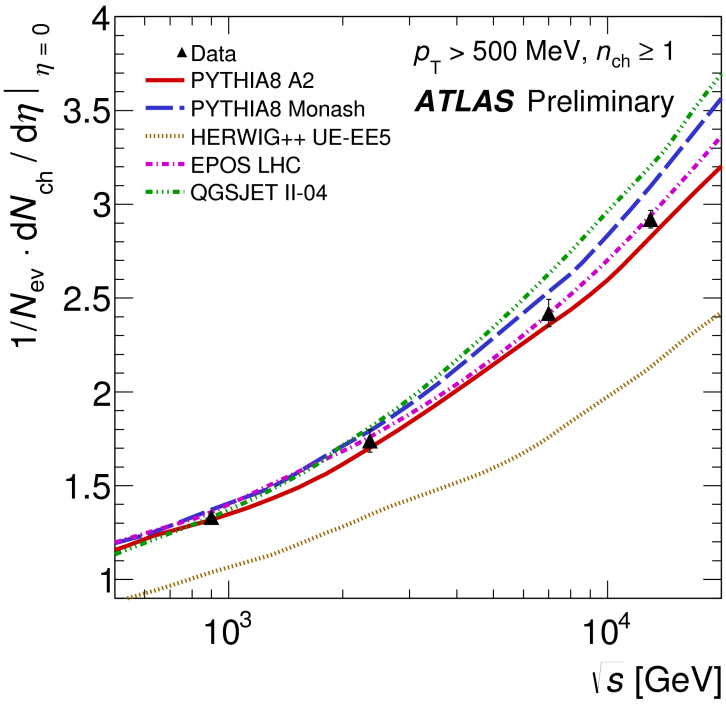


Figure 20: Number of particles with 900GeV, 2.36TeV, 7TeV and 13TeV center of mass energy in the central detector region. The step between 7TeV and 13TeV, the two rightmost data points, shows about 20% increase in the number of particles. Plot from [46].

collision, see Figure 20. Some algorithms have very high computational complexity, which means their runtime increases superlinearly with the size of the problem. Therefore, their runtime will increase more than algorithms with a lower complexity when the number of measurements per event increases.

3.2 Track Reconstruction Complexity

My tests show that reconstruction runtime of events expected for Run 2 is about seven times slower than reconstruction of events as seen in Run 1. This section explains the algorithmic complexity of central track reconstruction principles and their interplay with other algorithms. The algorithms used in the event reconstruction have differing complexities, which are reflected in their runtime. Within reconstruction, the largest amount of time is spent in Inner Detector algorithms as it is faced with the most complex pattern recognition problem to solve. For events with an average of 40 pileup-interactions approximately two thirds of the total time is spent here with the event reconstruction software of 2012, a large fraction of which are track reconstruction algorithms. Reconstruction time not spent in track reconstruction is distributed over hundreds of Athena modules with each module responsible only for a tiny fraction. Track reconstruction complexity increases with the space point density with $\mathcal{O}(n^8)$, see Figure 21. In a detector with an arbitrary number of layers, to find *all* possible tracks, all n space points would need to be combined with all others, leading to n! combinations. If a track should have exactly one measurement on each layer, the number of combinations drops to $(n/8)^8$, assuming the space points are equally distributed over all layers. The base can be further reduced by constraints on where the track originates from, but the exponent remains. In addition to having a high complexity, the track reconstruction uses expensive calculations. To fit a potential trajectory through space points from two layers, multiple fourth-order Runge-Kutta extrapolation steps are performed, making track reconstruction computationally very expensive.

One way to reduce the runtime could be allowing less costly and in turn also less accurate methods. The efficiency of tracking must not be reduced, such that the only way to allow this would be to increase the number of found combinations. These additional combinations are necessarily not from related measurements in that the measurements of one combination did not stem from the same particle. This is also called a fake track or just fake. Changes to algorithms that influence the results may have unforeseen implications because an algorithm's output serves as input for the next algorithm in a job's algorithm chain. Changes in one algorithm that change its result may therefore influence the runtime of the subsequent algorithms. For instance, although the runtime for the seed finding algorithm can be improved by allowing more seeds in the results that contain measurements that do not stem from the same particle (fake seeds), it will increase the runtime of the track finder, which uses these seeds. Decreasing one algorithm's runtime may therefore result in an overall performance reduction if it increases subsequent algorithms' runtimes. Such implications favor developments that do not affect the results, which can only have a limited impact on runtime. If a change affecting the results is to be performed, it requires a good understanding of all affected code parts, which is difficult due to the size of the code base, and a good understanding of the affected physics analyses.

Huge efforts have already been put into improving the algorithmic complexity of track reconstruction, yet the amount of space points per event still has superlinear influence on the runtime. Despite some algorithms having a higher complexity than others, no bottleneck can be pointed out in reconstruction. All aspects of the event reconstruction have to be optimized for significant effects, as my measurements presented in Chapter 4 show. One way to influence many different parts of the reconstruction is to optimize infrastructure shared throughout the reconstruction. The Event Data Model is where the data format of the intermediate and final results is defined, such that it is used throughout the different stages of data processing. It therefore serves as a good starting point for optimization.

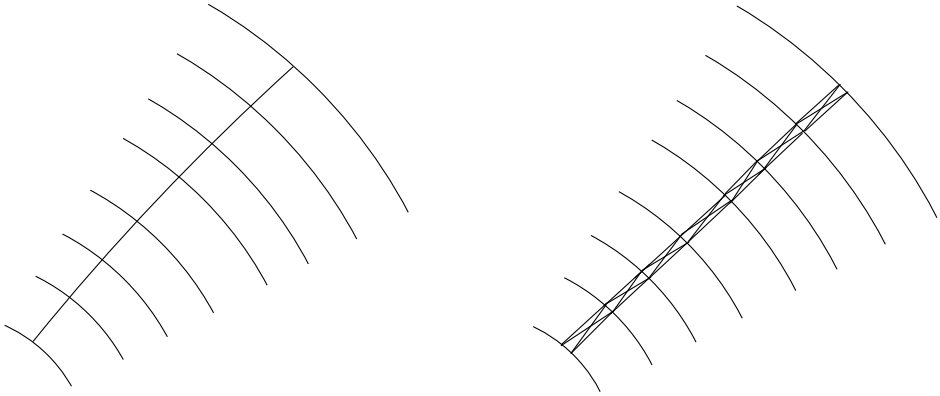


Figure 21: Illustration of the increasing combinatorics with increasing number of particles. On the left side only one particle in the shown detector region allows only one space point combination. As the right side demonstrates, just two particles close to each other already create $2^8 = 256$ possible combinations in a detector with 8 layers when trying to reconstruct one track.

3.3 Event Data Model

The Event Data Model or EDM describes the different data representations required for all stages of data processing. For each representation, persistent and transient data formats exist. The purpose of each persistent data type is to hold the data for the next processing step as compactly as possible. The detector outputs RAW data containing the local measurements in each detector module together with a channel identification and potential error flagging. Its size is between 1MB and 5MB per event depending on the event topology and pileup. Storage size is critical due to the large data output of the detector, amounting to 3.4 Petabytes in 2012 only for the events selected by the ATLAS last-level trigger [47], which have to be redundantly archived. Such amounts of data generate significant cost for storage. While processing this data, it is converted into new data formats in each step. Each subsequent data format reduces data further such that conversion between data formats is only possible in one direction. The subsequent data format after RAW data from the detector is Event Summary Data (ESD), which contains a detailed output of the reconstruction. It contains e.g. the measurements on a track, which can be used to refit tracks under different assumptions. The smallest data representation with all reconstructed physics objects is the Analysis Object Data (AOD) that is sufficient for physics analysis. Smaller custom data formats abstract further or leave parts of the information out. The AOD format contains only parameters needed to identify each physics object and to estimate its quality. For tracks, this is only the vector of track parameters at the so-called perigee, which is the point closest to the interaction zone where the interaction is assumed to have happened. It also contains measures for the track quality, the number of hits found for this track and their associated error matrices. During the LS1 significant effort has been taken by the ATLAS collaboration to shift to a newly designed xAOD format that is based on a flat tree-like data structure. This structure follows the structure of arrays memory layout and allows a dual use of the xAOD within the Athena frameworks and the most common analysis framework in high-energy physics, Root [48].

3.3.1 CPU and Memory consumption

In 2012, the offline reconstruction was responsible for around 22% of the total CPU time spent by ATLAS (see Figure 22) and therefore a significant contributing factor to

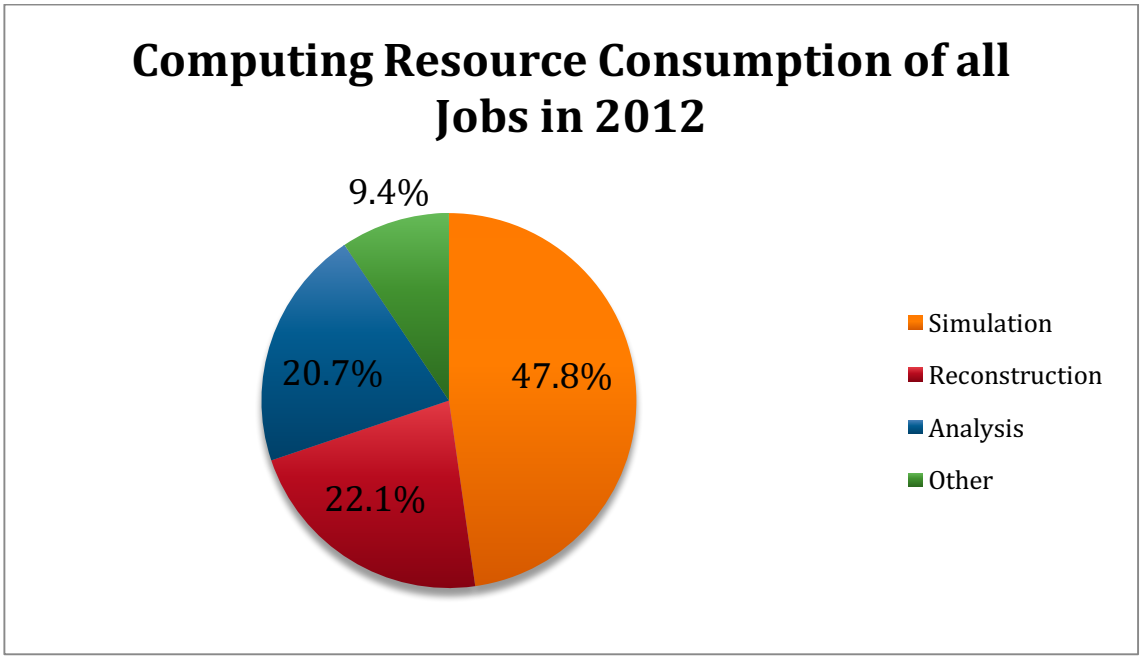


Figure 22: Time consumption of different jobs in ATLAS. All reconstruction jobs combined take about 22% of total time. Most of this time (17% of the 22%) is spent reconstructing events generated in simulation jobs. Data collected using the ATLAS Dashboard.

computing costs. Using the ATLAS dashboard [49], I deduced that this corresponds to 859 million HEPSPEC06-hours in 2015, with an average of 10 HEPSPEC06 per core meaning 86 million hours of CPU-time. Virtual memory usage during Run 1 had to be kept below 2GB as the agreement between CERN and the WLHC tiers only required providing 2GB per CPU core. Therefore, computing sites were configured to cancel all jobs exceeding this limit and only low capacity was available for jobs requiring more resources. Attributing memory to certain parts of the software is inaccurate because many objects in memory are shared between modules. It is tedious to monitor which objects are shared and it is not clear how these shared objects should be attributed. Instead, memory usage changes are monitored. The ATLAS nightly build system, explained in 3.4, runs checks on the software compiled with the changes of the day, so a change in memory from one build to the next must come from one of the packages updated that day. The amount of leaked memory is monitored and attributed in the same way. This is the only way memory is attributed to modules. Memory available per core is higher for Run 2 than Run 1, and while a single instance of the reconstruction software also uses more memory in the software for Run 2, overall memory usage could be reduced by running multiple events in parallel by forking Athena processes, exploiting Linux' built in copy-on-write mechanism. This is what the AthenaMP framework does, which is the standard framework for Run 2. Nonetheless, memory will continue to be an issue in the foreseeable future as memory cost does not decrease with the same rate as the cost per computing core, and therefore memory becomes scarcer with increasing number of cores per machine.

3.4 Software development in ATLAS

The reconstruction software has grown over decades of development, starting even before ATLAS was founded, as before its approval simulations had to be conducted for the ATLAS design studies. Pre-Athena code was written in Fortran but Fortran was abandoned with the introduction of Athena in favour for C++ in 2000. Much of the code was converted to C++ over several years while some modules were only converted recently.

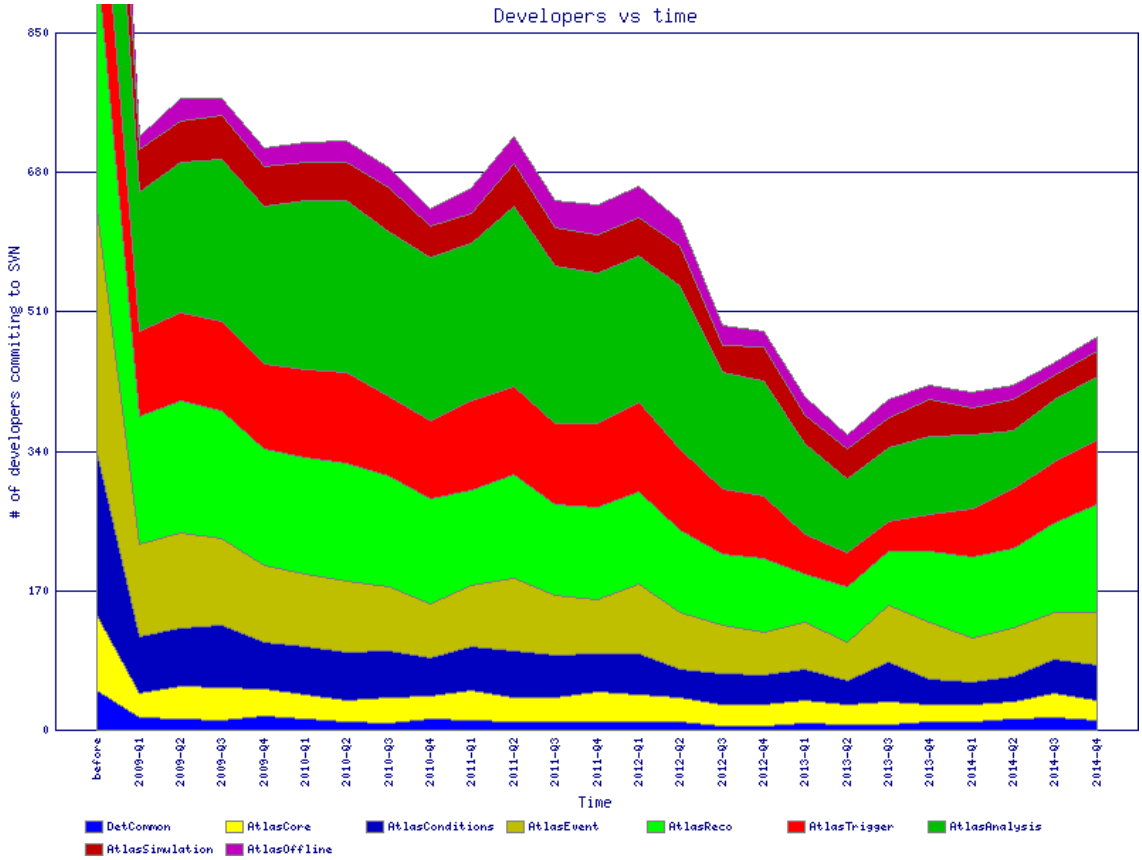


Figure 23: Developers with at least one code submission per quarter. The colors refer to the different development domains. Since the end of Run 1 in 2012 the number slowly recovered, but is still far from what it was in 2009 [4].

Thousands of modules have been written since then by thousands of developers, most of whom are not part of ATLAS anymore. Many of the developers were PhD students who do not stay with ATLAS after finishing their studies. This leads to a quick developer turnover and abandoned code. The current ATLAS svn repository contains 455 Athena algorithms in 2293 packages and 7 million lines of code [4]. An analysis of the SVN logs shows that the number of developers committing code at least once within 3 months has seen a steady decline since the start of Run 1 (see Figure 23), reaching below 50% of the previous 750 developers per quarter at the beginning of LS 1. Most developers have a physics background with little focus on computer science, although the small number of people who are responsible for the majority of svn activities (see Figure 24) have experience in software development and follow the ATLAS coding rules. Both figures illustrate that ATLAS is depending on a small core developer team. To reduce this dependency ATLAS needs to increase the number of skilled and dedicated developers.

The heterogeneity of developers means that many parts of the software are developed by people without formal education in computing, a problem common in scientific computing [50], [51]. This is in part due to the fact that the institutes that are part of ATLAS have an obligation to participate in activities required for operation. Estimates are that around 600 people are required for the operation of ATLAS, which is $1/6^{th}$ of the entire workforce of the 3600 members of ATLAS [52]. These activities include monitoring of the detector activities, hardware and software calibration and software development. Almost all ATLAS institutes are from the field of physics and software development is not a fundamental part of most physics study programs. This conflict of interest leaves results short of what could be achieved with a comparable workforce of dedicated developers. For this reason, the software has inefficiencies, contains untested and unreviewed code, memory leaks and unsafe pointer handling and is not well documented. This situation is allowed by the

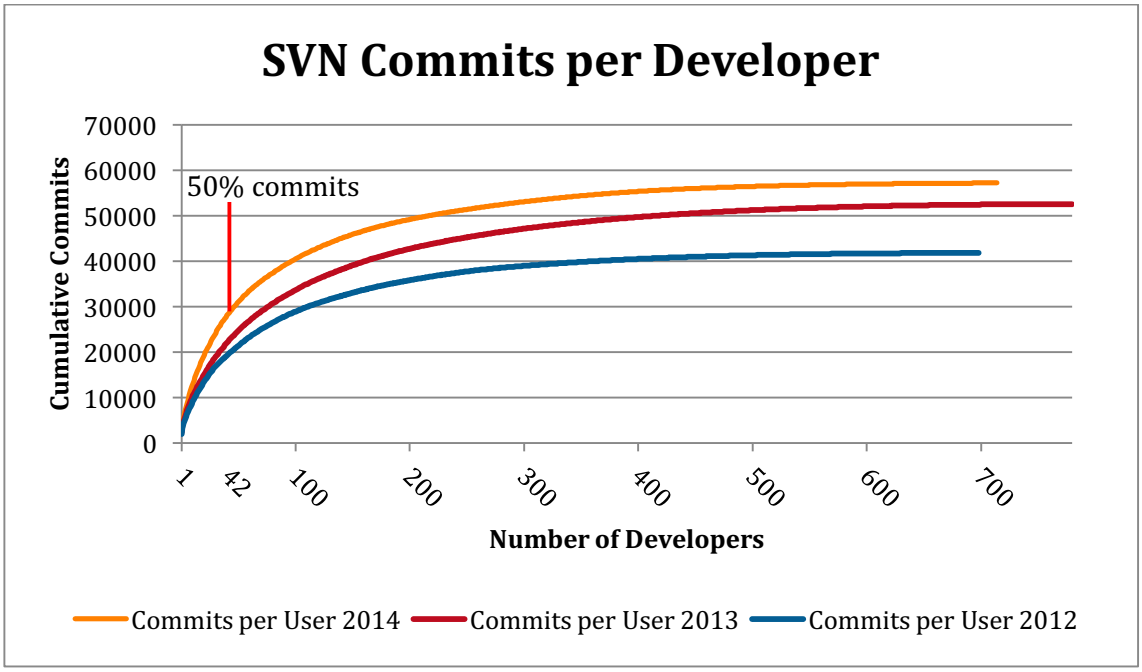


Figure 24: Load distribution of software development in ATLAS measured by number of svn commits per person. In 2014, 42 developers were contributing 50% of the changes. Data extracted from svn.

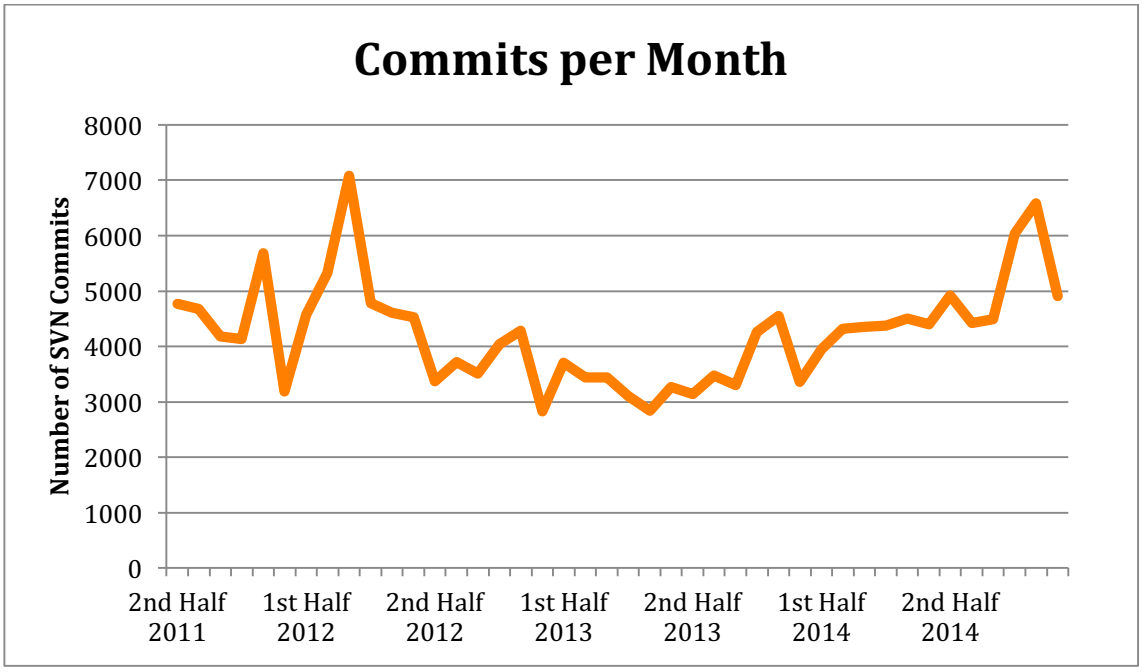


Figure 25: Number of new software package versions committed to ATLAS SVN each month. Data extracted from svn.

insufficient code quality assurance mechanisms in ATLAS, which are unsuitable for a project with thousands of code changes every month (see Figure 25). The introduction of Jira [53] as issue tracking and management system lead to clearer responsibilities, such that many of the problems found with the Coverity static code checker [54] in place were tracked and fixed. While Coverity led to the discovery and elimination of many problems, it does not replace tests, which can assure that the behaviour is not only not undefined, but is also the desired behaviour. Currently, the only automated tests are integrated into the ATLAS nightly build system.

3.4.1 Building and Testing in ATLAS

This build system knows final releases, development (dev) nightlies, development-validation (devval) nightlies and migration releases. Final releases represent a fully consistent state of the software that should be bug free. Development nightlies should also be free of major bugs, but represent ongoing development. Within devval nightlies developers can test if their changes work well with a full build before they are integrated into dev. Migration releases are used to implement changes that will break the release for a prolonged period, such that all changes can be integrated into a dev or devval release once the migration is complete. Release numbers follow the scheme W.X.Y.Z. Final releases are defined by numbers W for major changes and X for minor changes, while Y and Z are used for bug fixes and performance improvements. The dev and devval nightlies have a one-week cycle, being rebuilt every night overwriting the release that is one week old. A change is always first included in the devval release, from where it can go to dev if the developer deems it stable. These multiple levels of fully built releases prevent non-working code from going into production but require a lot of resources, as a full build on a dedicated high-end machine takes about 9 hours. This also means, an error will make the whole release unusable by the whole developer base until the error is removed and rebuilt, which is at the earliest the following day.

A release may be validated with respect to its physics results. Such a release can be used in production, and is barred from changes that affect an algorithm's result except for critical bugfixes ("frozen") to maintain reproducibility. A release is usually only validated for either simulation or reconstruction but not for both, because time schedules for either purpose are different. For this reason, separate versions are validated separately and development also continues separately due to the policy of frozen releases.

For performance monitoring, a Runtime Tester (RTT) system has been set up. This system runs a series of predefined jobs after a new nightly has been built, scans the log for crashes, warnings and errors and allows inspecting the logs and results. This is a minimal test if a whole domain runs without crashing, but checking the results for sanity remains manual and is done at the developers' discretion.

Performance is monitored in a similar way, comparing performance between two releases or nightlies. Before allowing a release into production, a physics validation is performed, which is the process of manually verifying the sanity of results of the whole software chain. In modules responsible for one or more percent of the used CPU time, inefficiencies are spotted faster because usually only experienced developers are assigned to work on such code. The vast number of modules requiring only a few milliseconds per event remains a problem, as they are too small to attract attention and remain unchanged even though they might have become completely unnecessary. A study has shown hundreds of tables in a job's output that are created but never filled. Another problem is that the reconstruction software is divided in different domains, reflecting the organizational structure of ATLAS. In some cases, responsibilities are not clear where areas of multiple groups are touched, slowing down development.

3.4.2 EDM Design Considerations

Considering the development of the C++ language in the past years, many well-meant decisions affecting large parts of the software should be revised. Generally, a strictly object oriented design was enforced, leading to very deep inheritance chains. Members of EDM classes were created lazily, although they are always accessed. Lazy initialization means allocation and initialization takes place at time of access instead of at time of object creation. This can save computing time and memory in case the members are not always accessed. Dynamic memory allocation is expensive, such that time is wasted if dynamic memory is allocated for multiple small objects rather than for all of them at once. Other parts of the EDM were identified through dynamic casts, as the associated cost was

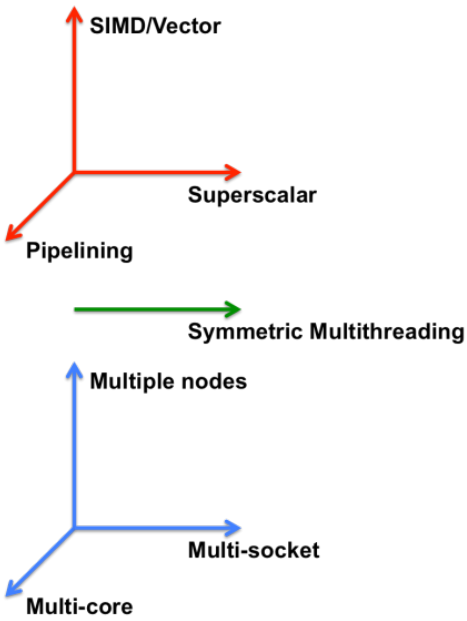


Figure 26: Seven performance dimensions as taken from [55]. The dimensions are orthogonal to one another with the exception of symmetric multithreading.

expected to diminish to the cost of an integer comparison in future C++ versions, which never happened. Some design choices such as the originally strictly object oriented structure intended to allow easy maintenance were not taken for high performance. This illustrates that the software was not built for high performance but to perform the needed work. The consequences of these decisions continue to affect the performance to this day. At the time, staying within the permissible resource limits was not problematic as computing resources were sufficient for the problem size. This changes drastically for Run 2 because of the greater complexity of events and their larger numbers on the one hand, but also with the change in hardware development explained in the next section.

3.5 Hardware evolution

During the Athena design and development phase, multicore computers were not yet widely available and the developments towards having more and more cores were not foreseen [40]. Subsequently, this and other hardware developments have become available at the computing sites over time as hardware is exchanged on average every four years. This section details the development of hardware and its support in the production code after Run 1.

3.5.1 Parallel Resources on Modern Computing Hardware

In order to optimize, we have to be aware of the available resources and how they can be used. Notably, many different forms of parallelism have been introduced. Optimization for these parallel resources is not always possible for several reasons. One problem that is particularly difficult to address is that algorithms may have precedence constraints requiring a sequential order of execution. This means if the order of operations is not interchangeable without changing the result, they also cannot be executed in parallel. Enabling parallelism may require structural changes in the data or even different algorithms. Concentrating on CPU capabilities for the hardware side, I will use the 7 dimensions of parallelism introduced in [55], see Figure 26. The dimensions are orthogonal in the sense that their impact on performance is multiplicative. Most implementation details are given on the example of a Haswell architecture CPU.

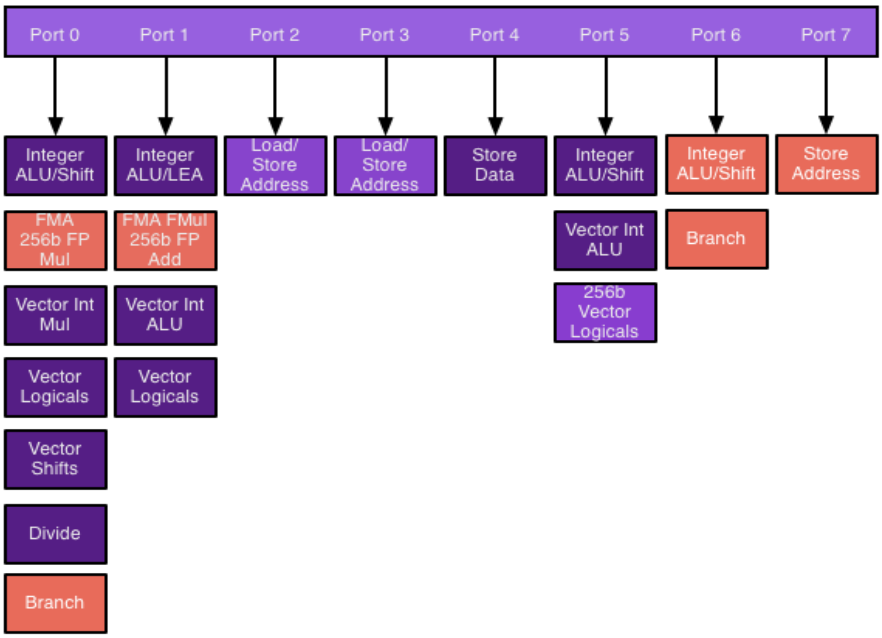


Figure 27: Ports on a Haswell CPU core. The ports can be used simultaneously, even by instructions from different threads.

Hardware parallelism exists in the following concepts:

1. **Pipelining** allows retiring one micro operation (µOP) per clock cycle, even for operations that take more than one clock cycle. This works because an execution unit can already start computation on another µOP one clock cycle after starting to process a µOP, even if they take more than one clock cycle to complete. Without pipelining, a port would be blocked from accepting more µOPs until the execution of one µOP is fully finished. With pipelining, the next µOP can already be fed into the pipeline of a port one clock cycle after the execution of the previously inserted µOP began. Pipelining is usable for most types of µOP with the exception for division. A µOP is not equivalent to an instruction. Instructions denote the operations that a CPU can decode, but cannot execute directly. Instead, the instructions have to be decoded to (in some cases multiple) µOPs which can then be fed into the respective pipeline of a port on a CPU core. A µOP can e.g. be loading or storing data or elementary arithmetic operations on integer or floating-point numbers.

2. **Superscalar execution** refers to the parallel usage of multiple ports on a single core. Parallel ports allow multiple µOPs to be dispatched to different ports in parallel. This requires support of out of order execution by the CPU where µOP dependencies are automatically detected such that the program logic doesn't change. Modern Intel processors have multiple parallel ports that can execute different µOPs, although not all ports can execute all types of µOP. A Haswell architecture CPU core has 8 different ports allowing different operations [56], see Figure 27.

3. Wide **SIMD** (single instruction multiple data) support allows execution of one operation on several values. If the values are in adjacent memory locations, execution takes no longer than computing a single value. A Haswell CPU core has ports that can process up to 256bit of data at once, corresponding to 8 single precision values or 4 double precision values, such that a factor of 4 or 8 in performance may be lost if SIMD instructions are not used.

4. **Symmetric Multithreading** (SMT) allows multiple threads to simultaneously execute instructions on the same CPU core. If not all ports are fully used by one thread, one or several threads can dispatch instructions to unused ports. SMT does not increase the theoretical maximum throughput but adds to parallelism by allowing utilization of resources by multiple processes/threads, which would also have been fully available to a single thread. A Haswell CPU supports issuing instructions of two threads simultaneously per physical CPU core.

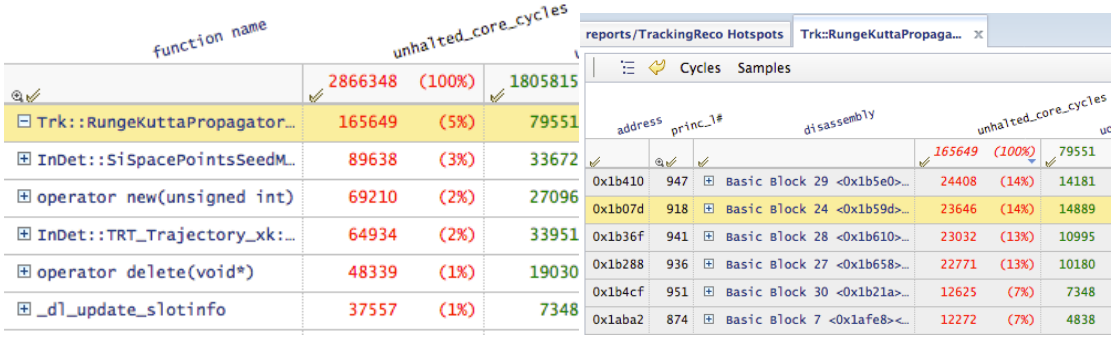
Of these core-internal dimensions, directly influencing dimensions other than SIMD or SMT (and thereby also, to a point, superscalar execution) is non-trivial with a high-level programming language. µOP scheduler behavior is only known empirically as Intel does not publish implementation details. Enforcing SMT works by pinning threads to a core or simply creating threads equal to the number of logical cores. SIMD parallelism can only be exploited by writing SIMD instructions or ”SIMD-friendly” code, for which the compiler is able to generate SIMD instructions. Both often require deep structural changes to the code to have the data for SIMD instructions in adjacent memory locations and to restructure a problem such that many operations of the same type are executed without intermittent different operations. Compilers automatically recognize only trivial cases of SIMD parallelizability and writing SIMD commands (intrinsics) is error prone and deprecates with new CPU generations.

None of these dimensions of parallelism have played a role in the design of the ATLAS reconstruction software. The remaining dimensions of parallelism are based on multiple cores and allow embarrassingly parallel execution although other models may be more beneficial. Embarrassingly parallelism in the ATLAS reconstruction is achieved by running multiple Athena instances processing different events.
Embarrassingly parallel exploitable concepts are:

5. **Multicore processors**. The number of transistors still doubles every two years, following Moore’s law. This led to an increase in the number of processing cores on a single die. The trend is going to further increase the number of cores on a die by making each core less complex and therefore smaller. This way, the power to flops ratio tends to be better than in highly complex CPUs with only few cores. The multiple cores on a die usually share the level 3 cache, such that on the one hand multiple threads can profit from cache locality but on the other hand have less cache per core available. With the Many Integrated Core (MIC) architecture, Intel has released a co-processor with over 60 fully x86 compliant cores. The cores are simpler and have a lower frequency clock than high end Intel Xeon CPUs. Each core allows four SMT threads on each core as compared to two threads on a Haswell CPU architecture. Nonetheless, performance per watt of such a processor is very competitive compared to classical server CPU architectures with fewer heavy-duty cores but requires that the majority of cores can be kept busy.

6. **Multi-Socket**. Some motherboards have multiple so-called sockets which can host a CPU each. All CPUs on a single motherboard have access to all memory installed on this mainboard, but memory is only directly connected to one socket. Access to memory connected to another socket has to go through the connected CPU leading to considerable delays. This is called non-uniform memory access (NUMA).

7. Separate processing **nodes**, i.e. completely independent machines, naturally have all resources replicated so there is no interference between different nodes unless they access the same resources over network. Any communication between nodes has to go via network, which is the slowest connection of all other interconnected parallel resources.



function name	unhalted_core_cycles		
	2866348	(100%)	1805815
Trk::RungeKuttaPropagator…	165649	(5%)	79551
InDet::SiSpacePointsSeedM…	89638	(3%)	33672
operator new(unsigned int)	69210	(2%)	27096
InDet::TRT_Trajectory_xk:…	64934	(2%)	33951
operator delete(void*)	48339	(1%)	19030
_dl_update_slotinfo	37557	(1%)	7348

reports/TrackingReco Hotspots | Trk::RungeKuttaPropaga…

Cycles Samples

address	princ…l#	disassembly	165649	(100%)	79551
0x1b410	947	Basic Block 29 <0x1b5e0>…	24408	(14%)	14181
0x1b07d	918	Basic Block 24 <0x1b59d>…	23646	(14%)	14889
0x1b36f	941	Basic Block 28 <0x1b610>…	23032	(13%)	10995
0x1b288	936	Basic Block 27 <0x1b658>…	22771	(13%)	10180
0x1b4cf	951	Basic Block 30 <0x1b21a>…	12625	(7%)	7348
0x1aba2	874	Basic Block 7 <0x1afe8><…	12272	(7%)	4838

Figure 28: Hottest functions and hottest blocks within the hottest function. Each of the blocks accounts for less than 1% of the total runtime. Data taken with GOoDA profiler.

New Intel and some other vendor's developments are usually available to CERN shortly after or even before their release for testing purposes due to the close collaboration with Intel and other hardware vendors mainly through the CERN openlab but as well through other channels. While reconstruction in its current form cannot exploit some of the new features, it is used for some applications such as random number generation, a significant cost factor in simulations, and projects to make use of these architectures for reconstruction are ongoing.

3.5.2 Parallelization on modern architectures

Parallelization to make use of a CPU's SIMD capabilities requires careful code adjustments by experts and as such only pays off for hot spots in the code, which are hard to find in the ATLAS reconstruction software, see Figure 28. Even when finding a hotspot, vectorization may not be possible for a particular problem at all.

Multi core, multi socket and multi node parallelism can all be exploited by creating more instances of a program. For a more memory-efficient exploitation of the multiple cores per machine, Run 2 production will run on a multi-process version of Athena, AthenaMP. The advantage of forking processes over creating multiple independent instances is that Linux offers a copy-on-write mechanism such that all memory that remains unchanged after forking is shared between the processes. For multi socket, one CPU has only access to one set of memory banks, such that for access to other memory the data has to pass through the respective CPU. Leaving thread management to the OS, this may lead to significant performance penalties due to NUMA effects. This is why AthenaMP assigns processes to a core, which improves throughput by 20% [39]. The current developments of increasing number of cores per CPU with each generation lead to a smaller amount of memory available per core, which cannot economically be solved by acquiring more memory. Multi core and multi socket could also be exploited by running multiple threads of the same process, although communication between threads on multiple sockets has an increased overhead because separate CPUs do not share any fast memory, unlike cores on a single CPU which share on-chip memory called the level 3 cache.

All memory is shared between multiple threads of the same process by default unless specified otherwise. In contrast, between multiple processes only unmodified memory pages are shared. Modern operating systems are able to intelligently share read-only memory between multiple processes of the same process tree. Making use of this functionality required only changes to the Athena framework to be able to fork, leaving the algorithms as is, leading to the AthenaMP framework. Threading, with the approach of declaring everything that is not explicitly unshared as common memory, allows sharing much more memory, but requires to implement safeguards such that memory is not shared unsafely. This is why threading comes at a much higher development cost than multi-processing, especially if introduced in retrospect to a single-threaded design such as

the ATLAS code and the Athena framework, which in some cases enforces use of non-const objects.

3.5.3 Multi-core CPU versus GPU processing

In recent years, GPUs have become popular for problems outside the video-rendering domain. GPUs are highly optimized for massive data parallelism. To exploit the GPU parallelism, the operations have to be of the same type, similar to SIMD operations but with wider registers and more advanced mechanisms to automatically maximize the number of parallel operations. This makes GPUs applicable for problems that have many floating-point operations of the same type on data in adjacent memory locations. To run a program, the program and data must be uploaded to the GPU's own memory. After execution, it must be copied back to the host machine. To compensate for this overhead, the problem size must be large enough. In return, modern GPUs are usually much faster and more energy efficient in performing floating-point operations than CPUs. On the downside, GPUs usually have a very slow single threaded performance such that only heavily parallelizable parts of the ATLAS software could be efficiently run on GPUs. Already some prototypes for tracking exist [57], but no in-depth studies about the competitiveness of the approaches compared with the heavily optimized single threaded CPU tracking or with multi-core CPU based approaches exist.

3.6 Software evolution

Since the start of development, the available software evolved and continues to change at a quick rate. Compilers and languages change and libraries come and go. ATLAS is not always quick to pick up new features for different reasons, sometimes because it requires the CERN infrastructure to be changed, requiring the agreement of all concerned collaborations. Other changes may imply huge implementation effort. Since the start of development many tools for performance analysis have been released for easier and more detailed profiling, which helps ATLAS to become aware of benefits of new developments and deficiencies of the current setup.

3.6.1 Profiling evolution

Before and during Run 1, Valgrind [58] was the most used profiling tool. It has a number of plugins allowing profiling different aspects, such as heap analysis, wall time analysis and others. Valgrind does the profiling by introducing an additional layer between the profiled software and the hardware resources. This allows very accurate measurements of the used resources, but the additional software layer slows down execution by, depending on the type of measurement, a factor of 10 to 100. This severely limits profiling as it costs a lot of resources and takes too long for exhaustive testing. In recent years sampling tools have emerged (e.g. gperftools [59]), which measure wall time spent by sampling the stack trace every set amount of time, typically every 10ms. This allows pinpointing costly functions and to a degree even a line of code. The line cannot always be accurately determined as code may be reordered and restructured depending on the optimization level. Due to the sampling nature, functions requiring a very low amount of time will not show up in the profile, which is usually not a problem as the heaviest functions are usually the most interesting. It can also show if calls from one code location take more time than from another location. These tools add only about 5% to the runtime of the profiled software. Not purely software related, Intel introduced hardware performance counters inside their CPUs, which allow reading low-level performance measurements such as cache misses and instructions per cycle, leading to the creation of tools to create profiles based on these counters. Using the counters does not slow down the profiled software and they are in

theory 100% accurate. In practice, some implementations on some CPU generations are faulty and do not accurately count every event, or have to be sampled because of the otherwise too high data rates. The performance counters will always measure performance of all programs on the same core. For both sampling and performance counter based profilers the machine used should be dedicated for the tests and not run any other programs so the measurements are not affected. In 2009 the Performance Monitoring Board [60] was introduced at ATLAS, which monitors the changes in performance and allows an overview of the amount of time spent in the different domains. A set of tests is run on each nightly build, scanning for changes. If an unacceptable increase in resource usage is detected, only a small number of packages that were changed have to be checked.

Concurrently to the ongoing development of Athena as it is used now, frameworks for future requirements are in development. One of them is the multithreaded version of Athena, AthenaMT, which allows the parallel execution of independent algorithms in an event and is expected to allow reducing the memory footprint per busy core in future multi core architectures, see also Section 4.5.

3.6.2 Parallel programming trends

Of the many innovations in recent years, parallel technologies have seen the most development and attention. On the software side this is especially reflected by new languages and language features:

- CUDA [61] is Nvidia's proprietary language to write programs for Nvidia GPUs, released in 2007.
- In 2009 Apple released OpenCL [62], a multi-platform language allowing writing programs that execute on CPUs, GPUs and other hardware.
- The Cilk language [63] has existed since 1994 already but was only popular in the super computer domain. It was made popular when Intel bought it and released a C++ library TBB [64] for easier threading based on Cilk in 2006.
- C++11, released in 2011, has finally introduced a formal model to support concurrent operations.
- OpenMP as one of the older standards, defined by a group of hardware vendors in 1997 but is only supported by GCC since version 4.2 released 2007.

All of these languages have seen much improvement since their introduction and parallel programming is wide spread. Much effort has been put into hiding away the low-level complexities of parallel programming.

3.6.3 Library evolution

Over the years, many dedicated libraries for highly specific problems were created and reached the end of their support life. Well written and maintained libraries dedicated to delivering high performance can make use of new hardware features without requiring work on the side of the library's user. Instead, the user has to rely on the continued maintenance by its developers. This allows outsourcing low-level code development for basic operations to specialists while development at ATLAS can concentrate on ATLAS specific code. ATLAS, LHCb and several other experiments have most (but not all) of the libraries their software uses managed in the LCG software stack, a collection of hundreds of libraries. The LCG software stack is coordinated by the EP-SFT group of CERN, which keeps the libraries up to date and provides compiled versions of the stack for different combinations of operating systems and compilers. This relieves the experiments of keeping their libraries up do date themselves and deduplicates the work which would otherwise be performed by multiple experiments individually.

3.6.4 Compiler optimizations

Compilers have seen many performance optimizations through static and even dynamic code analysis. Compilers used in ATLAS are regularly replaced by newer versions when performance improvements can be shown. Out of order execution, loop unrolling and auto vectorization are only some of the technologies that have been introduced and in part only allow the exploitation of newer hardware technologies, for example until the end of Run 1 GCC4.3 was used which does not generate vector instructions. During LS1, the compiler was changed multiple times for newer versions, currently GCC 4.9 is officially used and GCC 6 is being tested. Clang is also considered in ATLAS, mainly because of the improved error output which allows a developer to pinpoint the problem more quickly. Intel's icc is provided for free to all CERN experiments, but cannot be used freely by external users, which prevents ATLAS from using it at all.

3.6.5 Continuous development on Athena

During Run 1 the ATLAS software was mostly adjusted to perform well from a physics point of view. Unforeseen properties of the then-new data had to be reflected in the software. A main goal for these changes was therefore the improvement of physics performance, to gain a more accurate reconstruction. Computing performance was of secondary importance and only improved when reconstruction performance did not seem sufficient to process the data on Tier 0. Every year the Tier 0 software was replaced by a newer version during Run 1, each with a better physics performance and faster processing.

3.7 Conclusion

The ATLAS collaboration has written millions of lines of code to solve the reconstruction problem responsible for a large fraction of the computing cost. The problem size has been increasing over the years and will continue to grow. Since the start of the ATLAS collaboration, both the available hardware and software have seen huge developments and shifts of paradigms. Computing hardware has been continuously replaced with newer hardware at the computing centers, such that most new hardware developments are available to ATLAS. New software technologies were incorporated into the growing software where possible but some design decisions have prevented profiting from others. Notably, non-embarrassing parallelism is painfully absent in the ATLAS software, in part because the ATLAS reconstruction has no palpable hot spots.

A codebase the size of the ATLAS reconstruction would normally be managed following a strict workflow in professional software firms. Instead, ATLAS has a resource intensive and slow multi-layer build system designed to catch errors before they go into a production release. This is owed to the heterogeneous developer base consisting mostly of physicists with little formal education in software development.

New profilers show shortfalls of the software in unprecedented detail. Previously unrecognized deficiencies can be observed and fixed. Recent compilers are capable of optimizing more complex code, making the adjustments necessary for automated optimization more achievable. Much potential and hope is particularly on the so far unused parallel capabilities of the hardware which compilers also can help. The development process in ATLAS has severe shortfalls, which can partially be addressed by introducing a more professional structure.

The common understanding is that the ATLAS reconstruction problem size will continue to grow. The arising challenges can only be addressed within a medium-term timescale because it includes changing the grown design of the software.

4 Computational Performance Analysis

Before any software can be optimized in a meaningful way, it has to be carefully analysed for its resource usage. Software performance depends on many parameters such as input data, hardware used, the OS and other surrounding software and resource usage of other programs running concurrently. Well-defined test cases set artificial conditions that are easy to reproduce and mimic some of the original conditions such that similar software behaviour can be expected during production.

The test cases show an immense increase of a factor of 7 in time spent per event during Run 2 in comparison to events recorded during Run 1, plus the number of recorded events increasing by a factor of 2.5 from 400Hz to 1000Hz. This poses a huge challenge to the developers to process the load during Run 2. The expected increased available computing performance reduced some of the pressure on the developers, nonetheless increasing the computational performance defined the goal for the LS1 software campaign: All domains were advised to speed up their software by a factor of 3 to be able to deal with the Run 2 load without creating queues.

In order to quantify potential gains, I measured different aspects of the reconstruction software using different profilers for different aspects. A structural analysis shows a detailed picture of the data flow and combines it with timing measurements for a parallelization study. For each analysis, a short conclusion gives the potential gains versus the expected cost to implement them.

4.1 Decomposed Reconstruction Performance

In order to quantify potential gains or simply find bottlenecks in the current ATLAS reconstruction software, I performed a thorough performance and code analysis. The most important measure for the performance of the software is the throughput, which quantifies the amount of work completed per wall time on the available resources. To determine the throughput, I measured the wall time per event for a single process job with one thread. Some parts of the software are memory-bound but concurrent access to these resources by multiple instances did not significantly slow down execution in tests with multicore machines, such that measuring the throughput this way remains valid. Convertibility between throughput and performance measurements is important. To be able to

	Track Finding	Ambiguity Solver	TRT Segment Finder	Total Reconstruction
Autovectorization ON	+1.7%	+3%	+1.4%	+2%

Table 2: Wall time change with autovectorization. The tested algorithms were slightly slower after turning autovectorization on, but the difference is so small that it may be due to measurement inaccuracy. Although few fixed size loops with low trip count were vectorized no improvement was achieved and autovectorization was not turned on for any production release during Run 1. Tested with release 17.2.9.7 used for production in the last year of Run 1.

scale my measurements on the heterogeneous hardware available to ATLAS, I assume that improvements seen on one CPU generation have a comparable relative impact on the other CPU generations used by ATLAS. This assumption is justified since the vast majority of CPUs are x86 architecture mostly from Intel and within three processor generations. This would not be justified if ATLAS made heavy use of SIMD, which changes quickly between processor generations and could therefore lead to large differences in performance but ATLAS doesn't make use of SIMD at all. Tests showed turning autovectorization "on" or "off" has no positive impact. Some algorithms even showed slightly worse performance with autovectorization but the difference is within the measuring inaccuracy, see Table 1. Running verbose autovectorization showed that only some fixed size loops for initialization were successfully vectorized, presumably because most loops where work is performed iterate over C-style arrays with unknown location in memory. This section presents my measurements of the wall-time of individual steps and my analyses of the state of the ATLAS reconstruction software before LS1.

4.1.1 A general test case for performance observations

To base the performance analysis on realistic scenarios, I created test cases reflecting the setup in production of both before and after LS1. The test cases take into account the environment in which the software runs, the state of the software analysed and data. Tests reflecting the requirements before LS1 run a full Inner Detector reconstruction with 20 pileup interactions, Monte Carlo simulated samples with a center of mass energy of $\sqrt{s} = 8TeV$, according to the conditions at the end of Run 1. Pileup interactions per event continuously increased during Run 1 up to a peak of 37, but can differ by a factor of two to three over the course of a single LHC filling, and average pileup in 2012 was 20.7 interactions per event. This test event type will henceforth be referred to as "Run 1 event". To simulate data as expected to be delivered by the LHC in Run 2, the ID reconstruction runs with events with 40 pileup interactions, Monte Carlo simulated samples with a center of mass energy of $\sqrt{s} = 14$ TeV. Energies during Run 2 are now set at 13 TeV, but the difference in event topology is small. The event type used for the test case produces a top-antitop quark pair, which decay to a large number of particles. I omitted one reconstruction step, the muon reconstruction, which does not lead to significant differences in runtime due to the low number of muons per event in comparison with the number of Inner Detector tracks, but avoids complications with the performance measurements which led to crashes. All timing measurements run over 150 events to get the average time per event. The time for the first event is not taken into consideration because many modules are initialized during the first event. The first event does not significantly influence the total runtime because production reconstruction jobs typically run over 5000 events.

4.1.2 State of ATLAS reconstruction software before LS1

At the end of Run 1 in 2012, the production release for reconstruction was 17.2.7.9. Development of the software in use is constrained during operation, as an algorithm's output may not change so results can be reproduced. This guarantees a consistent dataset for the entire data-taking period of one year. Changing behaviour or performance throughout the operation phase would make a reprocessing campaign necessary, re-reconstructing all data taken up to this point with the changed software. This constraint limits changing algorithms where floating point operations are an integral point, as even numerical differences cannot be tolerated and prohibits changing the flow of execution completely. The term used internally is that a release is "Frozen Tier 0" to state that no changes except for bugfixes are allowed. Still, development was not halted but continued in parallel releases. These releases would not go into production before the next shut-down, which is usually at the end of each year. The exception to this rule is if computing resources are available to reprocess all data taken up to this point with the new release. The resulting missing promise of immediate return of investment and ATLAS' clear focus on data taking during the first years of operation left development going slow in these periods. Nonetheless, the production release delivered the required physics performance and kept within the permissible limits regarding resource requirements.

The two most prominent of these resources are CPU and memory because they often constitute a bottleneck. Jobs submitted to the grid, which distributes tasks to grid sites around the world, were allowed to take a maximum of 2GB of memory, which is close to what some simulation jobs required, but well below the between 2GB and 4GB of memory required by a reconstruction job in release 17.2.7.9 using my test cases. This restricts reconstruction to Tier 0 and some Tier 1 sites. Memory leaks are present but small enough not to pose a problem due to memory usage well below the limit per job configured by the grid sites. Grid computing resources are shared between all experiments, and each experiment has a share of the CPU time it can use. ATLAS has a history of requiring a large share of these resources, which is granted due to higher complexities in the reconstruction process reflecting a more complex detector architecture and model.

Event reconstruction in different parts of the detectors requires different technologies and different approaches and to a degree can be done independent of one another. Thus, it is natural that software groups are organized in different domains, which, however, are all developing for the same framework Athena. The impact of each domain on the runtime can be seen in Figure 29. As the graphic shows, the total runtime increases by a factor of more than 7 from Run 1 to Run 2 events. More than 50% of the time is spent in the Inner Detector domain. The dominance of the ID domain increases to two thirds for events with 40 instead of 20 pileup interactions. The increasing share can be explained by the high complexity algorithms required for ID track reconstruction. The generally high share is only in part explained by the high complexity. Additionally, only the ID and the calorimeters have to deal with a very high number of tracks and particles. The calorimeters surrounding the ID are designed to stop most particles such that only few particles reach the other detector elements, i.e. the Muon Spectrometer in the outer detector region. The calorimeter algorithms, despite dealing with high number of tracks, are largely unaffected by event complexity, as complexity is linearly dependent on the number of calorimeter cells, which does not change. The number of higher energy particles scales sub-linearly with pileup, leaving muon reconstruction largely unaffected, while the number of low energy particles increases linearly. Particles measured in the ID are reconstructed above a certain energy threshold, but even below the threshold they negatively affect runtime because their measurements are indistinguishable from higher energy particles until after reconstruction. Note that the linear scaling of the number of low energy particles does not mean linear increase in runtime, but instead the higher occupancy leads to more combinations leading to a much longer runtime. This is why pileup is the most important factor for runtime.

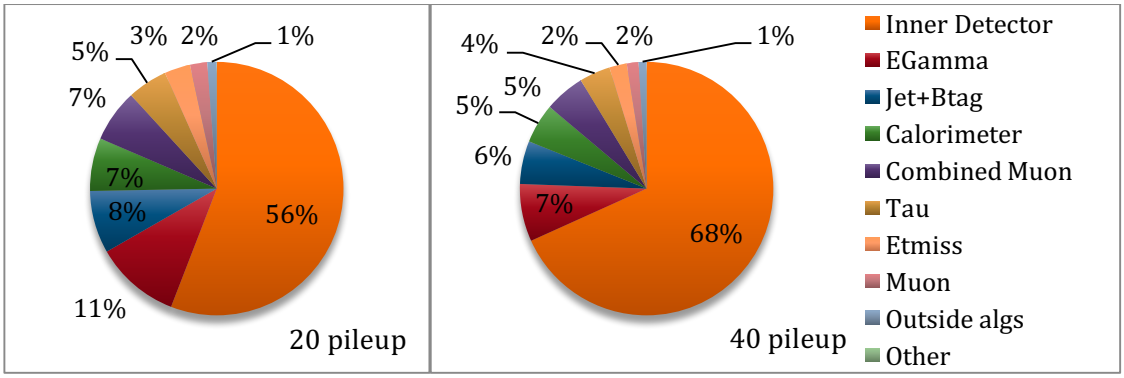


Figure 29: Domain breakdown for release 17.2.7.9. Measured for events with 20 (left) and 40 pileup interactions (right). Average runtime per event is 11,677ms for 20 and 84,389ms for 40 pileup interactions on a Nehalem CPU L5520 with 2.26GHz and 24GB memory running Scientific Linux 6.

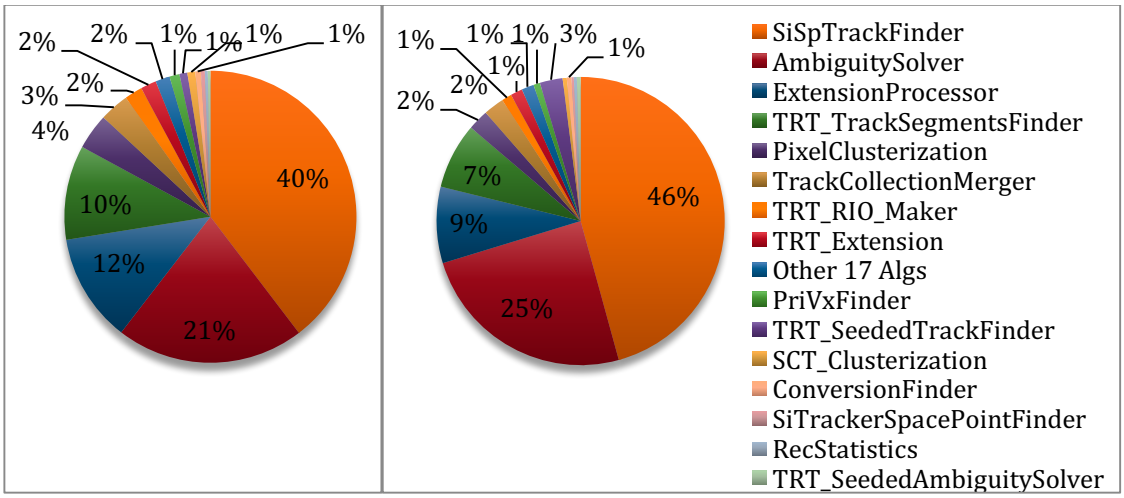


Figure 30: Breakdown of Inner Detector domain runtime for ttbar signal events with 20 (left) and 40 pileup interactions (right), the largest two algorithms, Silicon Space Point Track Finder and Ambiguity Solver increase dominance with higher pileup. The test setup is the same as in Figure 29.

An event is recorded and reconstructed if the configured multi-level trigger system accepted the event. This is usually due to specific event topologies. This was the case for only 0.002% of the events during Run 1. That means that with very high likelihood only one of the collisions per bunch crossing leads to a triggered event, with usually a larger fraction of high-energy particles in the detector while from the other collisions mostly low energy particles originate.

An average of 40 interactions per event is expected for Run 2, so the software is analysed for how it behaves for this type of event compared to the average Run 1 event. These two event types will in the following be referred to as Run 1 event and Run 2 event respectively. The breakdown chart for the ID is shown in Figure 30. Two algorithms dominate the ID reconstruction runtime for both Run 1 events and for Run 2 events. Both algorithms deal with combinatorics. The SiSpTrackFinder, short for Silicon Space Point Track Finder, constitutes the combinatorial track finder as explained in Section 2.7. The AmbiguitySolver resolves multiple tracks that share the same measurements to only reconstruct the most likely tracks and performs the final track fit. Both algorithms use a number of tool chains and services where much of the actual work is done. The time of the SiSpTrackFinder contains the seed finding and the combinatorial track finding. The speed of the reconstruction is very much dependent on the quality of the results from the seed finding

because the seeds are used in the subsequent, most expensive steps. My measurements show that seed finding is responsible for 25% of the time required for pattern recognition for a Run 2 event with ttbar signal decay, which corresponds to 8% of the total runtime in the ID according to the measurements presented in Figure 30 in Subsection 4.1.2. This means the subsequent track finding accounts around for 23% of total ID reconstruction runtime. While during data formation there is no obvious option to reduce the load on the subsequent algorithms as it is a quite straight forward step, the seed finding creates 60 seeds for each track that is found. Each of these seeds is tentatively reconstructed as a track candidate, which in total makes up the rest of the time spent in the pattern recognition. Reducing the number of the seeds which do not lead to a track, so called fake seeds, can therefore yield large gains for reconstruction, and is the reason why much effort is spent to improve the seed finding. Improvements have to be carefully tested, as all seeds that are disregarded although they belong to a particle are lost if no other seed for this particle is found. Therefore, changes are not allowed to reduce the number of seeds belonging to particles.

Analyses using gperftools and Valgrind reveal the SiSpTrackFinder spends with around 16% the largest amount of its time in Runge-Kutta propagation [65], half of which is spent in other tools further down the call chain. The Runge-Kutta propagation is needed to solve the transport of a particle trajectory through the inhomogeneous magnetic field. It has been extensively optimized, so further improvements to the algorithm does not promise large gains. Even if large improvements on this implementation were possible, reducing the time spent here to zero would only gain 2% on the total reconstruction runtime. As most algorithms and tools, it relies on other tools and is depending on input from other algorithms.

4.1.3 Source-code efficiency and hotspots

To tackle the shortcomings of computational performance of a software, it is necessary to understand why it does not perform as desired. Software is a complex product depending on many other factors, there are many properties with which the efficiency can be observed and analysed. Understanding how CPU time is spent can show what is the limiting factor of the executed program. Information why a CPU is not reaching its theoretical maximal throughput may give programmers enough information to improve the software. Executing wrong instructions because of branch misprediction are an example of an inefficiency. The information how many branches were mispredicted and many other of such low level measurements can be collected in modern CPUs in performance monitoring units (PMUs), which can be read using perf [66], a program available in modern Linux kernels. As the raw data is difficult to interpret, analysis tools such as Intel's VTune [67] or the now discontinued Google's Generic Optimization Data Analyzer (GOoDA) [68] allow in-detail view how CPU resources are spent and can attribute these effects to code lines. Attribution to code lines is done sampling the stack trace, therefore the attribution is slightly inaccurate but accurate enough to look for hot spots and inefficiencies. For the example of branch mispredictions, restructuring conditional branches or reducing their number can enable a CPU to predict more accurately which instructions to load. A look at total cycles spent in track reconstruction with GOoDA shows that 58% of all cycles are stall cycles, see Figure 31. At the same time, the proportion of instructions per cycle retired (IPC) is almost one, which means during the 42% unstalled cycles on average 2.38 instructions were retired per cycle, with the theoretical maximum of modern CPU cores being 4 IPC [69].

An IPC of 1 is what Intel considers to be a good performance as a rule of thumb for enterprise applications [70]. For the hot spots, a detailed profile is generated, showing how CPU cycles are spent line by line and side by side with the corresponding assembly.

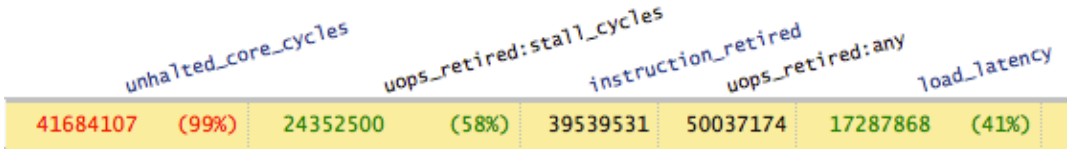


Figure 31: GOoDA analysis of track reconstruction. Although 58% of all CPU cycles are stall cycles, IPC (instructions retired per cycle) is almost one.

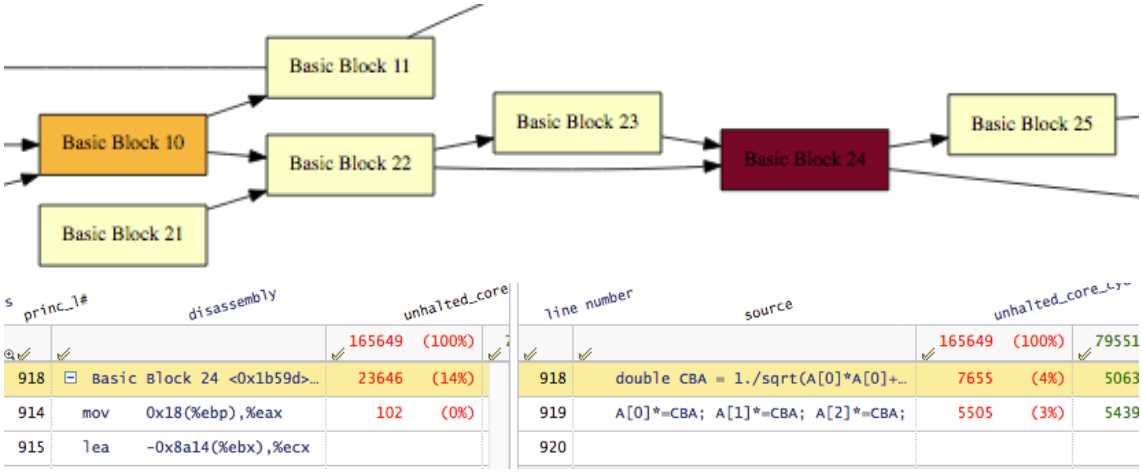


Figure 32: The upper figure shows an excerpt of the control flow of the Runge Kutta propagator with basic blocks color coded by time spent in each block. The lower figure depicts how GOoDA shows assembly and source code side by side.

Both the amount of time spent and how the time is spent in a library, function or line of code are shown at a glance. For each function, a graph of basic assembly blocks shows execution paths and cycles spent per block. In [68], Calafiura et al. use GOoDA to pinpoint lost efficiency and deduce possible improvements in the ATLAS reconstruction software. The Kalman filter and the magnetic field in the ATLAS reconstruction are analyzed more closely in [71], each contributing about 3% to runtime in the selected setting. They have been selected because they spend most of their time very differently; while the magnetic field is limited by bandwidth because of non-consecutive memory access, the Kalman filter uses very expensive instructions. While for the magnetic field only suggestions for optimization are given, the Kalman filter is optimized using SIMD instructions, improving speed by a factor of 1.5 to 2.4 depending on precision and vector width. Solving these issues requires different approaches, but neither solution can improve more than the time spent in this function. These gains, as impressive as they are, reduce runtime by only one to two percent, but using SIMD instructions makes code unreadable and hardware dependent. Maintaining an additional version that is independent of processor features was considered too cumbersome and error prone such that the improved version was never used for production. Another analysis of the reconstruction software using the PMUs in [39] shows that 30% of the time is lost due to call overhead, showing that the object oriented software design with many very small functions can imply a serious overhead. This is a general problem of object oriented design, which is hard to tackle in retrospect. Another analysis in [71] even suggests that tackling this problem using inlining will lead to larger binary sizes that lead to even higher delays due to more frequent memory accesses. It may also not be possible at all to improve the memory locality, e.g. when memory access is unstructured. The performance counters allow viewing an application's performance in terms of how CPU cycles are used, but this information is mostly helpful for an in-depth analysis of single functions or blocks to deduce possible optimizations. Optimization of large scale software can achieve larger gains with analyses acknowledging and recognizing complex relations.

4.2 Optimization areas

There are different areas where one can look for optimization opportunities. In the following I divided them into software environment, hardware environment and own software.

4.2.1 Software Environment

I split up the software environment into three groups: Operating system (OS), compiler and libraries. The OS provides access to the hardware and system libraries. Some hardware features may be unusable on some OS, such as 64bit pointers on a 32bit OS. System calls may be slower on some OS than on others. Hardware access through the OS can be configured to provide best performance only for certain access patterns. In the WLCG the OS configuration is setup by the grid sites and is not configurable for each experiment separately at Tier 0. Although plans to make this possible exist, it has prevented such optimizations as ABI32 builds, which requires the OS to support 32bit pointers on a 64bit OS saving up to 20% memory in the process, as suggested in [72]. Although compilers have been improving, only very simple code patterns are detected and automatically optimized, such as small loops over a structure of fundamental data types. In the ideal case these can be transformed to highly efficient code, potentially making this part of the code many times faster. Though the recognition of more complex patterns has been getting better in newer compiler versions, in the ATLAS code these types of loops only make up a very small part. The most part of the code needs to be manually optimized or at least massaged to fulfil the requirements of automated compiler optimization. The external libraries used at ATLAS are compiled for the systems they are used on but their source code is not modified save to resolve compatibility issues. Libraries may provide the same or similar functionality but have greatly differing speeds, which may also depend on the use case.

4.2.2 Hardware Environment

The hardware environment of a software consists of all physical parts relevant during execution. CPUs and other hardware components evolve more quickly than most of the used software, providing new features or allow faster execution of some operations. Specialized hardware often allows greatly improved execution speed for some scenarios, but require many changes in the code. Often it is not known if a problem can be reformulated in a way it suits new hardware architecture. Additionally, to testing runtime performance, costs for acquiring and running the hardware, i.e. electricity consumption, have to be considered. Normally, some parts of the hardware can be pinpointed as limiting factors for execution speed of particular software. This is referred to as CPU-bound in case calculation takes more time than memory access or memory-bound if the opposite is the case. Though this seems simple, there are many reasons and even more solutions to either limitation. Memory issues are particularly complex due to the deep hierarchy of memory. These range from CPU cache hierarchies to main memory, where connections to different sockets may play a role, down to mass storage. It is also relevant how the components are interconnected, influencing throughput and latency. For highly interconnected environments collaboratively calculating on multiple different nodes, physical distance plays into possible connection speed. For ATLAS, distributed calculation played only a minor role, as the problem is embarrassingly parallel. Embarrassingly parallel means that the problem can be divided in independent sub problems that can be computed without the need for communication between the different sub problems. For the ATLAS reconstruction this is the case, because each event can be processed fully independently. With the surge of many core chips, memory per core is decreasing, disallowing running separate instances of the program equal to the number of cores on each machine. This holds also for Tier 0 re-

sources which are controlled by CERN, as the ratio of memory per core cannot economically be maintained. Several approaches to use memory more efficiently are evaluated in the Subsections 4.3.2 and 4.3.4.

4.2.3 Own Software

Full control over code development, in this case the framework and reconstruction software, permits algorithmic and structural changes. This offers the largest potential for optimization because the source code and expertise is available, allowing in-depth changes to both algorithms and architecture. While in the software environment and hardware environment optimization areas one can only exchange one element for other predefined elements or tweak parameters such that they suit the requirements better, the own software can be changed completely. Software environment changes may require code changes, such as deprecated language features in a newer compiler version or deeply integrated external libraries, but these typically do not lead to a paradigm shift. Typical optimization methods are to find hotspots and concentrate optimization efforts on those parts. Optimizations may include change of data types and structures, minimizing overhead or replacing inefficient calculations. This type of optimization is most suited for very encapsulated problems that can be improved upon without the application in mind. In the large-scale ATLAS software project consisting of thousands of modules, a single or even low number of hotspots does not exist. Concentrating on optimizing each single part of the software is therefore not practical. The largest improvements come from algorithmic changes. If a problem can be remodelled such that it requires less computational effort instead of reducing the time spent executing the existing problem model, huge gains may be achieved touching only a few parts of the code. The time taken to calculate certain results may still be the same, but there may be fewer results required to solve the problem. Algorithmic changes may also shift hotspots such that other optimization methods can be applied. Care must be taken to consider the interplay of algorithms when changing results, as explained in Section 3.2.

4.3 Optimizations

4.3.1 Compilers

Many different compilers exist for many different languages. Prominent examples for C++ are Clang, G++ and the Intel C++ compiler. These compilers are being actively developed and have frequent updates, adding new language features or integrate more sophisticated code analysis to apply optimizations. These compilers differ slightly in language support and in their licensing. For ATLAS, this e.g. means that CERN employees may use the commercial Intel compiler due to a license agreement with Intel, but collaborating institutes, where much of the development is being done, do not profit from this license. While the Intel compiler provides better performance in some cases on the mostly Intel architecture grid sites, the licensing prohibits ATLAS from using this compiler. Also, usage of non-commercial compilers by non-CERN developers and recompilation on site with the Intel compiler is not feasible due to the differences in language support leading to incompatible code. Compilers are tuneable with so-called compiler flags, offering many different types of optimizations, some better for particular use cases than for others. The choice of compiler and compiler options also influence runtime speed. My tests with the GCC 4.7 O3 optimization flag showed 2-3% improvement over the O2 flag used in ATLAS. O3 optimization among other optimizations turns on auto vectorization and function inlining. Tests with the Ofast flag showed no further improvement, but, unless large gains were achieved, could not have been considered anyway, because Ofast violates IEEE standard making physics validation necessary for any small change. This is because errors may propagate

quicker or propagate in cases that normally wouldn't lead to error propagation. Therefore, this flag does not guarantee bitwise equal results for code changes that with IEEE 754 compliance would not have changed at all. ATLAS cannot easily use these flags, as releases have to be validated even for numerical changes, which is unpractical, and against policy for production releases. In the past, updates to the GCC compiler have all had positive impact on execution speed with new compiler option combinations being evaluated leading to further gains in the range of few per cent per version change.

4.3.2 Framework optimizations

The Athena framework determines the way events are processed, which is in a strictly linear and sequential fashion, and therefore also defines how its modules have to be written. The chain of algorithm is called with one event as input, and after an algorithm returns, the next algorithm in the chain is called. It is able to access any data from previous algorithms or tools written to the StoreGate service. The Athena framework did not allow simultaneous processing of different events within the same program instance before Run 2. Parallel processing without creating multiple instances of Athena is interesting as it has the potential to increase the proportion of number of cores utilized to memory required. This is important because of the CPU developments mentioned in Section 3.5. This problem has been addressed by introducing AthenaMP and AthenaMT, two branches of the Athena framework, which allow parallel processing of multiple events. The copy-on-write mechanism provided by modern Linux distributions is exploited by AthenaMP, requiring no changes to the algorithms and only small changes to the framework mechanisms as the underlying OS transparently resolves all potential conflicts. By forking after processing one event, most modules and services have already been initialized, maximizing the shared memory. Memory regions that are only read are not copied, leading to memory savings of around 40% with 8 concurrent processes while using the same amount of CPU time, see Figure 35. Shared between the instances are for example the geometry and the magnetic field, which don't change very frequently. The production release for Run 2 is AthenaMP.

AthenaMT is currently under development and based on the current multithreaded development of Gaudi. It allows additional parallelism compared to AthenaMP. AthenaMP spawns copies of algorithms and conditions and other data that changes between events assigning them to a process, and all objects created after forking only live in the process they were instantiated in. In AthenaMT these have to be centrally managed, requiring extra bookkeeping but also allowing a more fine-grained sharing of memory. Running algorithms in different threads of the same process means everything not allocated on the stack and not specified as thread-local can be accessed by all threads. Memory intensive data, such as the conditions data defining the detector conditions during at least one lumiblock of data taking, such that they remain the same for many consecutive events, can be reused. Parallelizing algorithms requires making them thread-safe, which has pitfalls and requires expert knowledge to avoid inefficiencies. Multiple threads, for example, can use thread-safe algorithms at the same time. Non-thread safe algorithms fulfilling less strict requirements can be either individually cloned for parallel execution and be reused to process another event in a pipeline-like fashion before other events finish processing. Independent algorithms of a single event can run in parallel, reducing the number of events that have to be maintained in memory to use more cores to their capacity. Changing the way Athena processes events to allow parallel execution of multiple events requires changes in many integral parts of the framework because it has been designed without parallelism in mind. Most algorithms will also require changes to make them fit such a model.

Figure 33: Processing model of AthenaMP. The flow does not change with respect to Athena except that all processes pick events from a single queue, balancing the load between processes.

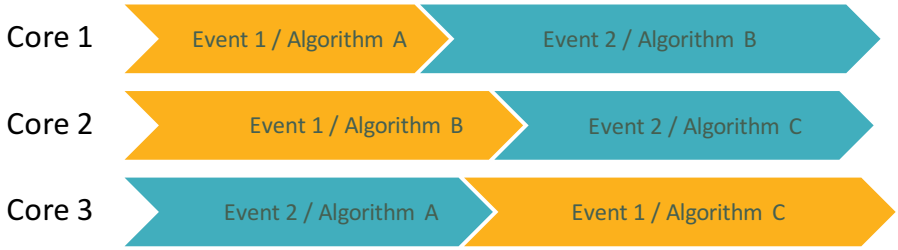


Figure 34: Processing model of AthenaMT. Independent algorithms of a single event can run in parallel. The goal is to fully utilize all CPU cores with less events running in parallel and therefore using less memory. The conditions during which an event was recorded make up a large part of the reconstruction.

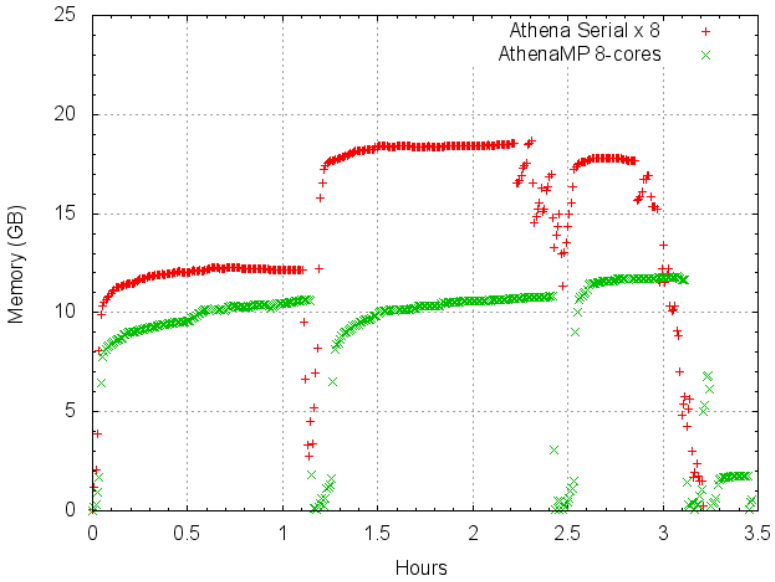


Figure 35: Memory and time spent using Athena with 8 individual instances and AthenaMP with 8 worker threads. Processing 2000 Run 1 events. The step between 1.2 and 2.5 hours is reconstruction which runs with release 20.1.4.12. Peak combined memory usage of the reconstruction step of the Athena instances is 18.5GB, but only 11GB for AthenaMP, corresponding to 40% memory saved. The time spent is almost the same for both frameworks. AthenaMP manages the load for all threads such that all threads end at the same time which shows in the more abrupt ending of the steps. The additional AthenaMP step at the end is merging of the output files, which isn't done for Athena and could therefore be left out. Plot from [73].

These already spent and expected future efforts can be justified because it allows using hardware resources that would otherwise be left unused due to the increasing gap between available memory and available number of computing cores.

4.3.3 Refactoring

In this subsection, I want to address possible changes in the code without changing algorithms, maintaining type, order and number of high-level operations performed. With high level I imply that underlying instructions or their order may change, but the mathematical formulas to calculate a result do not change. Changes made are purely structural, in that source code is restructured, data structures are changed or are accessed differently. This allows making code more maintainable in the case of code restructuring or may reduce the overhead of accessing data to perform mathematical operations.

In ATLAS, the Event Data Model (EDM) is the common data model of Athena for all subgroups, most prominently defining classes for particles and tracks, which are used across all detector subgroups. The design of the EDM has several flaws. Class members are created lazily but the access patterns show that they are always accessed, making lazy loading more expensive than creating all members of the enclosing object at the same time. Because identification of particle classes was done by type, to add another identifying binary property required creating an extra type for each already existing type, which was done by creating a different class for each type. The implemented changes are discussed in Section 5.5.

Another candidate for refactoring was the service providing access to the magnetic field. This service was written in Fortran, although Fortran had been abandoned in favor for C++ with the adaptation of Gaudi in the form of Athena. The Fortran magnetic field service was integrated into Athena using a wrapper. The Fortran code was structured in few large functions in a single file with more than 5000 lines of code. Variable names were three letters maximum and without comments or documentation. All this poses difficulties for developers to comprehend the code by reading it may have contributed to leaving the code in Fortran and as logs show in fact even untouched for almost a decade.

4.3.4 External libraries

The ATLAS software uses in the order of 100 different external libraries. Most of these libraries are rarely used, but some account for a significant amount of time spent. These include the GCC standard math library for trigonometric functions in which about 14% of total runtime was spent at the beginning of LS1 [12]. Another external library used extensively was the memory allocator with about 10% runtime spent.

Allocator and standard library provide functions the vast majority of applications require, which is why many alternative implementations for these functions exist, with varying performance. Some libraries are tailored to specific scenarios, e.g. some math libraries are optimized to perform well for the execution of the same operation many times over or for allocation of large blocks without the need to free them quickly in case of allocators. Some libraries provide the full API of the library they seek to replace while others just provide alternative implementations for certain functionality. Either case allows replacing the functionality by preloading the new library. A preloaded library is given higher preference than other libraries when locating a particular function. This way, functions provided by this library will be executed instead of functions with the same signature in other libraries. Preloading doesn't require any code changes but can be set when running the program but some libraries cannot reach peak performance if preloaded because they cannot be inlined by the compiler.

A third library is the CLHEP [74] library, whose linear algebra functions are heavily used for extrapolation. The CLHEP replacement is discussed in 4.3.5 and 5.3.

4.3.5 Linear algebra operations

Linear algebra functions from the CLHEP library were responsible for 8% of the runtime [12]. Linear algebra operations are required for extrapolation to navigate the geometry of the ATLAS detector, to compute least square track fits and for Kalman filtering. This has to be done many times for each track, e.g. every extrapolation step to generate tracks from space points or calculate particle trajectories in simulation is often only a few millimetres through the detector. CLHEP was written and maintained by CERN, but development has been discontinued except for bug fixes, making this library a candidate for replacement. Unlike the previous examples of external libraries, CLHEP, short for "Class Library for High Energy Physics", is highly specialized, such that no replacement with a sufficiently similar interface exists. Alternative libraries provide similar functionality, but with a different API, requiring code changes everywhere the library is used. Using a wrapper to mimic CLHEP was not possible on the tested replacement libraries because their member classes interact differently from CLHEP. Exchanging the linear algebra functions only in the main contributors is infeasible as there are no hot spots of CLHEP usage, respectively the main contributors do not amount to a significant fraction. Therefore, only a large-scale intervention can lead to significant gains. Analysis of a reconstruction job [12] with the Intel Pin tool [75] shows millions of Vectors and Matrices being created and hundreds of thousands of matrix-matrix and matrix-vector operations, see Table 3.

Operation Type	Dimension 1st Arg.	Dimension 2nd Arg.	Calls per Event
Matrix*Matrix	3 x 3	3 x 3	29333
Matrix*Matrix	3 x 2	2 x 2	28139
Matrix*Matrix	3 x 5	5 x 5	13003
Matrix*Vector	5 x 3	3	23676
Matrix*Vector	3 x 5	3	11802
Matrix*Vector	1 x 5	5	4718

Table 3: Matrix and vector operations per event in a reconstruction job on a Run 2 event with release 17.2.7.9. Note that CLHEP has different classes for vectors and matrices leading to the fact that the measured 1x5*5(x1) operation is not a vector-vector operation but a matrix-vector operation.

4.4 Time per Tracking Step in Different Pileup Scenarios

In this section I analyze the time of different tracking steps for different pileup scenarios and predict their impact for future high pileup scenarios. The three steps are seeding, tracking and ambiguity solving. I measured the average over 100 events for each step for different pileup scenarios and fitted curves through each step, such that their behavior in higher pileup scenarios can be predicted. Knowing the runtime of each intermediate step from seed creation to ambiguity solving and the number of combinations in each of this step allows visualizing which step is dominant in which scenario, and how improving the step impacts the total runtime at different pileup scenarios.

Figure 36, Figure 37 and Figure 38 show curves fitted through multiple datapoints, predicting the impact of pileup for different algorithms. The plots show the created number of output as a function of pileup and runtime as a function of pileup. The 2nd order function agrees very well with the measured data and visualizes that the algorithm's runtime is not linear with the event pileup. Algorithm analysis suggests an even higher complexity, and tests with higher pileup suggest an even steeper curve, but testing conditions were different such that they could not be included in this analysis.

Stacking the data from the different plots shows the extrapolated development of the

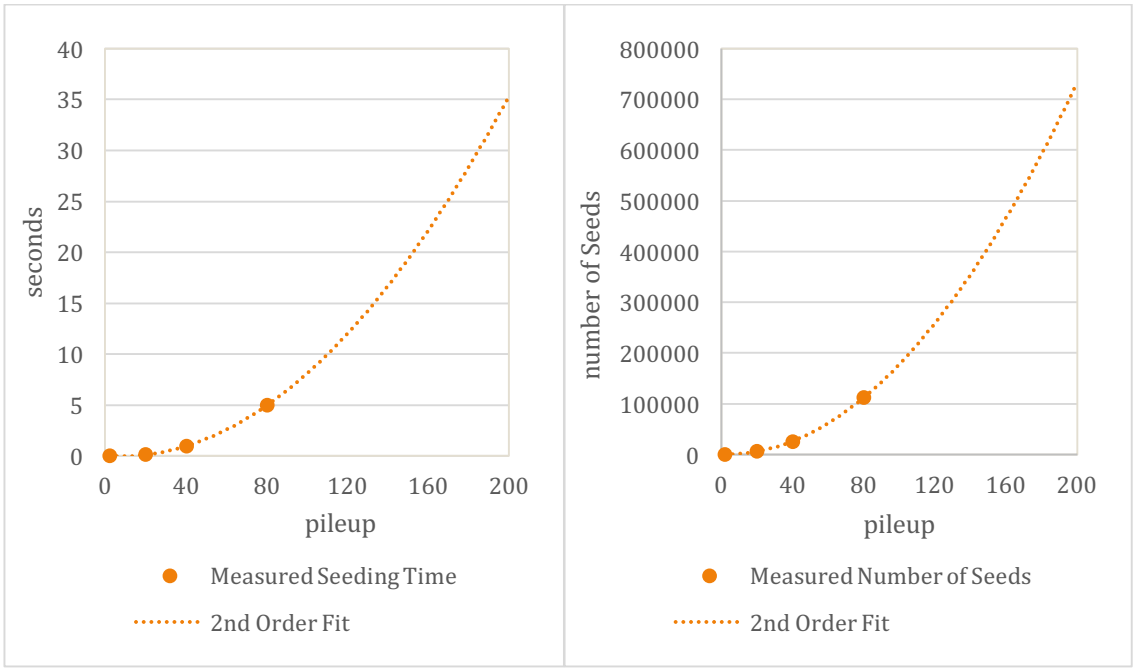


Figure 36: 2nd Order fit to measured data points of data relevant for seeding performance. The left graph shows the time spent in the seed finding for different pileup scenarios. The right graph shows the number of seeds generated in different pileup scenarios. Both graphs show that the runtime increases unsustainably for very high pileup scenarios.

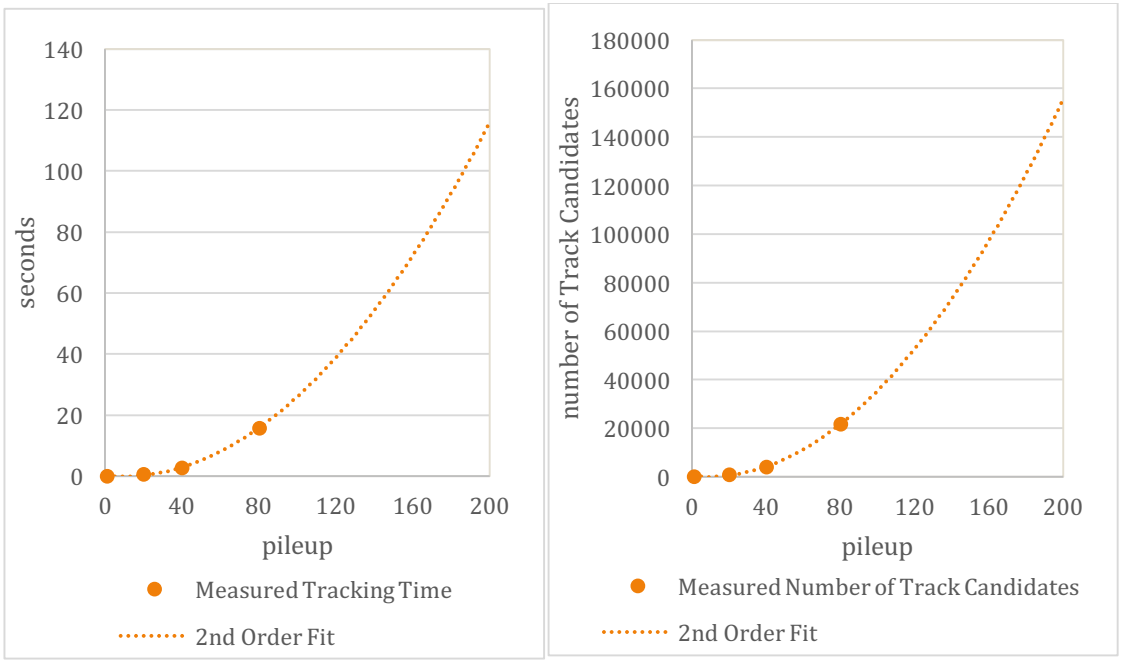


Figure 37: The equivalent data shown for the Seeding in Figure 36 is shown here for the track finding.

whole tracking timing up to events with 200 pileup interactions in Figure 39. The first observation is the clear dominance of the tracking step over the seeding and ambiguity solving, which apparently increases with higher pileup. It also shows that the ambiguity solving takes up a much larger share at smaller pileup scenarios than at 200 pileup events. This extrapolation assumes a nearly linear growth with the number of output combinations. Correlation between the measurements of timing and combinations is above 0.999 for both seeding and tracking. This assumption allows to increase or decrease the time

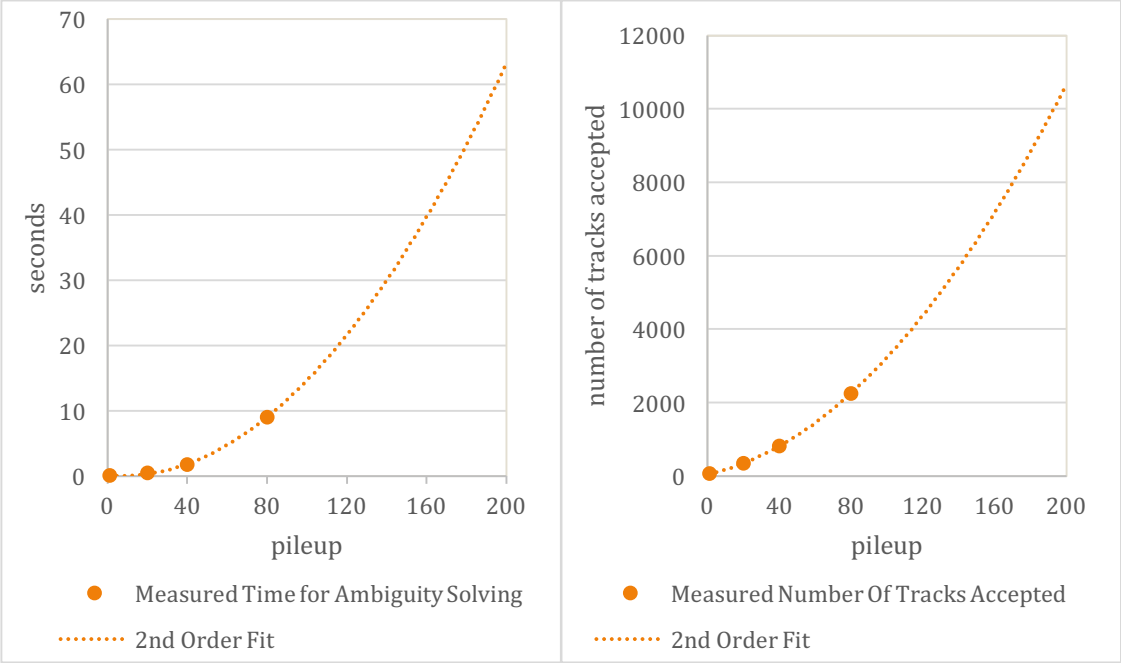


Figure 38: Timing and tracks accepted for the ambiguity solver fitted with a trendline. The number of tracks accepted does not increase linearly, indicating that a higher number of fake tracks is accepted. The number of tracks accepted does not influence any of the analyzed algorithms such that the correlation between the two curves is not important. The number of tracks increases linearly with higher pileup, but with 80 pileup a sharp increase can be observed. This suggests the Ambiguity finder cannot distinguish real tracks and fakes at higher detector occupancy.

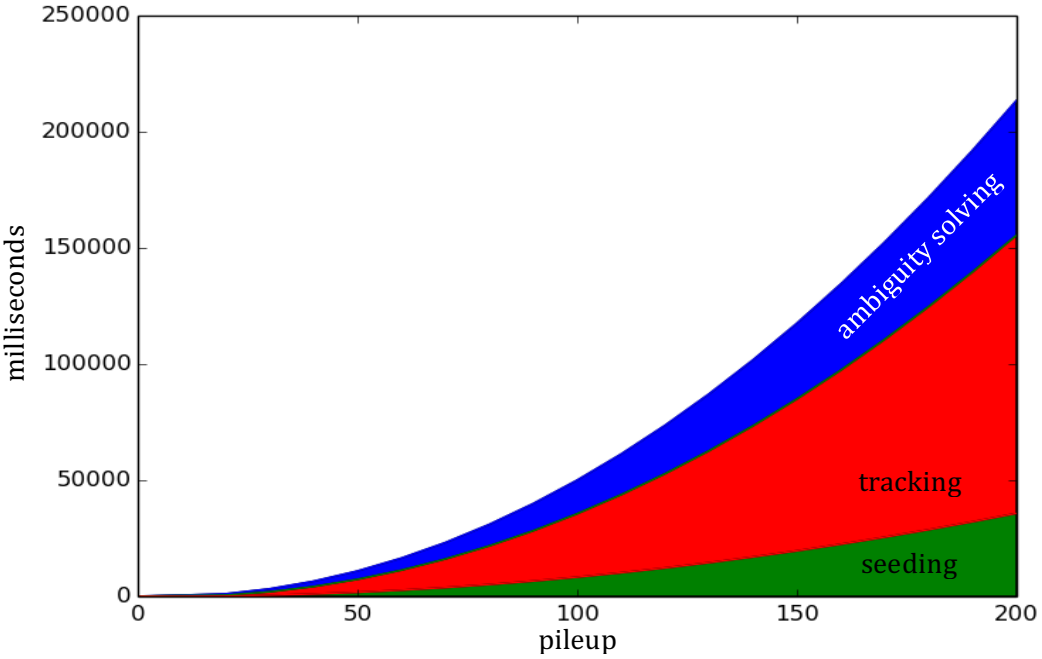


Figure 39: Extrapolation of measured times of each tracking step to high pileup scenarios with up to 200 pileup interactions per event. The tracking is expected to take more than half of the total runtime for events with 200 pileup interactions.

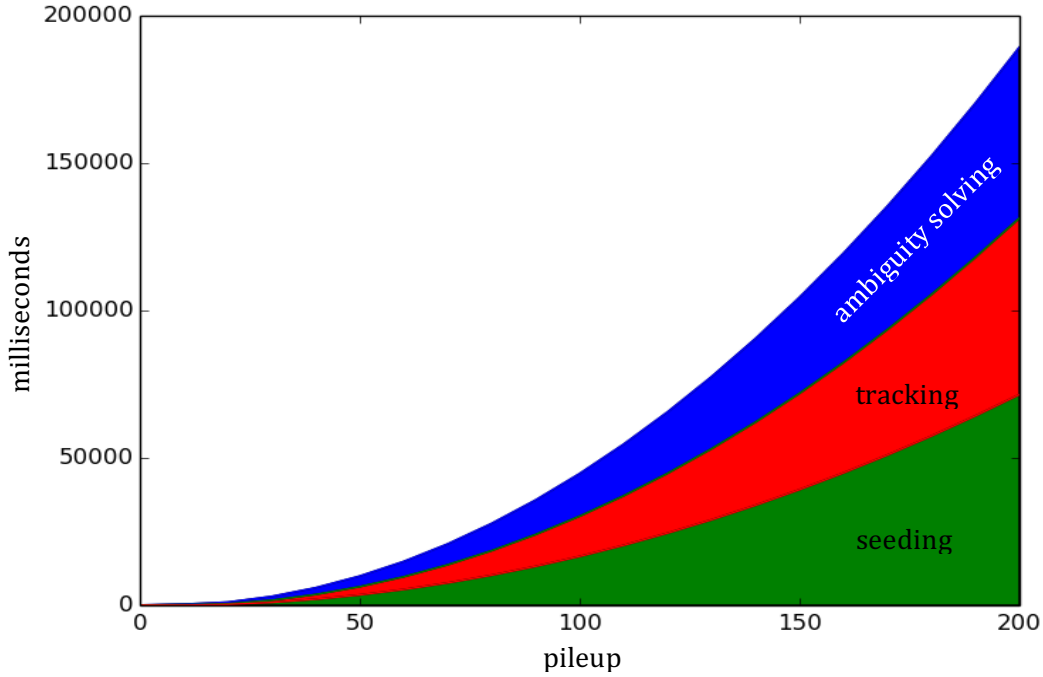


Figure 40: The effect of doubling the runtime for seeding while halving the number of created seeds. While the total runtime is slightly reduced for high pileup scenarios, the seeding step is dominating the algorithm chain.

taken per combination and change the number of outputs and observe the effects on the total runtime. If some theoretical optimization was able to reduce the number fake seeds by half, while at the same time doubling the runtime, the time of the subsequent algorithms decreases. Overlaying these graphs shows if it results in an overall gain or not.

The tracking algorithm chain shown in Figure 40 shows that such a change could reduce the total runtime by a small margin, but it would also shift the weight of the most expensive algorithm, which in ATLAS currently is the tracking to the seeding. Such a change could be desirable as the seeding does not have complicated bookkeeping mechanisms which prevent the parallelization of this step.

4.5 Dependencies of the ATLAS Reconstruction

Most Athena Algorithms in a reconstruction job are part of a dependency chain. Their input comes from another Algorithm and their output serves as input for a next Algorithm. Communication between the different steps of the algorithm chain is done via the StoreGate service by allowing Algorithms to read and write data using this service. An ATLAS script allows observing which Algorithm writes and reads which data to construct a dependency graph. Some Algorithms bypass StoreGate by communicating through shared Tools (violating ATLAS coding rules). To achieve a more complete picture I modified Tools that are known to be (ab)used for this purpose, logging which Algorithm accesses which data they hold. The combined graph from both monitoring methods, shown in Figure 41, does not show all modules in the chain but only those communicating or writing a final output. The graph nonetheless gives an impression of the complex dependencies in reconstruction. Reconstruction consists of more than 600 modules interacting, most of which are tools used by the algorithms in the shown chain. Some algorithms do work independently of others, but most take output from or create input for some other module.

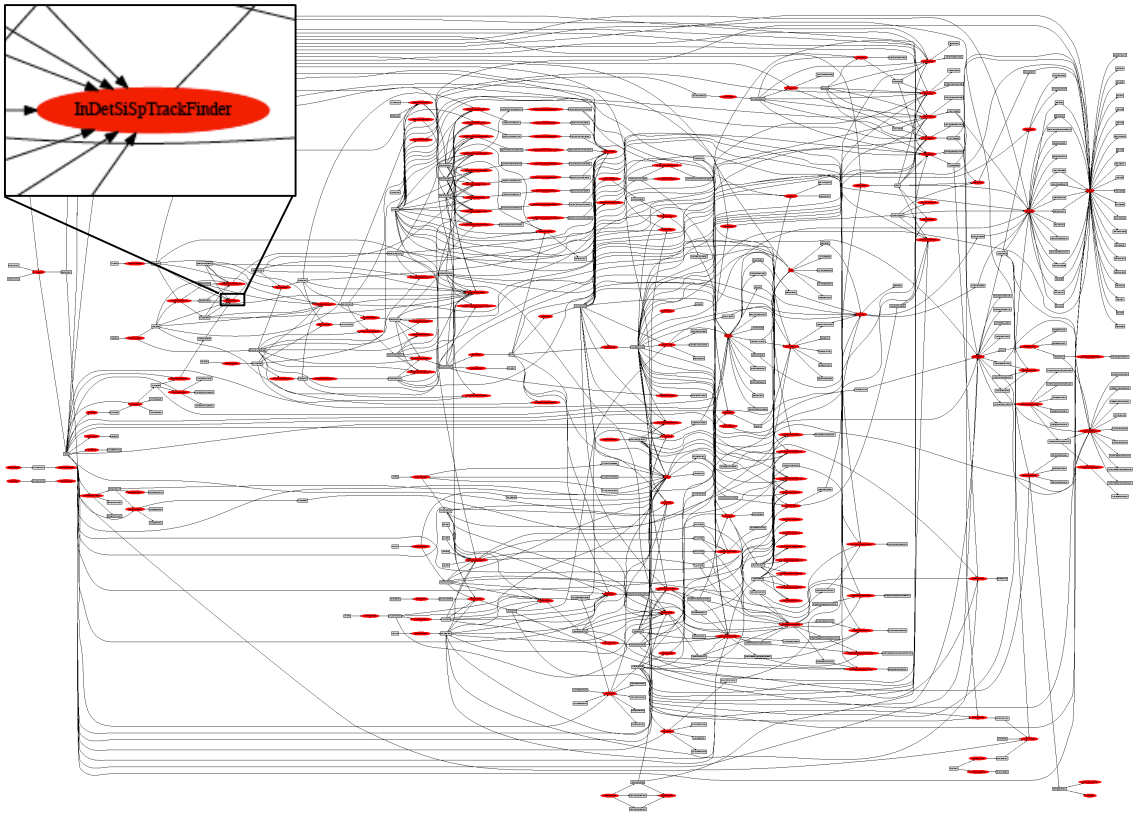


Figure 41: Dependency graph of all algorithms writing and reading from StoreGate with Athena release 17.2.7.9. Boxes in grey denote data while boxes in red denote Athena algorithms. The highlighted InDetSiSpTrackFinder is the most expensive algorithm. It depends on seven other algorithms' output and writes a one collection which is read by multiple algorithms.

This defines an order of execution, which is important for parallelizing algorithms. Additionally to the dependencies observed through data access patterns, a deep call chain exists. Most algorithms use many tools and services and each of the tools can in turn also use tools and services, which are not shown in the graph if they do not access data through the communication service StoreGate.

4.5.1 Dependency and Intra-Event Parallelizability Study of ID Algorithms

The Athena framework is based on the Gaudi framework that was developed by the LHCb experiment in 1998 [76], ten years before the inauguration of the LHC, not yet foreseeing the developments in computing hardware and in the performance of the LHC. By 2005, leading scientists designing the CERN computing grid assumed that clock rates would increase for far longer than they did and that multicore CPUs would not enter mass market [40]. This is why Gaudi and Athena are designed for sequential execution of events, and why the algorithm chain for reconstruction has been designed and optimized without parallelism in mind. Many of the reconstruction algorithms have been developed before ATLAS was built, before parallel programming had entered the server market. This lead to the current sequential model processing an event fully before starting to process the next event within one Athena instance. Within the chain, each algorithm processes an event before passing its output into the next algorithm. The full dependency chain of ID reconstruction contains 39 communicating algorithms, creating complex dependencies. It should be noted that the ID reconstruction consists of many more modules, as each of

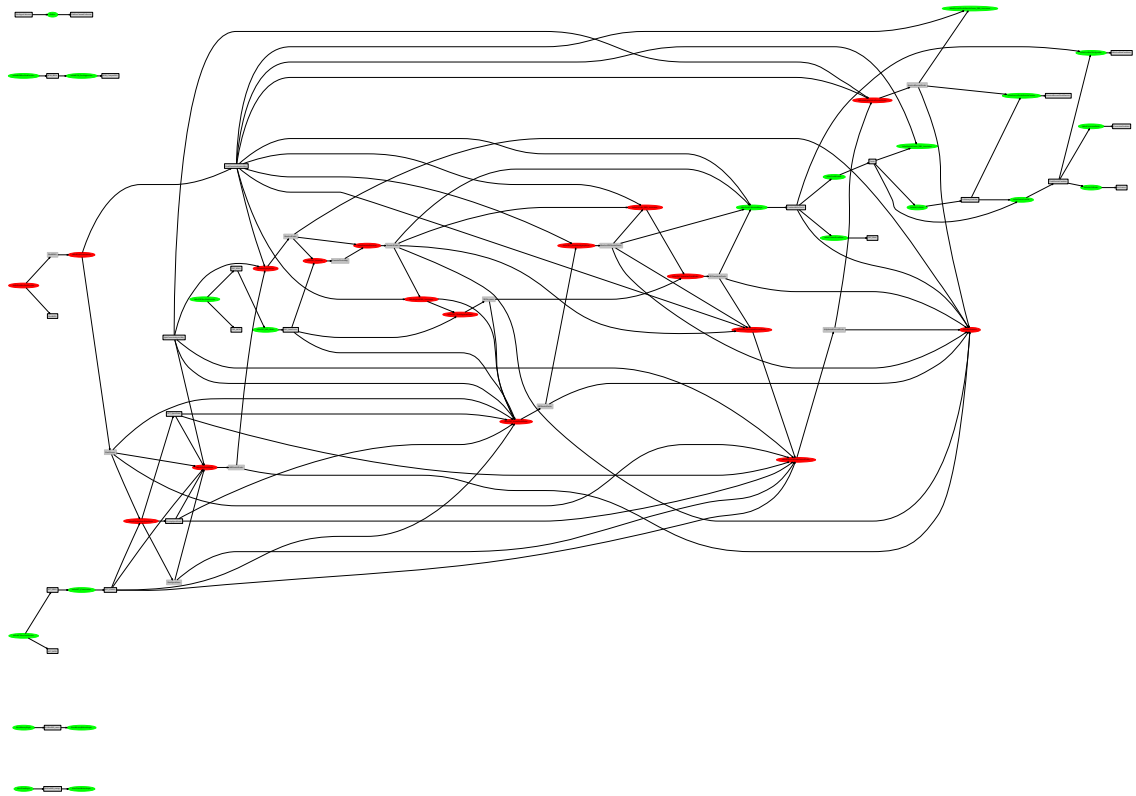


Figure 42: Dependency graph of ID algorithms accessing StoreGate. Algorithms on the critical path are colored *red* and modules not on the critical path are *green*. The data collections are depicted in grey. Connections to a data collection coming from a module left of it means the module is writing to it while a connected module right of the data collection is reading it. The graph shows the modules are strongly interconnected leading to many modules being part of the critical path. Tested with Release 17.2.9.7.

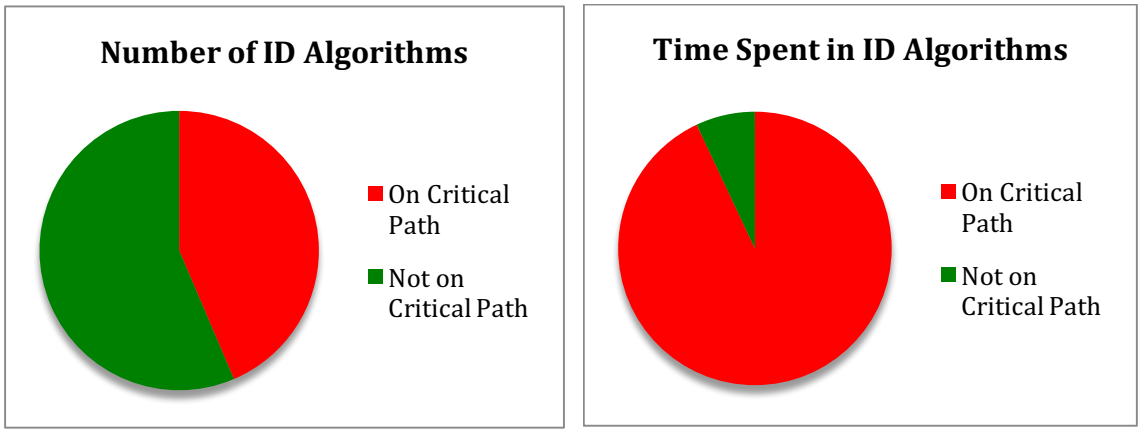


Figure 43: Dependencies of the full ID algorithm chain by number and by time spent. 17 of the 39 algorithms are part of the ID chain but they make up at least the measured 95% of the ID reconstruction time. Some dependencies may have been lost due to the (discouraged) usage of Tools to communicate instead of StoreGate. Adding these modules may lead to an even larger fraction of algorithms on the critical path.

these algorithms is at the top of a deep call chain. With my experience of having analyzed a limited number of algorithms I estimate each algorithm uses around ten tools and services, which would mean hundreds of modules are involved. The dependency graph of the ID modules accessing StoreGate in Figure 42 has the critical path highlighted. The

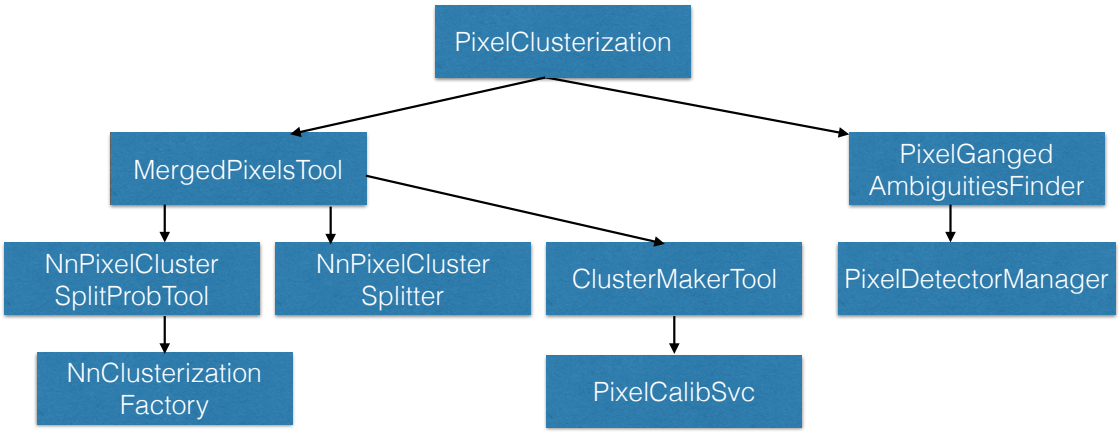


Figure 44: PixelClusterization call chain of Tools and Services. The Services call chain is not resolved any further.

critical path is defined as the path of modules with sequential dependencies that takes the longest time to execute. 17 of the 39 algorithms are part of the critical path. I combined the graph with the algorithm timings using Run 1 ttbar events to generate a typical workload. A clear dominance of the critical path shows as depicted in Figure 43. With around 95% of the time spent in algorithms with sequential dependency, a maximum of 5% of the ID time can be saved by running algorithms in parallel within one event. The most cost intensive algorithms use bookkeeping mechanisms which order the input data, such that they can only run after the previous algorithm has generated all its output. A simplified chain of algorithms for the Inner Detector reconstruction doing the most compute intensive work is explained below and their conceptual similarities and differences important for parallelization stressed.

Within ID reconstruction, *data formation* is the first step, which prepares data for further processing by other algorithms. It takes the raw data from the detector and transforms it, resolving derivable information.

First is the PixelClusterization, which takes all pixels that measured a charge deposit over threshold during one event. Adjacent measurements are joined to clusters, which are assumed to stem from at least one particle. Clusters can be split into multiple clusters depending on shape and charge deposit or are considered to stem from more than one particle. The distance from the interaction region is a major factor to distinguish the measurements of multiple particles. The IBL layer, which is closest to the proton-proton interactions, has the highest resolution with 6 million readout channels. Clusterization for the SCT is simpler because each strip only has two neighbors instead of the eight neighbors in the grid-like structure of the Pixel detector. The data formation for the TRT is the conversion of measurements to drift circles. There is no clusterization for the TRT. All of the above tasks can be executed in parallel by dividing the detector in regions, making sure no detector modules are shared, these tasks require no synchronization. The Pixel detector consists of 1744 modules, the SCT has 15912 modules and for the TRT, each straw can be operated on independently. The data preparation accounts for around 4% of reconstruction runtime in a Run 2 event as shown in the measurements presented in Subsection 4.1.2. The largest fraction of time of the data preparation is spent in the PixelClusterization due to the more complex algorithms. Parallelizing over all pixel modules would be sufficient to fully utilize even manycore architectures and reduce the sequential PixelClusterization time to insignificance. In an attempt to parallelize this step, I started by rewriting the PixelClusterization algorithm to be thread safe. The PixelClusterization uses multiple Tools and Services, which have to be adjusted for thread safety themselves or guarded with mutexes. The tools could not be guarded by mutexes without compromising performance. The Tools use other Tools and Services in turn, Figure 44 shows the call tree omitting the data structures. Athena foresees access to Tools and Services through han-

dles, which automatically make the tools mutable. All Tools themselves also have mutable member variables. The complex structure and the small expected return finally made me abandon the project, but it showed thread safety cannot quickly be achieved by a small work force in the current state, even on such a comparably simple processing step. It requires involving the various module developers, they have to contribute a significant amount of work and have to be educated to program in a thread safe way. This will happen when developers are asked to get their software ready for AthenaMT, which requires thread safe modules as presented in Subsection 4.6. Once thread safety of a full call chain of some algorithm is achieved, parallelization schemes as naïve as the presented ones promise high degrees of parallelism with low implementation and maintenance effort.

For SCT and Pixel detector, the measurements are converted to space points in the space point formation. Space points contain the 2D location on its detector surface and an error matrix. For SCT the space point formation consists of the combination of SCT clusters from the two layers per detector element. Strip clusters of two strip layers glued together are combined to one space point if both clusters have measurements in the same region. Due to the tilt of two layers with respect to each other, this area will be very small yielding an accurate 2D measurement on the detector element. For Both SCT and Pixel detector, the 2D locations on the detector element and the location of the detector element can be combined to establish a space point's 3D location. The drift circles from the TRT are converted to drift circles instead, reflecting the location ambiguity around the central wire and the missing measurement in wire direction. Similar to the clusterization, parallelization could be achieved by creating tasks for each detector module of the Pixel and SCT detector.

The *pattern recognition* combines the space points in different ways to prepare and build the tracks. This is the most computationally expensive reconstruction step. It is explained in Section 2.7 in detail.

The seed finding and combinatorial track finder doesn't allow naïve parallelization approaches such as the ones mentioned before. Together they account for 46% of the ID reconstruction runtime in a Run 2 event. It is therefore the most important step to parallelize, albeit a very complex one. It keeps track of measurements already used to create a track candidate to avoid reusing them, reducing the complexity of this step. This prevents parallelizing indistinctly over the combinations. Though, a parallelization would be possible over regions, maintaining the bookkeeping locally for each region but reusing the measurements of tracks crossing region borders. The clean-up of the additionally generated combinations due to the lack of bookkeeping in the cross-border regions could be resolved in an additional step. Critical for this approach is the number of cross-border combinations, because it cannot be reduced by bookkeeping. A higher degree of parallelism therefore would lead to a higher amount of work done in vain, such that the degree of parallelism is limited as long as bookkeeping is important to reducing the number of combinations. As shown in Section 4.7 it loses importance with higher pileup scenarios. Approaches to parallelize over regions are presented in Chapter 6.

The ambiguity solving step took 25% of the ID reconstruction runtime with a Run 2 event. Efficient parallelization of this step is particularly difficult because of the large role the bookkeeping plays. A similar scheme to sort tracks by regions could be employed as for the combinatorial track finder. Sorting within each region by rating, shared clusters only exist for tracks crossing a region. A second sequential step could compare the remaining tracks for duplicates from tracks crossing region borders.

Due to the sequential nature of Athena, parallelization within the Athena framework is only feasible over the data of a single event. Fortunately, all expensive algorithms iterate over set of data large enough to make parallelization within an event feasible. The bookkeeping some algorithms do to reduce complexity could be broken down into smaller parallelizable segments with as little overlap as possible and comparing the results from neighboring segments with one another in a second step. A trade-off between parallelization and additional work due to a higher number of segments needs to be found.

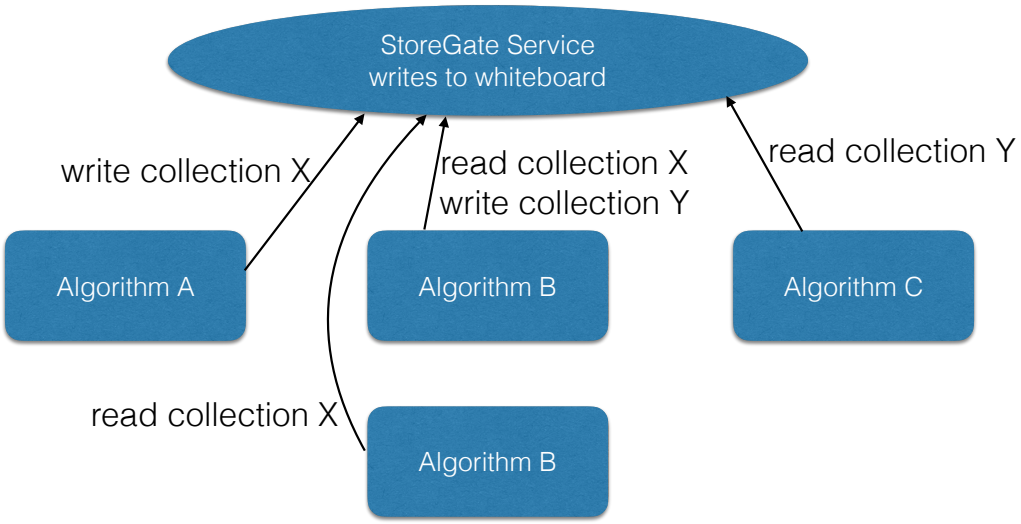


Figure 45: Using the requested collection names, the underlying Whiteboard behind the StoreGate Service can resolve algorithm dependencies and schedule algorithms. The dependencies are used to create precedence constraints and allow parallel execution of unrelated algorithms. Manually specifying the order of execution is not necessary.

4.6 Caveats for Parallel Processing in the Reconstruction

The ATLAS software suffers from problems of many grown solutions, which also affect parallelizability. The framework has been designed without parallelism in mind and algorithms have been optimized for sequential execution. AthenaMT tries to conquer some of these limitations without requiring a structural change in the algorithms. Parallelism is achieved by running multiple events in a pipelining fashion and running algorithms in parallel that are independent. AthenaMT requires the dependency graph of algorithms and exploits that algorithms are required to communicate using StoreGate. In multithreaded Gaudi, the storage mechanism behind the StoreGate service has been reimplemented to monitor read and write accesses, using it to determine satisfied dependencies without having to manually specify the order of execution, see Figure 45. A problem with this approach is that some algorithms do not write out their whole output at once but create empty containers they fill at a later point. The whiteboard doesn't see the content of the containers and assumes the dependency is satisfied as soon as the container is first created. These algorithms would need to be changed to be usable with AthenaMT, as would algorithms that communicate through other non-foreseen means such as shared Tools. Another parallelization approach supported by AthenaMT is to run multiple events in parallel on the same Algorithms, which either requires these algorithms to be thread safe or to be cloned. Unfortunately, many functions that are marked as `const` use mutable variables that affect the result. This requires even seemingly simple parallelizations to examine the full chain of tools and algorithms used for mutables. The cloning approach doesn't have these problems, but requires manual implementation and handling of the multiple instances in contrast to the fully transparent copy-on-write forking of multiprocessing used in AthenaMP. Additionally, the expected gains for reconstruction through in-event algorithm parallelism are relatively low as shown Section 4.5.1. The AthenaMP framework supporting forking processes with no changes required on the modules is already available and is used for production during Run 2.

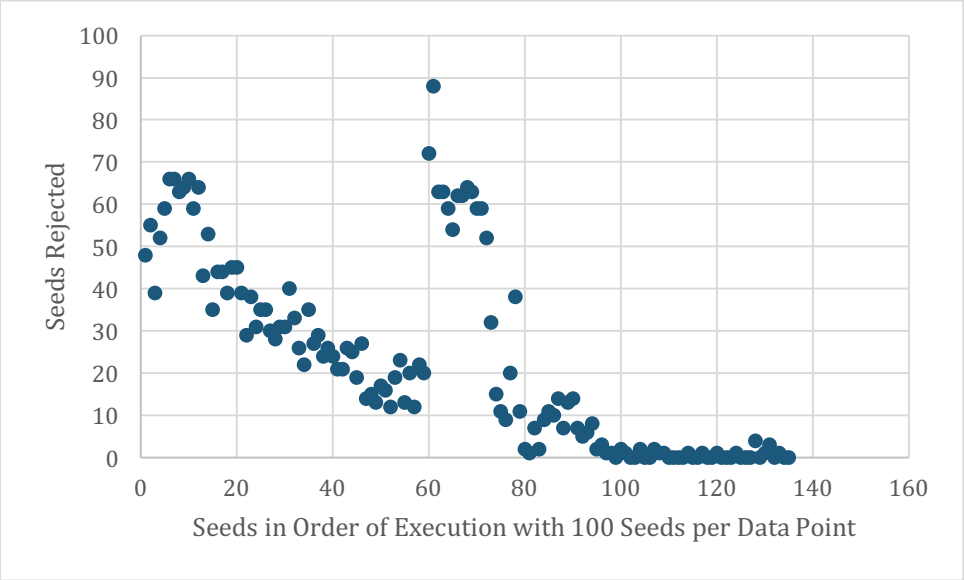


Figure 46: Seeds rejected before extrapolation over the course of one ttbar event with 40 proton-proton collisions. Each data point stands for 100 analyzed seeds. The spike in the middle stems from a different type of seed which has stricter criteria to be used for extrapolation.

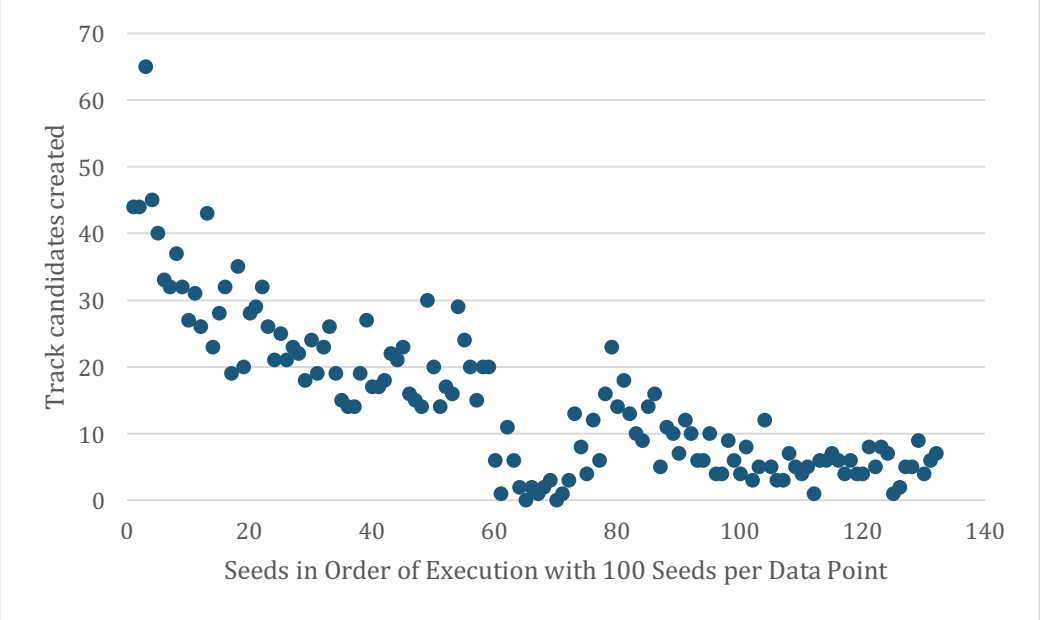


Figure 47: Number of tracks created over the course of one event. Each data point stands for 100 analyzed seeds. Initially, each seed has a higher probability to result in a track while in the end only few tracks are found. This is both due to the bookkeeping and because probable fake seeds have been sorted to be at the end.

4.7 Influence of Bookkeeping in Tracking in Run 2 Production

The bookkeeping in the tracking step of the reconstruction has been introduced during Run 1 to avoid reconstructing tracks that have already been found, because the reconstruction requires an extrapolation of the track through the whole Inner Detector. This is, as mentioned earlier, a very costly step. To achieve this, the space points of a seed are compared with the measurements of all tracks that have already been found, and only if the number of measurements of the seed used in a single track is lower than a threshold,

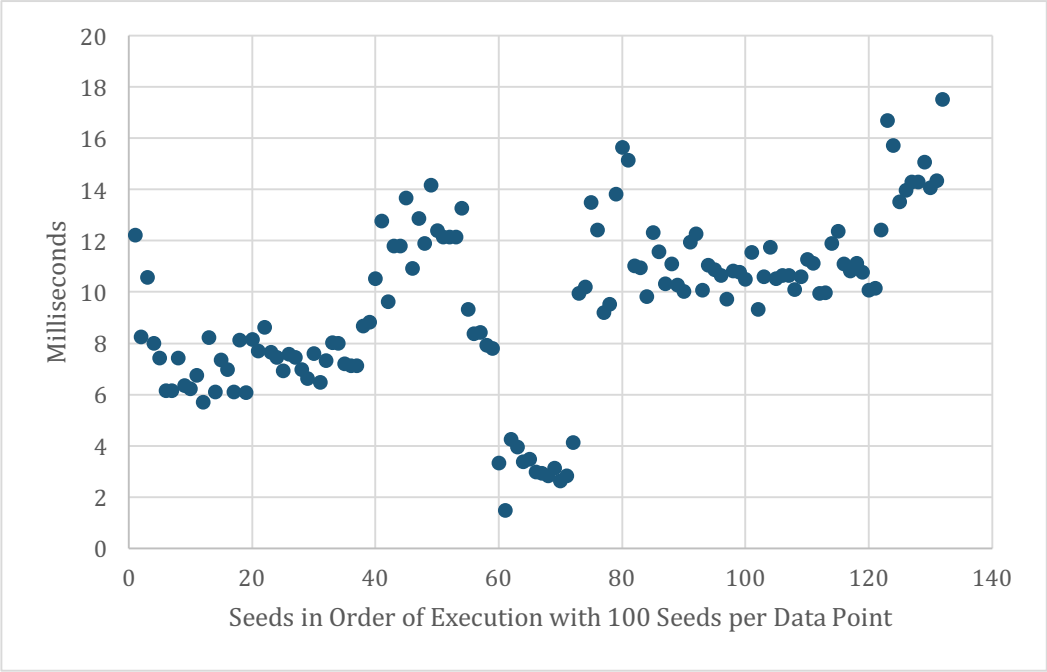


Figure 48: Time taken per 100 seeds over the course of one event. Average time per seed is higher at the end than in the beginning although fewer tracks are found as Figure 47 shows, indicating that the bookkeeping does not reduce overall processing cost at later stages.

the seed is used to perform a track search. The seeds are sorted such that the probability to find a track is higher for the first seeds, to maximize the impact of the bookkeeping. The comparison process becomes slower the more tracks have already been found. The bookkeeping can neither prevent fake seeds for which no track exists from being extrapolated. No previously found track can be found because none exists for a fake seed, such that the extrapolation can only be stopped once the algorithm decides it cannot find a track. The bookkeeping only prevents a fake seed from being extrapolated if enough of its measurements have already been used in another track.

The number of fake seeds increases with the number of proton-proton collisions per event as shown in Section 4.4, much faster than the number of actual tracks, which only increases linearly. With higher pileup, the number of found tracks is small compared to the number of fake seeds. Because pileup has increased many times over since the introduction of this feature and because it prevents intra-algorithm parallelization as shown in Section 4.5.1, I wanted to analyze its effect in current and future pileup scenarios. Instrumenting the code shows that while initially many seeds can be rejected, this number drops the further the algorithm progresses to seeds which are not likely to yield a track, as shown in Figure 46 and Figure 47. The graphs show aspects of the track finding over a single event with 40 proton-proton collisions. Although the number of found tracks reduces, the average time per found track does not reduce but tends to increase, as Figure 48 shows. Comparison of how much time is spent per seed in the 100 seeds represented by the first data point and in the 100 seeds represented by the 100th data point of Figure 48 show that the distribution is different. As Figure 49 shows, of the first 100 seeds, some seeds took a lot of time to process while others took a moderate amount of time. As the seeds are sorted by descending order of likelihood to yield a track, seeds that take only little processing time are part of a track which has already been found. These can therefore be found quickly in the still short list of already found tracks. The distribution of the seeds of data point 100 also shown in Figure 49 has much smaller tails. For these seeds the lookup in the list of increased size is performed, but the exclusion of the seed is much less

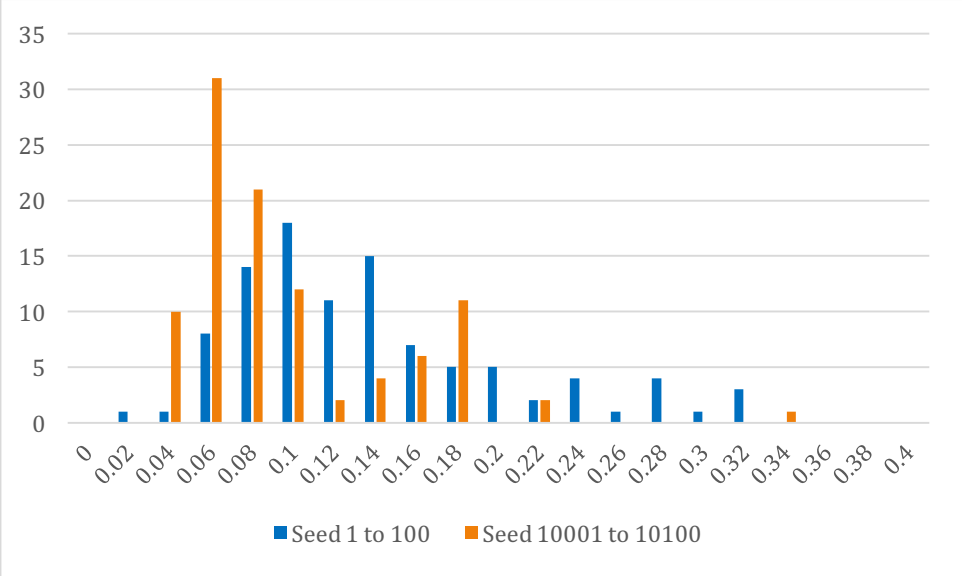


Figure 49: Distribution of processing time per seed for the first 100 seeds and 100 seeds after already having processed 10,000 seeds.

	Track Finder Unchanged	Ambiguity Solver Unchanged	Track Finder No Bookkeeping	Ambiguity Solver No Bookkeeping
pileup of 40	2315ms	1283ms	2733ms	1271ms
pileup of 80	14388ms	5660ms	15024ms	5728ms
pileup of 140	64528ms	16099ms	66976ms	16533ms

Table 4: CPU performance with and without bookkeeping for Run 2 and future pileup scenarios. Test run with 1000 events for 40 pileup collisions per event and 100 events each for 80 and 140 pileup collisions per event. The advantage of bookkeeping reduces with higher number of pileup collisions.

likely due to the seed sorting. Extrapolation takes less time than a full track as the search is stopped when not enough measurements can be found. This is why most seeds take a moderate amount of time to be processed.

As analyzed in Section 4.5.1, this bookkeeping mechanism is blocking parallelization of this step. To analyze its impact, I profiled reconstruction with bookkeeping deactivated. This cannot negatively affect physics performance, as without the exclusion of seeds more tracks can be found, but not less. Duplicates are not a problem because after a track has been found, it is always tested if this track has been found previously. The ambiguity solving after the tracking performs slightly more work because of the additional tracks that are found. Timing measurements show that the ID track finding and the ambiguity solving each take longer, see depending on the number of pileup collisions, see Table 4. The same measurements show that the effect decreases as expected with higher number of pileup collisions. Given that this mechanism prevents parallelization of the track extrapolation and yields especially for high pileup scenarios only low percentage gains, removing it may be acceptable if parallel resources would otherwise remain unused.

4.8 Conclusions

The analyses presented in this chapter identified several optimizations and ruled out others or showed the effort required to perform them. The critical path analysis showed little gains can be achieved from parallelising ID reconstruction algorithms within a single event, which is one of the main features of AthenaMT. The results led to prioritizing

parallelization within individual algorithms. The parallelizability analysis of the individual ID reconstruction steps shows parallelization opportunities in different steps and where obstacles for parallelization lie.

While optimizations on the OS have shown great potential, they are difficult to maintain and are unlikely to be included for ATLAS as the infrastructure common to all experiments would have to support it, which touches the interests of other experiments and institutes providing the computing sites. The focus for optimizations was instead put on intrusive and non-intrusive replacement of libraries, refactoring of algorithms, changing algorithms to run in parallel and exchanging algorithms by more efficient ones.

Order of implementation should be low to high effort to achieve the largest possible gain as quick as possible. Long running projects, such as replacing the CLHEP library need to run in parallel with other projects. Some optimizations will lead to touching large parts of the codebase and reconstruction contains unmaintained code, such that code cleanup should be conducted where outdated technologies or violations of the ATLAS coding standards are encountered. Some optimizations cannot be performed without profound changes, e.g. changing the algorithm flow or the underlying framework or acquiring new hardware. Breaking backwards compatibility may be necessary to introduce new technologies.

Optimizations applied during the writing of this thesis and to which I contributed are presented in the next chapter. The main focus lies on the side of the own software and the tools and libraries it uses, which promises the largest gains with the lowest effort and cost to implement the changes.

5 Software Integration of Optimizations

This chapter describes the software projects to improve the reconstruction software during LS1. The described projects differ much in effort required and improvement achieved. Each section in this Chapter discusses one or few closely related projects. The first subsection describes project assessment and how impact was predicted. The following subsection describes the project implementation and the last subsection describes the measured or estimated impact. Tests are run with Run 2 events as described in Subsection 4.1.1 on an Intel Nehalem CPU with 2.2 GHz and 24 GB memory with Scientific Linux CERN 6 (SLC 6) if not specified otherwise.

5.1 Impact expectation of optimizations

Independently of the implementation costs, I identified three core points for an impact analysis. The points are the immediate impact on execution speed, the future impact on the speed and the impact on maintainability. For most optimization efforts, immediate impact is the main motivation. For long-lived projects such as the ATLAS detector code, efforts to increase maintainability or to allow performance gains in future settings, i.e. with different data or with a different framework, pay off in a longer perspective. Simplifying the maintenance of deeply integrated libraries such that they can be exchanged in a single point may drastically reduce the cost of future optimizations such that one can expect to continue to gain from such improvements.

5.2 External Library replacement

5.2.1 Assessment

ATLAS uses various external libraries that have been in place for many years. In some of these libraries, significant time is spent. Three libraries totalling 35.5% of the reconstruction runtime have been identified, see Table 5 with tests on Scientific Linux CERN 5 (SLC 5) and release 17.7.2. In collaboration with the ATLAS Performance Monitoring Board I conducted measurements showing around 14% reconstruction runtime spent in

libm [12]. Significant time was spent throwing floating point exceptions, which should not occur frequently, and in case of occurrence not be time consuming, as this did not include any exception handling. 15% runtime are spent in the memory allocator tcmalloc version 0.99. The CLHEP library, which became a candidate for replacement with the end of its support, showed to be responsible for 6.5% of the reconstruction runtime in my measurements. Different options have been weighed for all three libraries, due to the complexity of replacing CLHEP it is described separately in Section 5.3.

	Reconstruction time spent in library
tcmalloc 0.99	15%
libm	14.1%
CLHEP	6.5%
Total	**35.6%**

Table 5: Libraries with a huge impact on execution speed. More than one third of reconstruction runtime is spent in these libraries.

Math library: The libm math library provides implementations for floating point mathematical operations defined for the C standard library on Linux. For ATLAS, the most used and most costly of these functions are the trigonometric functions and exponentiation. Because the API of these functions is in the C and C++ standard, they are very widely used. To allow users to choose other implementations, other libraries have adopted the same API and provide a subset of the functionality of libm. Two of these implementations are VDT and IMF.

VDT is open source and autovectorizable and provides multiple implementations of some trigonometric and exponential functions for double and single floating point precision. The VDT library does not guarantee IEEE compliance, but with on average less than 1 least significant bit difference between libm and VDT the inaccuracy of VDT is acceptably small. The VDT functions can be inlined to avoid function calls and improve autovectorization. It promises performance improvements of a factor of 2 to 3 without vectorization and up to a factor of ten with vectorization [77]. Intel Math Library (IMF) provides a similar set of functions and is IEEE compliant. Both IMF and VDT support the libm API, making it possible to preload these libraries. VDT also offers faster versions of these functions with severely reduced accuracy, but whether they can be used would have to be decided on a case-by-case basis which disallows using them as drop-in replacement. Running tests with both libraries by preloading shows that both IMF and VDT reduce the fraction of time spent in the trigonometric functions to about 6%, see Table 7. We chose IMF due to the slight advantage over VDT. VDT additionally has the disadvantage of requiring physics validation because of its non-IEEE-compliance. If after the change hot spots still take up significant time in these math functions, VDT may offer higher gains by inlining and by removing a function call and (if the application permits) from autovectorization.

	exp	**cos**	**sin**	**Sincosf**	**atanf**
Million Calls per Event	3.4	2.5	2.2	2.1	2.1

Table 6: Trigonometric functions with the highest number of calls.

	libm	**IMF**	**VDT**
Reconstruction Time Spent in Library	14.1%	6.0%	6.4%

Table 7: Reconstruction time spent in each library. IMF and VDT were preloaded to replace libm for this test. IMF shows slightly better performance while being IEEE compliant, leading ATLAS to using IMF for production during Run 2.

Memory Allocation: ATLAS used the allocator tcmalloc 0.99 during Run 1, which was developed by Google for multithreaded applications, reflected by its name which stands

for “thread caching allocator”. It implements sophisticated locking mechanisms which were not used in the single threaded ATLAS software. Tcmalloc is also faster in single threaded applications than the glibc version distributed with SLC 5, the OS used during Run 1. This is in part due to the fact that tcmalloc does not return freed memory to the OS but manages memory internally, avoiding costly system calls. Tcmalloc 0.99 does not support aligning memory regions, such that vector operations lead to undefined behavior. Newer versions of tcmalloc were available but had been dismissed in previous tests because of slightly higher memory requirements, which presumably stems from the memory alignment implemented in newer versions. The memory allocator distributed with more recent Linux distributions such as SLC6 are about the same speed as tcmalloc. A reliably faster memory allocator could not be found among ten allocators tested [78]. Some allocators, such as jemalloc [79] were dismissed due to non-reproducible performance results. Newer versions of tcmalloc lead to a slightly faster reconstruction but require about 1% more memory because they return aligned memory unlike tcmalloc 0.99. Tcmalloc 2.1 was chosen due to the reliably good performance and its focus on threading because several projects aim at using multithreading in the ATLAS code. The slight increase in memory usage is considered acceptable due to the increased limits per job on the grid, which is now at 4GB since LS1 as opposed to 2GB before.

5.2.2 Implementation

Both libm and malloc have a well-defined API that is used by the vast majority of applications. Due to their widespread use, many alternative libraries with the same API have been implemented. Preloading them avoids changing the thousands of places where they are used. Including the preload command in the common scripts used for production jobs allows exchanging the libraries again later on easily.

5.2.3 Immediate and Future Impact

For a Run 1 event I measured 6% runtime reduction and 8% for a Run 2 event with the new math library and 2% reduction for the allocator with some types of Run 2 events. Actual impact on a job may vary as the results particularly of the allocator have shown to depend strongly on the test case. The improvements could not have come from autovectorization as the allocator is preloaded at runtime, during compile time GCC assumes stdcmalloc is used which would have led to undefined behaviour and possibly crashes with tcmalloc 0.99, as this version did not yet align memory. Therefore, it is safe to assume that autovectorization does not grant any benefits for the reconstruction without specifically conditioning the code to vectorize easily enough for the compiler to understand. Yet, with the employment of autovectorizing libraries and empowering developers to create vectorizing code, there might well be additional gain come from this change in the future.

To gain from future developments tests with other libraries can be performed by simply exchanging the preload. This is not more difficult or easier than before, but the awareness of the importance of these libraries has increased. The original tests were conducted with SLC 5. Later I conducted tests on an SLC 6 machine and the optimized post-LS1 reconstruction software, again comparing the performance of libimf with the newer version of libm. The tests showed that either library makes up just about 2.5% of the total reconstruction runtime, with negligible time spent in floating point exception handling, suggesting much of the time may have been spent due to a bug in the older version of libm in SLC 5.

5.3 Eigen library project

5.3.1 Assessment

The Class Library for High Energy Physics (CLHEP) contributed around 6.5% of the total runtime of a reconstruction job in release 17.7.2, see Table 8. CLHEP is a library specifically written to support the needs of the high-energy physics community. It has been developed and supported by the community, which has extended it with new functionality as required. The CLHEP project has been put on halt for further development, providing only bug fixes [74]. This means that the library cannot profit from newer architectures the same way a library developed with modern architectures in mind can. Additionally, CLHEP was originally written more than 20 years ago and does not use techniques developed in the meantime, such that performance is worse than that of many competing libraries. Due to the specialized application of this library, a modern replacement with similar API is not available. The highest cost of CLHEP operations comes from matrix multiplications as well as other matrix operations because these are the operations most widely used in the ATLAS reconstruction. Therefore, a linear algebra library with highly performant matrix and vector operations needs to be chosen to replace CLHEP. Three libraries claiming high efficiency and known to be in use by various scientific or industrial projects are listed below. These were considered as CLHEP replacement and their performance evaluated. In addition, two ATLAS implementations are compared:

1. Root [48] framework implementation SMatrix, which is the de-facto standard for CERN physics analysis and many other purposes
2. Intel Math Kernel Library (MKL), an Intel library for linear algebra implementing LAPACK and BLAS [80].
3. Eigen, a template library used by many scientific and engineering applications e.g. Google's TensorFlow or Google's Ceres [81].
4. An ATLAS implementation of a vectorized 4x4 matrix-matrix multiplication using SIMD intrinsics.
5. An ATLAS implementation of naïve 4x4 matrix-matrix multiplication.

The ATLAS matrix-matrix multiplication was implemented using SIMD intrinsics only for 4x4 matrices. A comparative test with all libraries shows in Figure 50 that the intrinsics implementation outperforms all other implementations and is 18 times faster than CLHEP. Eigen is second best with twelve times faster than CLHEP, because at the time Eigen did not yet support AVX. The naïve matrix multiply and Root were 5 and 4 times faster and MKL was slightly slower than CLHEP. MKL is optimized for use with large matrix sizes and uses the same codebase for large and small matrices while Eigen has a codebase optimized for each small and large matrix sizes [81]. Tests with 5x3*3x5 matrix- matrix operations showed Eigen to be almost twice as fast as Root's SMatrix with a factor of 6 over CLHEP. MKL, which is optimized for large matrices, was the only implementation slower than CLHEP for the tested matrix dimensions. None of the libraries used vectorization with these matrix dimensions, which shows how much more efficiently these operations are implemented than in CLHEP or MKL, even without exploiting SIMD instructions. Matrix and Vector sizes are mostly between 2 and 5, with very few matrix operations with larger dimensions up to 50x50 [12].
Due to the clear lead of Eigen matrix operations with respect to the other libraries, Eigen was considered for closer examination and comparison with CLHEP and the requirements of ATLAS. While some geometry operations are faster in Eigen than their CLHEP counterpart, some operations are slower by a factor of ten and more, see Figure 51, although some more common operations such as the transform are still faster. As shown in Table 8, CLHEP geometry operations only make up a fraction of CLHEP's runtime

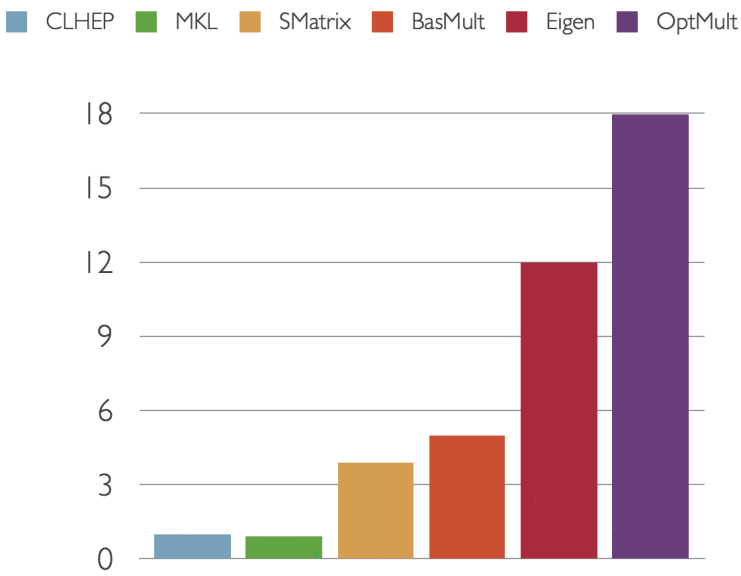


Figure 50: Comparison of 4x4 matrix multiplication of different implementations. Speedup relative to CLHEP. BasMult is a naïve matrix multiplication while OptMult uses AVX instructions. Plot from [82].

contribution. An option would be to use both libraries in parallel, each only for operations where they are the respective fastest option. Considering the awkwardness of using a different type for a geometry operation the results of which will be used in matrix operations makes this option appear less favourable. Therefore, Eigen was chosen to replace CLHEP wherever possible. Later measurements showed the slower geometry operations to be negligible.

5.3.2 Features

Eigen comes with many methods for geometrical transformations and matrix operations used in the ATLAS reconstruction. Being a template library, it can be extended easily without overhead, and provides a method to extend the functionality by own functions put in place directly with the Eigen native functions. This allowed to implement convenience functions frequently used in the ATLAS software.

5.3.3 Integration for Athena

CLHEP types and methods were used directly in many places throughout the ATLAS software. Although the API of Eigen and CLHEP are similar, it was necessary to modify all lines of code where CLHEP was used in order to make use of Eigen's functionality. In order to facilitate this effort for future changes, we designed a wrapper interface using a typedef to avoid any overhead. The wrapper function names mimic CLHEP where possible to minimize the necessary code changes and facilitate the transition for users. The wrapper also allows modifying the Eigen types (e.g. double or single precision) or to replace Eigen later or even just parts of Eigen for all of Athena in a single place, because it avoids the need to reference Eigen directly. We extended the Eigen functionality by implementing dozens of helper functions and constructors to increase the similarity between Eigen and CLHEP.

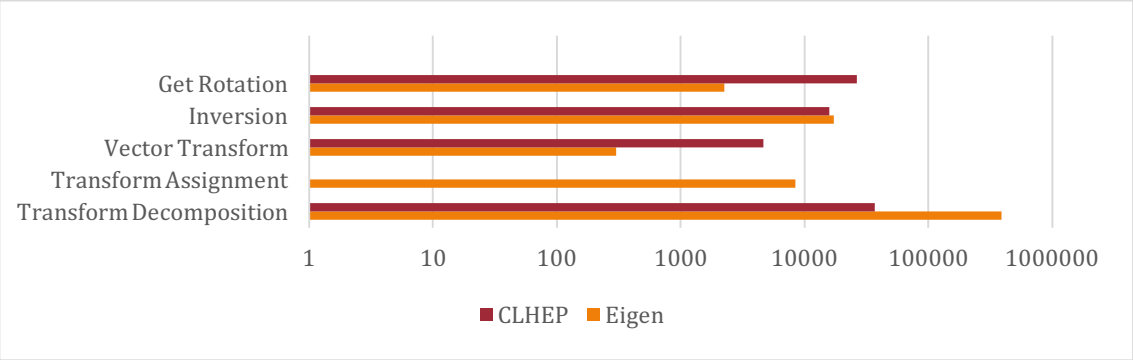


Figure 51: Comparison of some geometry operations in Eigen and CLHEP. Assigning the rotation matrix and the translation vector to the transform is a no-op in CLHEP, which is why it takes no time. In Eigen a rotation is internally not stored in matrix form.

Type	Total Runtime
Transform	1.1%
2D Vector	1%
3D Vector	1%
SymMatrix	0.9%
Rotation	0.5%
3D Point	0.4%
Matrix	0.2%

Table 8: Type of CLHEP operations and their contribution to reconstruction runtime. Point, Vector and Matrix are three distinct types in CLHEP. Data collected using gperftools.

The migration required a combined effort as more than 1000 packages needed to be updated. In the course of this large-scale change, the ATLAS reconstruction Event Data Model was also changed (see Section 5.5). This combined effort of all ATLAS reconstruction software groups took around 11 months. A caveat with CLHEP was that translation objects, point objects and vector objects among others are of different type although internally sharing the same representation. In Eigen these are all of the same type, requiring to specify operations that differ between the two types explicitly. It is therefore necessary to migrate function calls depending on the type they were used with. Another problematic change was that Eigen matrices are not initialized on construction, as opposed to CLHEP, which initializes unity matrices. Matrices that are later directly assigned some value do not need to be initialized, allowing the Eigen constructors to be faster than CLHEP constructors. Assert functions in Eigen allow monitoring access to these uninitialized matrices, avoiding bugs that would be otherwise difficult to find. The changes were first included in a migration release parallel to the normal development release so other developments would not be affected. Only after changes were tested for bugs, they were moved to the development release version 19.0.X. During the migration, I managed and coordinated the migration release, maintaining a wiki page to organize the order of packages to be changed and the persons assigned to do so. While migrating dozens of packages myself, I setup and maintained a webpage with descriptions of common pitfalls for the other developers involved in the project.

5.3.4 Immediate and Future Impact

Due to the invasive change in the software, multiple projects were conducted in parallel. While updating the code, the developers were asked to identify other unrelated ineffi-

ciencies. During the migration, other projects, developed in parallel on the development release, were also included in the release. One of the key advantages of Eigen is the inlining of functions, which saves the overhead of a function call. As a consequence, this also means inlined functions cannot be profiled separately as they do not appear in the call stack. If the speedup stayed close to what was measured in the tests presented in 5.3.1, Eigen operations now use about 1% of reconstruction time and therefore should have sped up the runtime by approximately 5%. An impression of the combined speedup of all projects during the migration can be found in the conclusions in Section 5.7. To validate the accuracy delivered by Eigen matches CLHEP I compared both by performing a translation and its inverse and comparing input and output. Results computed with Eigen showed to be equally or more accurate than CLHEP in 99.9% of all cases and in the other cases deviate from the CLHEP result by only one least significant bit.

Maintainability was improved by creating a well-defined interface that allows exchanging of the library and wrapping other library's API to fit ATLAS' use. The wrapper eliminates the need to reference Eigen directly and thereby also centrally allows changing parameters such as the precision to test impact on results and speed. Additionally, Eigen or parts of Eigen could be exchanged more easily in a potential future library change. As compilers advance in optimization techniques such as autovectorization and inlining, switching to a new compiler can bring further automatic improvements. The active Eigen community promises a modern library for the foreseeable future while its license allows ATLAS to customize the library with extended functionality or by exchanging algorithms to optimally suit the requirements.

5.4 Magnetic Field Service

5.4.1 Assessment

The ATLAS magnetic field service is used to provide access to the strength of the magnetic field across the detector. Measured field values are known for a fine-grained 3D grid of non-uniform size in cylinder coordinates. Requesting any point within the detector volume, the service loops over the map to find and retrieve the surrounding known measured values (the so-called bin) and interpolates the value at the requested point. This service is used by both simulation and reconstruction and contributed 8% to reconstruction runtime and 18% to simulation runtime with release 17.2. The magnetic field service was one of the few parts of the ATLAS code still in Fortran77. A complete rewrite of the service was necessary to not just facilitate but allow maintenance. The code had no self-explanatory variable names and no documentation; the code structure was highly conditional and contained functions with thousands of lines of code. A long callchain converted values several times between cm, mm and meter and between kilo Tesla and Gauss, requiring a measurable amount of time. A rewrite allowed reducing the call chain and unify the access that was possible through three different interfaces. An analysis of the field map, requiring 360MB of memory, showed field values were stored with higher precision than the uncertainty. Analyzing the code we discovered the same field values are requested consecutively hundreds of times during event simulation, retrieving it from memory each time. By adding a cache of the last requested bin, the speed improved dramatically.

5.4.2 Implementation

Masahiro Morii implemented a magnetic field service in C++ with a simplified and unified magnetic field interface while I analysed the physics performance and runtime efficiency in comparative tests with the old field service. I implemented a test bed for both proposes. To evaluate computational performance, the test bed requests field values

mimicking reconstruction accesses to the field service by requesting field values in 1cm steps in a straight line in an arbitrary direction from the center. Both field services cannot run at the same time, so I stored the field values and the positions and compared them requesting the same positions in a second run. For physics validation, I scanned the whole detector volume in small steps and compared them with the values from the other field service in a second run. The field map was changed to store short integers (16bit) instead of single precision floating point (32bit) values, which were found to hold the information in sufficient accuracy, halving the size in memory. An analysis showed that absolute differences do not exceed 4 Gauss or 0.2% of the field value. These differences in the two services were lower than the measurement uncertainty, such that the service can be used in production.

5.4.3 Immediate and Future Impact

The comparative tests I conducted use the new magnetic field service through the wrapper interface, such that the measured impact is less than after the full migration. This allows testing on the same release and avoids including other improvements in the measurements. The service sped up 20% in my testbed. To confirm the results from the testbed, I ran both reconstruction and simulation with Run 2 events. Simulation requires magnetic field values for points that are very closely neighboured frequently. These values are calculated by interpolating the measured values surrounding the requested point. Through the cache, 99% of the field map accesses could be avoided, requiring only the interpolation step. The time spent decreased from 18% to 2% total simulation time. Reconstruction gained 20% on the 8% spent in the magnetic field, reducing the magnetic field cost to 6.4%.

5.5 Event Data Model Update

5.5.1 Assessment

The ATLAS offline reconstruction software is a collection of several algorithms. The integration of these components relies upon a set of well-structured framework modules (described in Section 2.9) and a common Event Data Model (EDM). The EDM enables independent development of code amongst the different subgroups of the experiment, and also facilitates code re-usage and the definition and implementation of common algorithms and methods to be used in different reconstruction contexts.

It defines both transient and persistent representations for many different objects. The conversion between the two representations is done through converter classes. During Run 1, common functionality of objects within a group were bundled through virtual derivation. The persistification was object based and thereby in an “array of structures”, reflecting the object oriented transient representation. To save space, during the conversion from transient to persistent data was simplified, requiring deriving some information. Both decisions had the effect that the persistified data could not be used for analysis with the most common analysis tool Root. Instead, the persistified data was converted in an additional step to an analysis-friendly format, costing both CPU time and disk space and delayed the analysis.

One of the heavily used structures of the EDM in the ID reconstruction are the Track Parameters, see Figure 52. This group of classes defines the properties of a particle track at different detector locations. Track parameterization exists on all accessible surface classes of the detector geometry. This was realized by creating a track parameter inheritance schema analogous to the surface inheritance schema. For type safety reasons, tracks with different particle charge and different location in the detector have to be of a

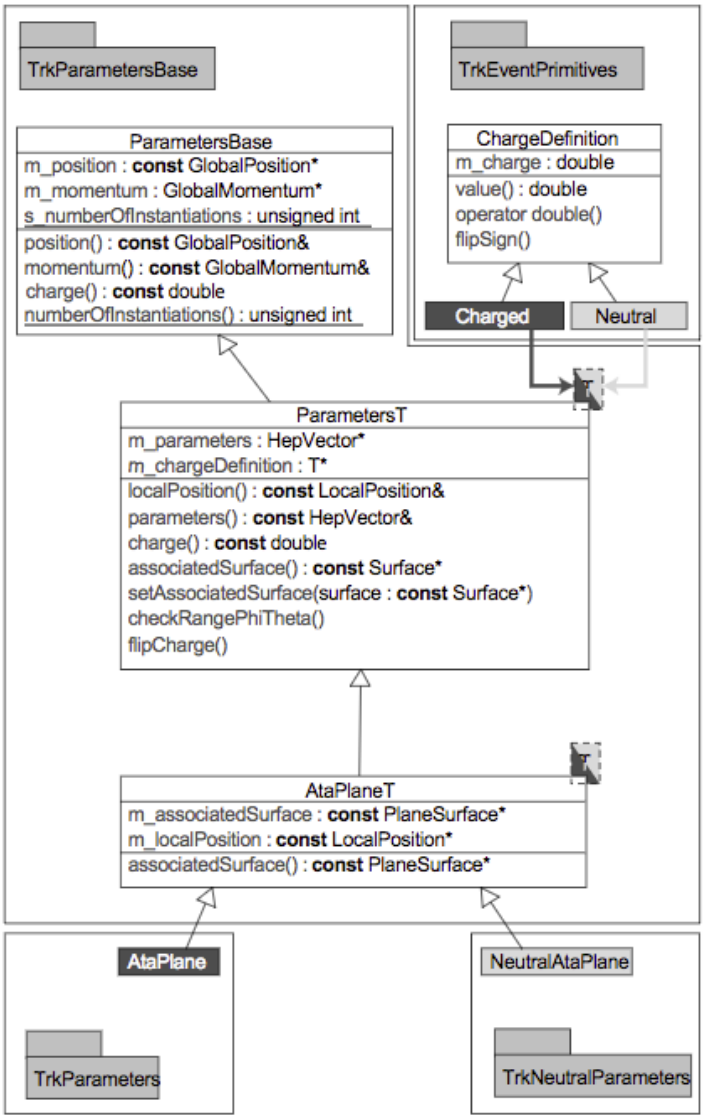


Figure 52: Charged and neutral TrackParameter as implemented in the EDM during Run 1 with the example of a plane surface. Six different surface types exist with a separate implementation each. The neutral flavor was added later on and lead to code duplication. Diagram from [83].

different class type. This led to significant amount of duplicated code. The derived class members were initialized lazily, although these values are almost always used. Lazy initialization requires dynamic memory allocation which is much slower than allocation during object creation and fragments the memory, and lastly requires a check if the member has already been instantiated before every access. When the type information was needed, a track parameter was converted from the base type to its particular surface type with a dynamic cast, which is far more expensive than e.g. an identifier comparison under typical conditions [84].

5.5.2 Implementation

A new EDM was designed with both transient and persistified representation as structure of arrays, which would minimize memory fragmentation and allow direct usage of the persistified EDM for analysis. Using the persistified EDM directly for analysis in Root means a single API can be used in both frameworks, and analysis results can be written

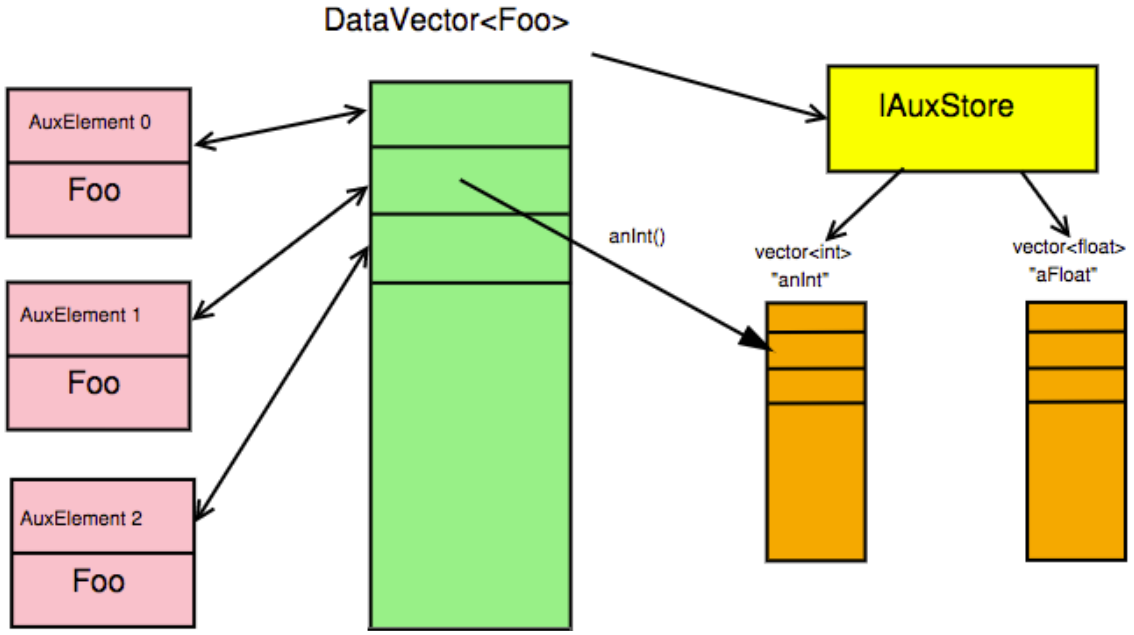


Figure 53: Access to the SOA structure through a wrapper interface which simulates an array of structures [85].

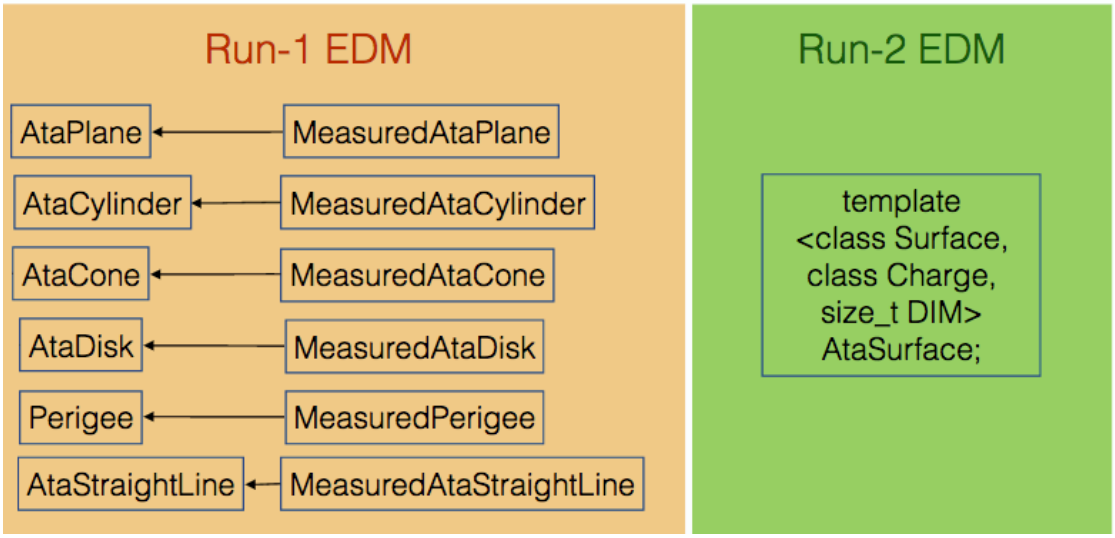


Figure 54: TrackParameters type describing the state as implemented in the new EDM. The inheritance structure remains the same as in Figure 52.

out in a compatible format. Compression algorithms are also more effective because values in a SOA structure tend to cluster value ranges, which is exploited by such algorithms. A wrapper as shown in Figure 53 allows to use this representation as if it was an array of structures in Athena, to simplify access and migration from the old model. The wrapper can be optimized out such that this form of access does not require additional resources. The decision to use structures of arrays is also intended to facilitate the use of vector instructions in the future.

The track parameters have been completely rewritten to change the complex and inefficient structure of Run 1. All parameters are now templated such that the different TrackParameters are still of different type, maintaining type safety, but use the same implementation, see Figure 54, making the definition of each separate type nothing more than a single line of code. To avoid dynamic casts to the respective type, an identifier has been introduced which allows efficient testing for the type. This allowed code size of the Track-

Parameter classes to be reduced by around 97% from 9180 to 277 lines of code. Members initialized lazily were changed to initialize on object creation. These changes were included in the design during the Eigen migration campaign in a separate project [86].

5.5.3 Impact

The exact impact on reconstruction runtime is difficult to measure due to other parallel changes. The immediate impact of the new structure is saved disk space due to avoiding the conversion step to an analysis format, but in medium to long term it also enables the use of vector units due to the representation as structure of arrays.

Standalone tests showed a factor of three computational performance improvements for the new TrackParameters [86]. Maintainability of the TrackParameter classes is improved due to the reduction of the code size.

5.6 Algorithm Reordering and Algorithmic Tracking Update

5.6.1 Seeding Improvements

While many of the algorithms used in ATLAS have been optimized to perform a particular problem as fast as possible, there are many parameters that can influence the runtime by changing the problem to be solved.

One of these problems is the ID tracking. The tracking is seeded by a direction, which stems from a triplet of measurements on three different layers, a so-called seed, see Section 2.7. A track can only be found with a good initial direction derived from the seed. The seed generation creates all combinations of measurements that fulfil certain requirements, but restricts the number of combinations per measurement. If a higher number of seeds is generated for a measurement, the seeds are sorted by weight and only the seeds with the higher weight are accepted. The weight of a seed was defined during Run 1 by the point of closest approach to the interaction region of a potential track. For Run 2, an additional factor for the weight is defined through the number of compatible seeds found that share the two measurements closest to the interaction region. Compatibility is defined as the curvature of a potential track spanned by the seeds being similar, such that they may belong to the same particle track. The sum of these two weights defines the total seed weight. This weighing scheme is more resilient to high occupancy which would lead to an explosion of combinatorics. In very high occupancy scenarios, the number of seeds created from a single space point must be limited to reduce runtime. This weighing scheme leads to a higher efficiency as real tracks are more likely to be found. Table 9 shows that seeds with at least one compatible seed have double the so-called purity, the percentage of seeds belonging to a particle track, for events with 40 pileup collisions.

Table 9 also shows how strongly the seed types differ in purity, which is exploited for another improvement: Seeds from SCT measurements are created and processed first, assuming that at least one track of each proton-proton interaction can be found with SCT seeds. After processing all SCT seeds, the interaction zone is now restricted to the region along the beam line between where the leftmost and rightmost track candidates created from SCT seeds have their point of closest approach. Seeds of other types, which contain many more fakes, must have their impact parameters within the newly restricted interaction zone. According to my measurements, this reduces the total number of seeds by 12% for a Run 2 event, reducing the runtime of the track finding by the same fraction.

A third optimization was to change the configuration of these algorithms, tightening the requirements of a track. By e.g. reducing the allowed number of missing measurements for a track, the track extrapolation can be aborted earlier such that less time is lost. Tracks with more missing measurements will not be found anymore, but measurements

show that this actually led to a higher efficiency in the reconstruction of the signal because fake tracks become less likely, see Table 10.

All three changes influence which tracks are reconstructed and therefore which tracks can be found. Proton-proton collisions which do not have particle tracks which are findable through SCT seeds are lost through the restriction of the interaction region. Because there are more Pixel seeds than SCT seeds [87] including a higher amount of fake seeds as shown in Table 9, it is more beneficial to define the region using SCT seeds because more seeds can be excluded this way. The optimization of the weights, which prefers seeds for which another compatible seed can be found, favors finding multiple seeds that correspond to a single particle. This will only exclude seeds from being created if more than a threshold of seeds has already been found for a single measurement. These improvements combined led to a 25% speedup of the ID track reconstruction according to [86]. My measurements showed an even higher total speedup of 1.54 for the reconstruction of Run 2 events.

pileup collisions	PPP	PPS	PSS	SSS	PPP+1	PPS+1	PSS+1	SSS+1
0	57%	26%	29%	66%	79%	53%	52%	86%
40	17%	6%	5%	35%	39%	8%	16%	70%

Table 9: Percentage of seeds corresponding to a particle track for different types of seeds or so-called purity. PPP seeds consist entirely of pixel detector measurements while SSS seeds consist entirely of SCT measurements. PPS and PSS seeds have measurements from both detector systems. Seeds requiring another compatible seed have a much higher purity and SSS seeds are generally more pure than PPP seeds [86].

pileup	Run 1 Cuts	Run 2 Cuts	Run 2 Cuts + Z bounds
40	0.939	0.946	0.946

Table 10: Efficiency as fraction of tracks from the signal interaction reconstructed. The new cuts increase the fraction of reconstructed tracks while the Z bounds do not have an impact on efficiency [86].

5.6.2 Backtracking Improvement

Particles produced during the collision of two protons may rapidly decay within the detector. Many particles have a lifetime so short that their existence can only be observed by their decay products. While some particles decay so quickly their decay products seem to originate from the interaction region, others travel a measurable distance but still decay within the ID. Products from particles decaying a measurable distance from the interaction region are called secondary particles. The origin of these secondary particles is less restricted than that of so-called primary particles which originate from the interaction region, making reconstructing secondary particles more difficult. To reconstruct the tracks of secondary particles, a Hough transform [88] is run on all measurements which were not yet used to construct a track from the interaction region. For each combination of measurements found with the Hough transform, the track finder tries to create a track. An analysis showed this only led to few found tracks [89], such that during for Run 2 it was decided to concentrate on electrons and positrons, because these frequently cannot be reconstructed otherwise as they lose energy due to an effect called bremsstrahlung. Electrons and positrons leave energy deposits in the electromagnetic calorimeter. By changing the order of the calorimeter and ID reconstruction algorithms, the energy deposits in the calorimeters are known before the tracking. Using these measurements, a region of interest can be defined in which to look for a secondary particle track. The TRT measurements are used to find a track pointing towards the detector center to find a direction.

This is called back tracking because the tracks are found in the reverse particle direction. Using backtracking drastically reduces the number of possible combinations and thereby its runtime [90]. This optimization was performed following the decision to allow for a small reduction of physics performance in exchange for a large improvement of the performance.

The finding of secondary particles, which was previously very expensive, was reduced to an insignificant contributor to runtime. In my measurements, it improved reconstruction runtime by a factor of 1.83, see Table 12, making this change the single largest improvement for Run 2 events, almost halving the reconstruction time. This optimization is particularly important for higher pileup scenarios, where the number of measurements that cannot be attributed to a particle track significantly increase. I describe this improvement in which I did not participate in this thesis because it shows how only a comprehensive understanding of the algorithm interaction allowed this algorithmic optimization, with an impact higher than any of the other optimizations that aimed to leave algorithms as they were. At the same time, the optimization requires the understanding which results are important and which may be dismissible and the approval of the experts concerned with physics performance.

Release / *Event type*	*17.2.7.9 32bit*	*17.2.7.9 64bit*	*17.7.0*	*18.9.50*	*19.0.1*	*19.0.X*	*19.1.1.3*
Run 1 events	11677	9569	8606	7910	6423	6201	5179
Run 2 events	84389	71845	51270	53147	37730	34322	18703

Table 11: Runtime of different releases for Run 1 and Run 2 events in ms per event. While a Run 2 event was more than 7 times slower than a Run 1 event on release 17.2.7.9, the difference is only 3.6 fold for release 19.1.1.3. The test machine is an Intel Xeon CPU L5520 at 2.27GHz with 24GB memory running SLC6.

Project / *Event type*	*32bit to 64bit*	*IMF pre-load*	*SLC 5 to SLC 6*	*New magn. field*	*Seeding optimi-zations*	*Back tracking*	*Combined theor. speedup*	*Combined actual speedup*
Run 1 events	1.22	1.06	0.96	0.96	1.17	1.19	1.66	2.25
Run 2 events	1.17	1.08	1.03	1.20	1.54	1.83	4.40	4.51

Table 12: Achieved speedup factor of different projects for Run 1 and Run 2 events. The number denotes how much faster (or slower if the number is less than 1) the software is after the respective change with Run 1 and Run 2 events. Tests with some optimizations include other optimizations which may have an effect on the results. The combined actual speedup is the difference between the first and the last column in Table 11. Tests with the exception for the second column were conducted on the same machine as the release runtime tests shown in Table 11.

5.7 Performance Optimization Results

The ATLAS reconstruction software speed was improved by a factor of more than three for Run 2 events, compared to the software at the end of Run 1. Table 11 shows the impact of the different changes on runtime. These measurements represent the state of

the software at different times during the development. It shows how each set of changes impacts reconstruction on a modern environment. The increase in runtime from release 17.7.0 to 18.9.50 for the new data set shows that improvements for one dataset can lead to worse performance in another dataset. Where possible to attribute to one or a small set of changes, this is measured and the changes are shown in Table 12.

5.7.1 How the Results were Measured

The differences between the Run 1 events and the Run 2 events as defined in Subsection 4.1.1 demonstrate how much the data has to be taken into consideration when measuring the impact of a change. Some changes have decreased the speed for Run 1 events while they have a very positive effect when processing Run 2 events. I could not conduct measurements independently of one another, as some optimizations only work with a release that already includes other optimizations, which may affect the outcome. Switching from building with 32bit to 64bit register support increases reconstruction software memory usage by 20% while runtime was decreased by the same amount. Compilation in an SLC 6 environment using GCC4.6 leads to a slight increase in runtime with Run 1 events and a slight decrease for Run 2 events compared to compilation under SLC 5 and GCC4.3. For these two tests should be stressed that the test machine was in both cases SLC 6, so also SLC 5 compiled releases were run with SLC 6. The same is true for the tests with 32bit and 64bit compiled releases, for the tests shown in Table 12 SLC 6 was used in both cases. The speedup from running on SLC 5 to SLC 6 is around 10% while the speed reduction of the SLC5 compiled binaries to SLC6 compiled binaries was only 4%, so in total reconstruction was faster also for Run 1 events after the change. The IMF preload showed only 2-3% speedup in respect to using GNU libm in later tests using SLC 6, showing many performance inefficiencies were fixed in libm since SLC 5 when the original tests had been performed. The new magnetic field was tested using a wrapper that introduced even more unit conversions instead of with the newly implemented interface, because it allowed testing on the same release without requiring code changes wherever it is accessed. Therefore, not the full impact of the change was visible, just the impact of the cache and the slightly improved access to the memory internally, leading to worse performance when running with old data. Unit tests had shown better field performance, which also becomes visible when running with Run 2 events. The impact is even higher in newer releases where the magnetic field is not called through a wrapper but directly, avoiding a deep call chain and several unit conversions. Changes in how and how often the field is called make a comparison through releases difficult though. The Eigen migration and the Event Data Model update took 11 months in total. The result is that many other improvements also went into the release at the same time. It is therefore not possible to measure these results in a single number, which is why they are not mentioned in this table. All seeding optimizations combined have had a large impact, reducing runtime by more than one third for Run 2 events. The largest single improvement stems from the backtracking update, significantly reducing the number of times the combinatorial track filter needs to run. It serves as example of the effects an algorithmic change can have and as motivation for the new tracking methods presented in Chapter 6. The backtracking update is particularly effective for expected future events, almost doubling reconstruction speed. Multiplying all measured speedups leads to a theoretical speedup factor of 1.66 for Run 1 events and 4.4 for Run 2 events, compared to the measured actual speedup between first and last release of 2.25 respectively 4.51. This is due to interactions between the different optimizations. The predicted improvement and the actual improvement differ by 26% on Run 1 events and by only 2% for Run 2 events, which is not much considering the large number of projects and the huge gains achieved. The software is in much better shape now considering the cleanup projects e.g. in the magnetic field, the EDM and with the Eigen migration that are only partially reflected by computational performance, but which have a huge impact on maintainability.

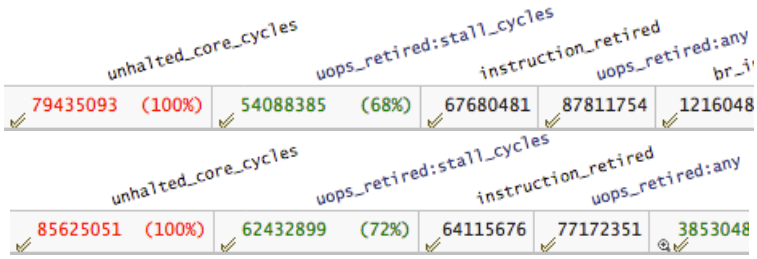


Figure 55: Core cycles and instructions retired for releases 17 (upper) and 19 (lower). The number of instructions retired per core cycle is worse for the newer release than for the older one. Nevertheless, the newer release processes more events in the same time. The number of events was chosen to achieve similar runtimes.

5.7.2 Interpretation of the Results

The results highlight the importance to test the changes with the expected data during production. It also shows a discrepancy of the effect of a change when first tested and in the last tested environment, which includes all improvements and may have changed in other unpredicted ways. An improvement can become unnecessary or even potentially harmful, as both trigonometric functions of libm and the system allocator of SLC6 deliver comparable results as the libraries they have been replaced with. The difference is that the system libraries may be updated during normal system maintenance or during a switch to a newer OS. These libraries may perform better, which would go unnoticed if new tests are not performed.

The performance of the software measured in instructions per cycle changed for the worse, as a comparison between release 17 and release 19 shows in Figure 55. This means the low-level hardware utilization has not improved. At the same time, the set goals for speed improvement were met. The improvements on the ATLAS code are algorithmic rather than improving the exploitation of hardware features, but some prepare for a better hardware utilization in the future such as the new EDM. It also indicates that the used libraries also hardly improve on the hardware utilization but rather on the algorithms used.

The xAOD format marks an important step from object-based design towards Structure of Arrays style storage of data. The chosen access patterns through a wrapper allow the use of the data in a similar way as before. The elimination of an intermediate conversion step for analysis speeds up the internal workflow and opens the door to vectorization by improving data locality. Vectorization was not achieved using vectorizing libraries due to the matrix dimensions typically used. The introduction of the Eigen library, while improving speed, did not lead to the desired vectorization, which could have served as example of outsourcing its complexity. While efforts to include vectorization in computationally expensive code regions persist, a higher priority for many groups is the preparation of the ATLAS code for parallelization through multithreading.

The optimizations with the highest impact during LS1 were algorithmic optimizations in relatively small parts of the code. While the affected algorithms consumed a lot of CPU time, changing them in some cases also affected the amount of CPU time spent in subsequent algorithms. Many of the changes affected the final result, which always requires the approval of groups concerned with the efficiency but also can only be considered by a physicist able to distinguish between important and less important results. A workflow to streamline the collaboration between physicists and computer scientists could therefore improve code quality and speed. Some deficiencies may have been spotted sooner if a clear code review scheme was in place, which would not leave single persons responsible for portions of the code. Another problem that could be addressed by a well-defined

workflow are unclear responsibilities for certain code parts, some of which were written by people who already left ATLAS. Despite the improvements to the ATLAS code during LS1, large parts of the code could profit from a thorough analysis and optimization, which is why I expect the largest potential with the smallest effort here. Parallelization through multithreading is currently being implemented. The importance of multithreading depends largely on the development of the CPU market, which is foreseen to increase the number of cores faster than memory to a point ATLAS is no longer able to utilize them just by multi-processing.

6 Analysis and Implementation of Tracking Improvements

Tracking is, as the analysis in Chapter 4 shows, one of the most expensive steps in the current implementation of the ATLAS reconstruction with a runtime increasing polynomially with number of space points. Through in-depth optimizations, performance goals could be met, but the complexity remains a problem for future medium-term requirements and will become unmanageable in the long term. Tests show that a 10fold increase in pileup leads to a 150fold increase in runtime, which is the workload expected for the HL-LHC, see [60]. One of the technical solutions to cope with the increasing computing problem is to exploit idle parallel resources. In Section 4.7 I have shown that the tracking can be efficiently parallelized. In this chapter I will explore the competitiveness of GPUs and CPUs for tracking. Alternatively, I present a low accuracy and low complexity tracking approach which has the potential to reduce the tracking time to an insignificant factor. This solution is suitable for both the expectable shift in computational resources towards highly parallel hardware and also the increasing complexity of the data. This chapter first presents the low complexity tracking approach and then the comparative study between GPU and CPU parallel tracking.

6.1 Transform Based Low Complexity Tracking

To reduce the runtime of tracking by orders of magnitude, it is necessary to reduce the complexity of the algorithms. The algorithms in place in the offline reconstruction have seen much optimization and perform well despite the increasing requirements. Nevertheless, the complexity of these algorithms remains unchanged, such that they will not remain feasible if the LHC performance increases as planned. To tackle this problem, the algorithms must be exchanged or the scope of these algorithms must be reduced.

A different tracking approach was used in the software trigger during Run 1. The software trigger has to reduce the number of events from about 100kHz to 1kHz. The decision if an event should be processed further, i.e. in case of a positive trigger decision, has to be

taken within the order of 10-50ms. To achieve such a reconstruction runtime a low complexity transform-based tracking was used. It has a slightly lower physics performance in terms of track reconstruction efficiency and accuracy, but was sufficient for trigger requirements during Run 1. To further reduce the runtime, the software trigger only reconstructs regions of interests within the detector as defined by the hardware trigger. The reason for discontinuing this approach in the trigger was the decreased performance of the vertexing algorithm for higher pileup scenarios. For Run 2 it has been replaced by an adapted version of the offline algorithms. Parts of the algorithm continue to be used to find the beamspot and a hardware implementation is planned for the High Luminosity LHC that is due to start operation in 2023. By implementing a series of improvements, I show that they can be applied in its original context as well as in other scenarios, which is presented in the following sections.

This method consists of two transform-based algorithms, for vertexing and for tracking. The tracking algorithm depends on knowing the primary vertices, i.e. the locations of the proton-proton collisions. The vertex finder is therefore critical to the approach: If a primary vertex has not been found, tracks originating from this interaction location will also not be looked for. If no other vertex is found in close vicinity, the tracks cannot be found. To make the approach usable for high pileup scenarios, I will therefore first concentrate on improving the vertex finder.

The following sections first explain the principle of the vertexing algorithm and analyze their capability to perform the required tasks before detailing implemented improvements. Subsequently, test results of the implementation are presented and lastly possible application scenarios of the algorithm are discussed.

6.1.1 General description of the vertex finder

The vertex finder tries to establish the vertex locations by finding points in the interaction region where potential tracks point to. A potential track is defined as the combination of two space points on two distinct pixel layers that are compatible with a helix of a particle originating in the interaction region. The minimum transverse momentum can be set. This means a helix with a certain minimum radius must be found that passes through both space points and any point in the interaction region, assuming a particle path under ideal conditions. There is at maximum one helix that can be found fulfilling the requirements. The locations in the interaction region found this way are recorded using a histogram with binning along the beam axis. The bin with the highest number of entries is subsequently returned as the most probable primary vertex of the signal interaction.

This algorithm has a complexity of $O(n^2)$ with the number of space points, because it combines two pixel space points. To combine only space points which are likely to be compatible, the space points are sorted by φ region and the delta-φ of the two space point is restricted. This way, combinations which are only compatible with transverse momentum lower than the threshold are avoided.

This approach has a high noise level as random combinations of two space points often point to the beam spot, leading to a high background sometimes indistinguishable from interactions. This becomes worse with higher number of pileup interactions.

To prove it is possible to distinguish the signal vertex from pileup interactions in high pileup scenarios using only tracks, I analyzed event topologies showing the theoretical distinguishability of certain signal events from pileup, presented in 6.1.2. The analysis is discussed in full detail in [91]. I implemented an improved version of the algorithm which is adapted to exploit the Run 2 detector geometry and reduces random space point combinations that do not belong to a particle track, so called fakes. This implementation shows the applicability and limitations of the approach as well as the physics- and computational performance in comparison with the offline tracking.

6.1.2 Distinguishability study

The high-level-trigger does not reconstruct tracks in all detector regions but instead focuses on regions of interest which are suspected to contain interesting physics objects. To reduce time even further, the vertexing algorithm was used to identify the signal interaction such that only one vertex would have to be reconstructed, significantly reducing the problem size. To be able to identify the signal vertex, it has to be distinguishable from other vertices with the information available. To see if the distinction is theoretically possible using only tracks visible in the Inner Detector, I analyzed Monte Carlo simulated events. These events contain information about the generated particles such as type, energy and charge, the so-called truth information which is not available in events read out from the detector. Using this information, I extracted only particles that are likely to leave a trace in the ID, i.e. charged particles that are unlikely to decay before they reach the outer regions of the ID. This does not mean that all of the remaining particles will leave a reconstructable trace in the detector, as every detector has inefficiencies such as non-working elements, coverage of the phase space only reaching up to a maximum pseudorapidity, and particles may interact before they reach the outermost regions of the ID. Nonetheless, using all charged and stable particles allows judging what distinction rate is possible under perfect conditions, and gives an upper boundary for the maximum efficiency of this algorithm. The efficiency here denotes the ability to find and distinguish a vertex. In a second step this upper boundary is compared with the distinction rates achieved with an implementation of the algorithm using the ATLAS Run 2 ID geometry and increasingly realistic scenarios.

6.1.2.1 Theoretical Distinguishability

Because it was previously not possible to reach the high energy now possible with the LHC, events with high collision energy, or high q^2, are the least investigated events. This is why the signals for which ATLAS is configured to trigger are associated with high q^2 processes. During such processes, high-energy particles are produced that quickly decay into one or more particles with a lifetime long enough to pass through the detector. For many of these events, high-energy tracks can be found, as the most common products of decays resulting in a particle with a longer lifetime are charged. The ID can only detect charged particles. Besides particles from the signal event, usually more particles result from a proton-proton interaction. These particles range from very low energy to higher energies, with the signal in the vast majority of the cases having the highest energy.

The ATLAS software is not configured to find particle tracks in the ID below 400MeV/s, which does not pose a problem because low energy particles contribute little to the information about the event, such as the total event energy. There are several reasons for excluding these tracks from reconstruction. One effect affecting detection is scattering, which can occur when a particle interacts with material inside the detector, making it change its path. The average scattering angle is inversely correlated with the momentum, such that these lower energy particles are more likely to deviate strongly from a helical path, see [92], and therefore have to be searched for in a much wider road, increasing computational cost and number of fakes. Another issue with low-energy particles is that their curvature in a homogeneous magnetic field is inversely correlated with their transverse momentum, such that helices of low energy particle tracks will have a smaller radius. A smaller radius means a particle track is covering a much larger detector region in φ around the beam axis, because the curvature is in φ. This geometrical spread is why the reconstruction of particles of a low energy range requires considering many more combinations, significantly increasing the computational cost. The vertex finder assumes nearly helical track paths and has no means of correcting for deviations, so that only particles with a transverse momentum greater 1GeV/c are taken into consideration for the distinguishability. For the test I filtered out particles, selecting only particles that are not likely to decay before reaching the outer regions of the ID (hereafter referred to as “stable

particles") and with a charge unequal 0. Also, the particle's origin had to be less than 0.2mm from the interaction point. A distinction metric has been created considering different characteristics of signal and pileup interactions. The particle energy has shown to allow a good distinction rate because the signal is high energy and a signal interaction usually produces other medium- to high-energy particles. Therefore, the transverse momentum (p_T) of a particle, which is the only energy measurement available in the ID, is used. To increase the effect of high-energy particles, the p_T measurement is squared. To avoid registering overlay of multiple low-energy pileup events as a single high-energy collision, the total measurement is divided by the number of tracks. Testing this metric against multiple other metrics, it has shown to deliver the best distinction rate for different types of signal events. This metric M is described by the sum of the square of the transverse momentum of all tracks divided by the number of tracks t:

$$M = \sum_{i=1}^{t} \frac{{p_{Ti}}^2}{t}$$

To take into account that high p_T pileup interactions may be rated similarly to a signal interaction with metric M, the three highest rated results are used. This means for a tracking algorithm that all three results are treated as vertices that have to be reconstructed. This increases the runtime of the tracking by a factor of three, but because tracking is orders of magnitude faster if only tracks originating from a certain point have to be reconstructed, the total runtime will still be lower. Applying the metric above lead to the distinction rate in Table 13. Tested events have been chosen to cover different types of different signatures in the ID. The signal in a Higgs to two photons decay does not leave a trace in the Inner Detector and has been chosen to show limitations of the approach. Using the low-level trigger information, it is possible to select this algorithm only for events for which a well distinguishable signal is expected.

Pileup	$t\bar{t}$ allhad	$t\bar{t}$ nonallhad	Zmumu	Hγγ
20	100%	100%	99.9%	75.7%
40	100%	100%	99.7%	63.6%
80	99.9%	99.9%	99.6%	52.8%
160	99.9%	99.7%	99.1%	43.7%

Table 13: Truth analysis for charged particles p_T >1GeV/c. Probability to successfully distinguish the different signal vertices from pileup interactions, i.e. to be among the three top rated vertices with the given metric.

As expected, the distinction rate is low for the Higgs to two photons decay (Hγγ), making this metric inapplicable for this type of event. The study shows that for other tested event types the distinction is possible with a high probability even in high pileup scenarios.

6.1.3 Algorithmic Details of the Vertexing Algorithm

The study suggests that it is possible to distinguish event topologies from one another using only charged, stable primary particles under ideal conditions. In a tracking detector, such as the ID, the tracks have to be found first before such information is available. The existing algorithm uses combinations of two pixel space points to calculate the particle energy and origin. While this is enough to find all tracks originating from the signal, it will also generate many fakes. The algorithm has been implemented from scratch to include the endcaps, account for changed Run 2 geometry and implements improvements for physics performance and speed.

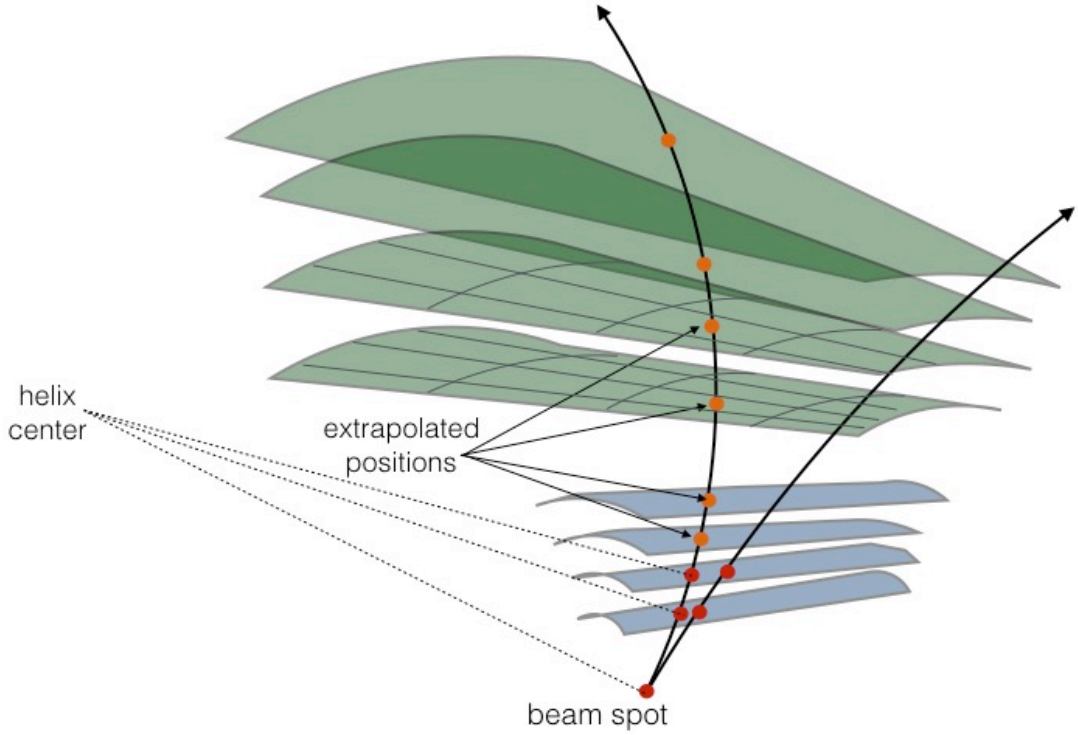


Figure 56: Schematic of how detector layers are binned in z and φ. The path of a potential track is calculated using two space points and the beam spot. If sufficient space points are found, a track pointing to the beam spot is considered found.

6.1.3.1 Principle of the Vertexing Algorithm

The algorithm uses combinations of two pixel detector space points to determine the origin of a possible track. It extrapolates a straight line through both space points in the r-z plane towards the interaction region. Here, r denotes the distance perpendicular to the interaction region, i.e. the radius, and z, the coordinate along the interaction region, with z = 0 being in the center of the interaction region. The φ distance, i.e. the difference in the angle around the beam axis and the distance in r between the two space points is used to calculate the transverse momentum of a particle originating from the interaction region. The vertex histogram bin corresponding with the location on the interaction region is updated according to the metric M. The algorithm exploits that ATLAS has a nearly homogeneous magnetic field in the ID and that the center of the magnetic field, the center of the ID and the interaction region are all in the same place. The tracks of charged particles originating from the center are therefore only deflected in φ direction, so they are nearly on a straight line in the r-z plane for higher p_T. All combinations of two Pixel detector space points that have a φ distance compatible with the tested p_T range (i.e. here above 1GeV/c) are considered. To quickly find all compatible combinations, these space points are binned in φ.

As mentioned, a major problem for the applicability of this algorithm were the fakes, that added up to a lot of background, as many space point combinations with a φ distance within the analyzed p_T range do not belong to a track. To effectively check if a combination belongs to a track, the existence of additional space points on a helical trajectory can be tested. To minimize the cost for these tests, all space points from the ATLAS silicon layers, the SCT and the Pixel detector, are binned in a 3D histogram by φ and z for barrel space points respectively in φ and r for space points on the endcaps. Bin sizes must be adjusted for a trade-off between noise elimination and allowing for deviation from the ideal assumed conditions. The radius for the barrels respectively the z coordinate for the endcaps is known because the subsequent layer can be calculated using the trajectory. With the

next layer known and the angle and curvature of the track calculated from the two initial pixel space points, it is possible to accurately calculate the coordinates on the next detector surface, see Figure 56.

This way, a check for the existence of a compatible space point completely avoids scanning through space points to find a match. For tracks with a p_T above 1 GeV/c, the locations can be approximated with a linear extrapolation. At these energies, the radius of a helix formed by a particle's track are so large that a linear extrapolation leads to insignificant errors. The maximum error is larger for lower energies and smaller angle to the beam axis. The maximum absolute error of the approximation for φ is calculated in Appendix A with 0.04° for the ATLAS detector. This linear extrapolation is faster than the accurate formula for a helix. By setting a single bit corresponding to the layer in the histogram for each space point, the only additional check is a single bit test for each detector layer. By requiring at least six compatible space points, the probability for noise forming a helical track is reduced. The ATLAS Run 2 geometry is designed to generate at least eight space points for a track within the boundaries, so this still allows for some inefficiency. Each time for two pixel space points sufficient compatible entries are found, the z location on the beam is stored in a vertex histogram together with the energy of this track. After looping over all feasible space point combinations, energies and number of tracks are used to calculate the metric to score each bin. The three highest rated bins are returned and assumed to contain the primary vertex of the signal interaction.

6.1.3.2 Further Improvements to the Algorithm

Prediction of the potential space point location on a subsequent layer from pixel space point combinations is limited by a combination of multiple scattering, the distance between these layers and the deviation from a homogeneous magnetic field. Such inaccuracies can lead to missing the bin in the histogram for an SCT detector layer by one, especially when an SCT space point would lie close to the border between two bins. To account for these inaccuracies, a compatible space point is considered found if there is an entry in the calculated bin or in any of its eight neighboring bins. My tests showed this improves finding compatible bins without significantly adding to noise. Noise levels are low for even high pileup scenarios because of the high resolution of the z-φ (for the barrel) respectively r-φ histogram (for the endcaps) which has in the order of 10^6 bins but only about 10^3 – 10^4 space points per detector layer.

Another inaccuracy related to finding the compatible SCT space points is that the barrel is not a perfect cylinder but consists of flat overlapping modules arranged around the center. Similarly, endcaps are overlapping to avoid cracks where particles could otherwise pass undetected. This leads to differences in r of 4cm between space points on the innermost respectively on the outermost position of a module for the barrel and similar differences for the overlapping endcap modules. A lookup in the histogram based on an r (barrel) respectively z value (endcaps) with a 2cm error can lead to a wrong lookup. This is improved by shifting each space point towards the average r of the barrel respectively average z value in the endcaps in the direction of the detector center. This is not accurate for all tracks as they can originate from the whole interaction zone which is $\pm$156mm from the detector center. The correction direction deviates from the actual track direction most in the central barrel regions, as the distance a particle travelled is shortest here. The measure is nonetheless helpful as the correction in z is minimal in barrel areas close to the barrel center. The φ direction cannot be corrected in the same way, as it depends on a particle's charge and transverse momentum, and is therefore left unchanged.

The transverse momentum resolution is not very precise for this algorithm. The algorithm depends on the beamspot's location being at r = 0 and cannot adjust vertex positions in r and φ direction. If the vertex is not exactly on the beamspot, all tracks originating from this vertex will have an error in the assigned p_T. The beamspot's location is usually known, so the effect can be reduced by shifting all space points by the distance of the beamspot

from the center of ATLAS. Although the locations of interactions can deviate by a few hundred microns from the known beamspot, this will nonetheless reduce the error of the tracks. By doing this before the previously described correction for the flat overlapping modules, the space point locations will be corrected for their perpendicular distance.

Second, a random combination of measurements is as likely to result in a very high p_T fake track as in a low p_T fake track. High p_T contribute much more to the total score using the metric introduced in 6.1.2.1, such that a single very high rated fake has the potential to create the highest rated vertex. To avoid this while still allowing real high p_T tracks to contribute more to a vertex than a random combination, I introduced a maximum p_T of 30, such that a single fake which is counted just once will not contribute much more than a high p_T track from an actual particle. A real track is more likely to be counted up to 10 times from different Pixel measurement combinations.

6.1.4 Performance Study of a Vertexing Implementation

6.1.4.1 Single Particle Study with Realistic Geometry

A second study has been conducted to estimate the effect of geometry on the performance of the algorithm and the metric. The events tested in the truth study are simulated in the Run 2 ATLAS detector geometry, though with ideally homogeneous magnetic field and no material interaction effects, such that the path of the particles does not change during lifetime. To measure the difference between lost efficiency due to the detector geometry, which cannot be corrected, and lost efficiency due to simplifying assumptions in the algorithm, minimum requirements to find a track are defined. These are measured on the truth information stored with simulated particles. A particle that does not pass through at least 6 distinct layers or has a pseudorapidity $|\eta| > 2.5$ is considered to be unfindable. The particles marked unfindable under these conditions make up 45% of all charged, stable particles with transverse momentum greater 1GeV/c. Under these conditions, the algorithm finds 99% of the remaining findable tracks.

6.1.4.2 Single Particle Study with Realistic Setting

In this subsection, the algorithm's ability to find single particle tracks in a realistic setting is tested. Magnetic field and geometry are both simulated in full detail. The particles do not originate from the center of the detector as in previous tests, but may originate anywhere along the interaction region. Material interaction is also simulated. Realistic particle behavior may render a track unfindable for more reasons, such that I extend the definition in 6.1.4: If a particle decays before passing through at least 6 distinct detector layers, it will be considered unfindable even if the decay products create further space points. A particle is only considered findable if the particle's creation vertex distance in r from the interaction region is less than 2mm. The interaction region spans from (0,0,−156) to (0,0,156), so 31.2cm along the beam axis. A particle is considered found if at least 6 bins containing measurements from this particle have been grouped by the algorithm. Using this configuration, the algorithm can find 94.8% of all findable particle tracks with $p_T >$ 1GeV/c. This is a lower bound, as in an event with many particles the space points caused by other particles may contribute to finding through random combination of a space point with space points belonging to the track. Including the measurements from many other tracks does not reduce the probability to correctly combine measurements, instead it introduces additional random combinations of unrelated measurements, so-called fakes. Therefore, all tracks found during this test would also be found running this algorithm on an event with many particles.

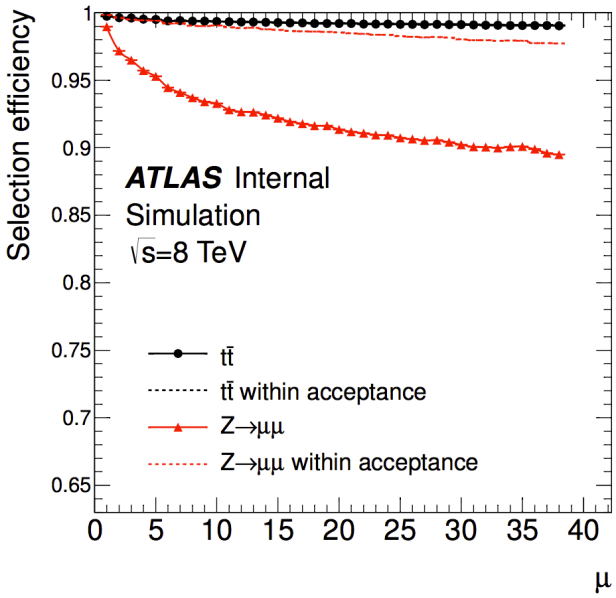


Figure 57: Vertex identification efficiency for lower pileup events with the Run 1 vertexing algorithms. Within acceptance denotes that the signal is fully detectable in the Inner Detector. Plot taken from [93].

6.1.4.3 Study with Realistic Setting on Run 2 Events

To compare the number of real measurements from the same particle combined correctly with the number of fakes, I created events with identifiers for each particle above an energy threshold of 0.3GeV/c. Particles with lower energy are not found by the algorithm, which is configured to find tracks above 1GeV/c transverse momentum. For tests with 500 events with 40 pileup collisions and a Z to two muons signal, on average 1835 tracks with more than 1GeV are found per event, but 390 of these tracks are fakes. This corresponds to 21.2% fakes. The number of found tracks does not correspond to distinct tracks because the same track can be found and counted up to 6 times in the barrel region and up to 10 times in the endcap region, by combining different Pixel detector space points of the same particle. My tests show tracks are found 6.5 times on average. A fake has a very low probability to be counted as often because measurements from different particles are less likely to have a high level of agreement between the different space points, such that not all combinations point in the same direction. Therefore, these fakes are more likely to be distributed over the interaction region rather than accumulate on one point. The primary vertex can be correctly identified in 74.6% of the cases. The huge difference to the theoretical measurements partially stems from the 45% lost tracks described in 6.1.4. Accordingly, in 45.6% of the tested 500 Z to two muons signal events, there were not at least two muons with $p_T > 15$GeV findable. In another 1.2% the interaction took place outside the interaction region in z direction defined to be between −156mm and +156mm. When disregarding these events with (partially) unfindable signal, 93.0% of all vertices are correctly identified. A study of the offline reconstruction vertexing algorithms during Run 1 shows that around 90% of the signal vertices are correctly identified in Z to two muon decay events when events with partially unfindable signal are also considered, and in about 99% of the cases if these events are factored out, see Figure 57. Although these results are only for up to 38 pileup interactions, the trend is recognizable such that similar results are expected for events with 40 pileup interactions. The better result of the offline reconstruction algorithm despite the same metric is because these vertexing algorithms run after the detailed track reconstruction, such that very accurate p_T measurements can be used, the exact location of the tracks' perigee is known up to a few microns and the fake rate is below 1% [93]. The additional low p_T tracks should, according to the metric, not significantly influence the result, as Table 13 in Section 6.1.2.1 shows.

6.1.5 Applicability Analysis of the Algorithm for Tracking

The presented vertexing Algorithm's probability to successfully select the signal vertex is crucial to the tracking algorithm that requires the vertices. If the signal vertex has been found but its identification cannot be guaranteed, more vertices would have to be reconstructed. The runtime of the tracking algorithm is linear with the space points and linear with the number of vertices to be reconstructed. The number of reconstructed real and fake vertices increases with pileup, such that the runtime becomes O(n*m) for n space points and m vertices. This would quickly get unacceptably high. To avoid reliance on the found vertices, the vertex finder algorithm itself can be used to group space points belonging to tracks, as it works on a similar principle as the track finder. By using the vertex reconstruction algorithm to find tracks, all tracks that could have been found with the tracking algorithms can be found as well. I will shortly explain the tracking algorithm to show that the vertexing algorithm can find at least the same amount of particle tracks:

The track finder algorithm does not create tracks, but groups space points that may be compatible with a track. The actual track construction and fitting must be done in a subsequent step. This subsequent step will be much faster than with standard seed finding, because it eliminates more fakes and already provides all possibly compatible space points, see analyses in Subsections 6.1.6 and 6.1.7. To create the groups, the algorithm maps all space points to a 3D histogram. Space points on the same helical path are assumed to belong to a single particle. Accordingly, space points on a helix are mapped to the same bin. This mapping only works for a specified curvature corresponding to a certain p_T and for a specified origin on the beam axis. This is why the mapping has to be done for all desired p_T ranges and all vertices to be reconstructed. To identify if a space-point belongs to a track, it is tested if it maps to a bin with sufficient entries from distinct layers according to a threshold. Entries in neighboring bins are also considered and added to the group. If such a group contains sufficient space points from distinct layers, they are assumed to belong to one or multiple very close tracks. The size of the bins defines the accepted p_T range, and counting neighboring bins ensures space points mapped on bin borders cannot prevent space points from being correctly grouped.

The vertex finder uses the same function to find space points compatible with a two space point combination as the tracking algorithm for its binning. But the vertex algorithm does so using the originating point and p_T calculated from these particular two space points. In doing so, it allows a higher deviation from a perfectly helical path with the same bin sizes and does not suffer from the uncertainty of a reconstructed vertex. The tracking algorithm's existence is legitimized by its linear complexity for reconstructing a single vertex, making this algorithm applicable if only a small number of vertices needs reconstruction and if they can be reliably found at low cost. With the scenario of requiring reconstruction of particles from all vertices, this condition does not apply. The vertexing algorithm already tests pixel space point duplets for compatible space points on the other layers, independent of their vertex location on the interaction region. By including particles with p_T between 0.4 and 1 GeV/c, the vertexing algorithm can emulate the full functionality of the tracking algorithm. In a test with realistic events I established the algorithm's ability to find lower p_T tracks, see Table 14.

	$p_T \geq 0.4$	$p_T \geq 0.5$	$p_T \geq 0.6$	$p_T \geq 0.7$	$p_T \geq 0.8$	$p_T \geq 0.9$	$p_T \geq 1.0$
truth-tracks found	82.2%	87.5%	89.6%	91.0%	93%	94.3%	94.8%

Table 14: The vertexing algorithm's efficiency for different p_T. For the tests, single tracks from a minimum bias event were selected.

While the efficiency is significantly lower for low-p_T tracks than for higher p_T tracks, these lower p_T tracks also contribute less to finding high energy particles. Overall efficiency is lower than offline reconstruction, but it is also much faster.

6.1.6 Computational Performance Comparison

The output of these algorithms are not tracks but rather groups of space points that may belong to a track. To create a track, one or multiple tracks must be fitted through these space points. This is a similar problem to trying to find a track from only a set of initial parameters given by a seed, as currently in use in offline track reconstruction, but avoids having to find the space points on each layer. The track reconstruction, as mentioned in Section 2.7, consists of three steps. The intermediate results are seeds, track candidates and tracks. With each subsequent step, the number of results is reduced and each step creates a structure closer to the final track. The effort required to create a final track from the filtering algorithm's output is somewhere in between the effort needed to create a track from a seed or a track candidate. For a reasonable comparison I will give the results of both seeding and the tracking part to contrast the results of the filtering algorithm.

The results of the speed comparisons are in Table 15. To measure track candidate creation and seeding, I ran a full offline reconstruction. Because the track candidate creation requires the seeding to run, I measured their combined runtime. In a separate run, I disabled the track candidate creation and measured only the seeding. As some seeds are excluded using the results from tracking, which does not take place when the tracking is disabled, the test of the seeding overestimates the actual time by a few percent. The results show that the algorithm scales similarly as the offline seeding and tracking algorithm with the cut on transverse momentum. The seeding does not gain much speed from changing the p_T cuts. The filtering algorithm is between 7.8 and 17.5 times faster than the seeding.

	$p_T \geq 0.4$	$p_T \geq 0.5$	$p_T \geq 0.6$	$p_T \geq 0.7$	$p_T \geq 0.8$	$p_T \geq 0.9$	$p_T \geq 1.0$
Seeding + Track Candidates Creation	3612ms	3040ms	2642ms	2284ms	2029ms	1891ms	1758
Seeding only	945ms	940ms	914ms	902ms	898ms	895ms	893ms
Filtering algorithm	120ms	98ms	84ms	74ms	64ms	57ms	51ms

Table 15: Timing of different tracking-related algorithms per event. The filtering algorithm has not been optimized for speed yet. Tests performed with Z to μμ signal events.

The vertexing algorithm cannot assert the quality of the found space point combinations because it is missing too much information. The distance of particle production vertices from the beam spot is unknown, and for each space point found in the histogram that is compatible with a duplet, the location is only known within the histogram bin size boundaries. This lack of information means that no meaningful bookkeeping that would allow to reduce complexity is possible, neither would a bookkeeping mechanism save much time due to the low complexity of the algorithm. Instead, the lack of bookkeeping allows a parallelization over all space point combinations. Parallel instances can use the same lookup histogram and space point lists, as no there are no writing operations after initialization of these lists, such that memory overhead is minimal. Computational overhead is also negligible as it consists only in the assignment of space points for each parallel instance. These properties make the implementation of a parallel version comparably trivial.

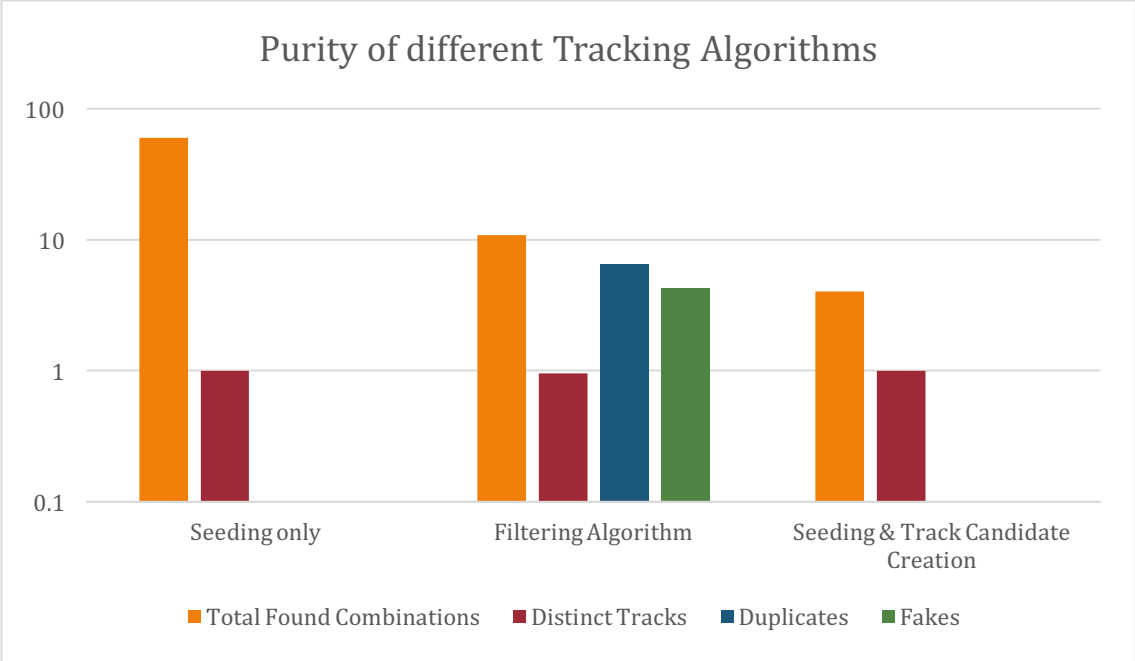


Figure 58: Comparing the purity of the found combinations of different algorithms. While I generated detailed data about duplicates and fakes for the vertexing algorithm, this information is not available for the other algorithms. Tested with Z to μμ signal events with 40 pileup interactions and transverse momentum > 1GeV/c.

6.1.7 Purity Study

This subsection presents the results of a study of the number of found tracks compared with the number of found fakes and the number of duplicates found. A duplicate refers to a collection of space points created by a particle, i.e. belonging to a truth track, which has been found multiple times. Excluding such duplicates is not always possible a priori because two groups of space points may share most but not all space points, such that both groups include space points from the same track. The same problem exists in offline tracking, where a so-called ambiguity solver decides which is the more probable track. The number of duplicates and fakes found influence the runtime of subsequent tracking algorithms which need to filter out all fakes and duplicates. As explained in 6.1.5, the results are applicable to both tracking and vertexing algorithm, such that I will not distinguish between the two. If the algorithm is used for vertexing, the number of fakes constitute the background, creating fake vertices and making a distinction of vertices more difficult.

To allow for a comparison with the offline reconstruction, this study compares the filtering purity and duplicates generated to the seeding and the track candidates in the offline track reconstruction.

As the chart in Figure 58 shows, the seeding finds 60 combinations per real track and therefore almost 6 times more than the 10.8 found combinations per found particle track from the filtering algorithm. The track candidate creation reduces the number of combinations to 4.3 combinations per real particle track. This comparison assumes a 100% efficiency for all algorithms except the filtering algorithm for which the measured 95% efficiency is used. This data visualizes how much the size and complexity of the tracking problem can be reduced by using the filtering algorithm. The number of combinations of the filtering algorithm is 6 times lower than the number combinations generated by the seeding. With the provided information a track candidate can in the worst case be created in a similar time as the track candidate creation from seeds. This means that in the worst

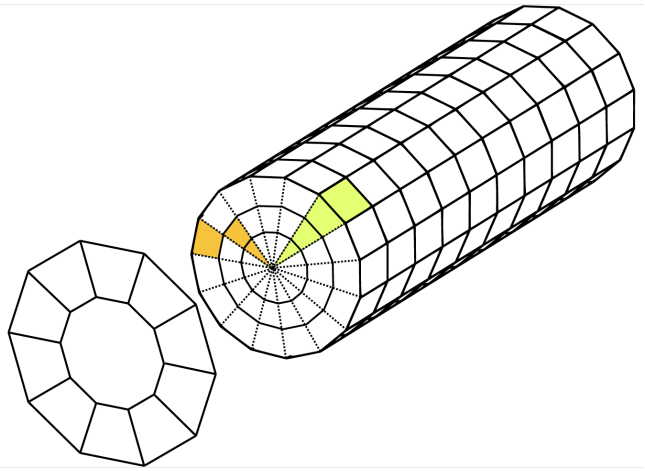


Figure 59: Sketch of ATLAS Inner Detector subdivided into regions [94]. Two independent combinations of regions have been highlighted. For the yellow combination, all layers are from one φ region. The region in orange is a combination of the two innermost layers from one φ region and the outermost layer from the neighboring φ region counter clockwise. In total, there are 5 different combination: All layers belong to one φ region, combining the two innermost layers from one φ region with the outermost layer from a neighboring region, and combining the innermost layer from one φ region with the two outermost layers from a neighboring φ region.

case, the track candidate creation using results from the filtering algorithm should take only $1/6^{th}$ of the time it takes in combination with the offline reconstruction seeding algorithm.

6.1.8 Proposal for Application for the Improved Vertex Finder

The improved vertexing has different possible applications. The algorithm in its original form was, is, or is foreseen to be used for some of the applications. These are the software trigger vertexing and tracking in the past, the vertex finding for the beam location currently and a hardware implementation for the hardware trigger for the high-luminosity LHC in the future. In the case of current and future developments, the improved vertexing algorithm can be adapted to the purpose and can be directly applied. In the cases where the original vertex finder and track finder have been replaced, the performance of the solution currently in place needs to be compared for differences in runtime and physics performance with the improved vertex finder presented here.

As another field of application, the vertex finder could be used in the offline tracking, as low accuracy tracking algorithm to reduce the complexity of the reconstruction in a high pileup scenario. The events generated with the High Luminosity LHC starting 2023 will have much higher pileup which will lead to prohibitively expensive reconstruction. A way to tackle the complexity would be to run detailed reconstruct only in the region of interest where the signal is located and run reconstruction in other detector regions with a faster algorithm allowing lower accuracy. Particles originating from pileup interactions are reconstructed mostly to be able to distinguish the energy deposited in calorimeters by these particles from the energy deposited by particles from the signal interaction. If the accuracy of the pileup reconstruction is reduced, it will have a minor impact on the accuracy of the attribution of energy deposits in the calorimeters. This may be acceptable if the loss of accuracy can be compensated for with a higher number of events which can be processed, which can in turn improve confidence levels. To reconstruct the pileup differently from the signal, the signal origin location must be known before reconstruction. As the tests presented in 6.1.4.3 shows, the vertexing algorithm does not find the primary

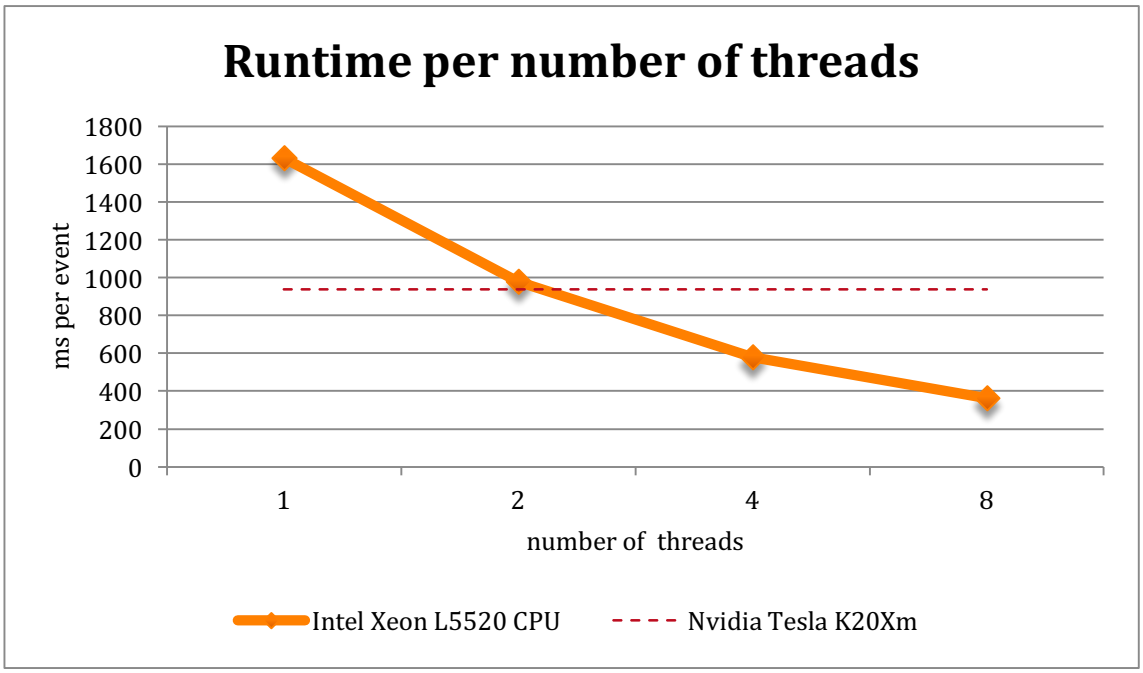


Figure 60: Runtime of the tracking per ttbar event with 10-pileup interactions on a Xeon L5520 CPU with different number of threads. The dashed line denotes performance of the same algorithm executed on a GPU. The CPU version is derived from the GPU version by [57].

vertex position among all pileup vertices. However, for many event types the signal event origin location can be pinpointed up to few centimeters using trigger information from all detector regions. Using this location, only this area can be reconstructed using high accuracy reconstruction while the pileup interactions in the other areas can be reconstructed using the improved vertexing algorithm.

6.2 GPU Parallel tracking on CPUs

Prototypes running tracking in the ATLAS ID on a highly parallel GPU have been developed as early as 2011 [57]. For this approach, the seeding and the tracking step are both parallelized. The seeding is parallelized over parallel regions, how the detector is divided into regions is shown in Figure 59. To improve memory locality, space points are sorted by detector region. After all seeds are created, track candidates are formed. This step is parallelized over the seeds, such that a very high degree of parallelism equal to the number of seeds can be achieved. Algorithmically it works very similar to the currently used tracking approach using a Kalman filter for fitting but is missing the bookkeeping which would make parallelization more complicated. Another GPU prototype parallelizes in a similar fashion but uses a different algorithm for the tracking. In [95], space points are combined using cellular automaton algorithms. Recent studies rely on these prototypes [96]. Because GPUs are not available on all computing sites, but multicore processors are, I wanted to study the performance that can be achieved with this concept using threading on CPUs. This would allow deploying these algorithms to run on the currently available resources. A study under my supervision showed that the algorithm presented in [57] is also efficiently parallelizable on CPUs. The GPU version and the version modified to run on CPUs produce the same result. Due to the prototype character of the parallel versions with significant physics performance deficiencies, a meaningful direct comparison to the current production algorithms for tracking is not possible. It also prevented running tracking for events with more than 10 pileup interactions because of a complexity explo-

sion. For a 20 pileup Run 1 event, the CPU version ran out of memory on the dedicated test machine with 24GB memory and the GPU version could not be started. To compare both the parallel CPU and the GPU version, we ran tests on a high-end Nvidia GPU for scientific computing, the Tesla K20Xm and compared it to results from an Intel Xeon L5520 CPU. Performance comparisons showed that the CPU version outperforms the GPU implementations for more than two threads, see Figure 60.

My other studies presented in 4.6 suggest removing roadblocks for parallelizing the tracking only adds few percent of extra work to be efficiently parallelizable. This implementation did not include all features and improvements of the highly optimized sequential algorithm, but serves as an example of how efficient parallelization can be. The elimination of bookkeeping has shown to increase the problem size by less than 10% for higher pileup scenarios. Reintroducing bookkeeping may reduce complexity but poses additional challenges to parallelizing this problem. If bookkeeping is introduced over regions, the problem may not be suitable for GPUs anymore due to the reduced achievable parallelism. There are, assuming a minimum transverse momentum of 400MeV/c, the bending in φ is 22.09° for the outermost SCT layer, so the detector can be divided into a maximum of $\lfloor 360/22.09 \rfloor = 16$ regions in φ (see Appendix A). Combining regions, as shown in Figure 59, this makes 80 combinations of regions, which is not enough to saturate a GPU with thousands of parallel processing units. Parallel regions along the beam axis are not feasible because bookkeeping would need to be over an entire φ region to avoid reusing space points.

6.3 Conclusions

The presented tracking methods have shown to improve certain aspects of the offline track reconstruction algorithms currently in production. A transform based filtering algorithm has been comprehensively analysed for its vertexing and track finding ability. A reimplementation of this algorithm with major improvements and subsequent tests showed that it is possible to reduce the tracking problem complexity from $O(n^8)$ to $O(n^2)$ and using much simpler operations. The improvements increase the resilience to pileup because the number of combinations found is six times smaller than with the seeding mechanism currently in use. The detected tracking and vertexing inefficiency requires careful consideration where the algorithm is applicable. One candidate is the trigger, where lower efficiencies are acceptable. A modified version without the improvements is currently used for the detection of the location of the interaction region, which could also be reimplemented with the improvements. As medium term goal, the algorithm can be discussed as low accuracy solution in detector areas of low interest to reduce time spent in these regions. The full development of a tracking solution was never scope of the analysis but results show the algorithm is versatile and has a huge potential, and can be changed to be used in different areas of application with low effort.

The parallelized tracking aims at improving the speed of algorithms currently in use and therefore maintaining both efficiency and purity of this approach. The study has shown it is possible to efficiently parallelize on a CPU the tracking prototype developed for GPUs and surpass the performance achieved with a full high-end GPU with only 4 CPU cores. This shows tracking can be efficiently parallelized on CPUs, allowing to run this parallelized tracking on the currently available hardware, unlike the GPU parallelized version which would require hardware acquisition on computing sites used for reconstruction. The parallelization of algorithms is a major project for the medium-term future, which is currently ongoing. This implementation serves as a proof of concept, implementing different forms of parallelization and combining them such that data locality is optimal for CPUs.

7 Conclusion

The upgrades to the LHC performed during its long shutdown from 2012 to 2014 (LS1) greatly increased the amount and the complexity of the data generated, while additionally much more of the data is selected for analysis by the ATLAS trigger system. For this reason, the ATLAS reconstruction software was improved in the course of this thesis to deal with the hugely increased workload. An investigation of the various aspects of the development of the ATLAS reconstruction software has been presented. Atypical aspects with respect to industrial software development include: its developer base with strongly varying degree of programming skill; many developers staying with ATLAS for only for a few years leading to loss of knowledge of the code; a code base of 6 million lines of code distributed over 2000 packages that has grown over many years; and the absence of dominant hot spots and bottlenecks. Typical challenges also faced include evolving requirements, adaptation of new technologies and dealing with legacy code, and budget restrictions. The key short-term goal during the shutdown was to speed up the software, while long-term goals included the optimization of the development process, improving software quality and the preparation for future computing hardware upgrades. For both goals, analysis of the software and the software development processes was necessary, which taken together form a major part of this thesis. In this chapter, the most important conclusions drawn from this analysis and the impact of the subsequently implemented measures are described and an outlook of possible future developments is given. The findings contributed significantly to the total increase in event processing rate by a factor of 4.51. They provided insight into the development process, revealed where further improvements are possible and facilitated future analysis.

7.1 Analysis and Result Summary

The applied improvements can be classified as technical improvements and algorithmic improvements, and for each, different types of software analysis are better suited. Technical improvements do not change how a task is being solved or its result, instead the use of resources is improved or techniques used to solve the same operations faster. As a consequence, these improvements can be applied irrespective of the problem and even without much insight into it. Results do not change except for rounding issues related to floating point arithmetic.

Algorithmic improvements solve the problem at hand in a different way or change the problem to be solved. This requires a good understanding of the task, and since it almost

always also leads to changes in the output, it requires understanding the physics requirements, i.e. the importance of different reconstructed physics objects for physics analysis. In the following, I will summarize the improvements alongside the analysis.

7.1.1 Hot Spot Analysis

A CPU time analysis showed there are no clear hot spots, as the hottest functions have a self-cost of less than 1% of the total runtime. Self-cost refers to (CPU) time spent in a function, without the time spent in functions that were called from this function. When combining all time spent in functions provided by a particular library, three external libraries stood out as major contributors to runtime. The allocator, tcmalloc version 0.99, the Linux system math library libm and the Class Library for High Energy Physics (CLHEP) combined were responsible for 35.6% of the total runtime. By replacing all three with faster libraries, the reconstruction time could be reduced by around 13% for Run 2 workloads. While tcmalloc and libm could be replaced by libraries with the same API, for replacing CLHEP, the Eigen library was chosen, which has a different API. Because CLHEP was used throughout the reconstruction code, the change required changing about 1000 code packages. To facilitate future changes, access to Eigen was wrapped using templates. This way, changing the precision or exchanging the library or only parts of it in the future can be performed with low effort, while the call to the Eigen library through the template causes no additional runtime cost. The new libraries all introduced support for vectorization, which opened up new possibilities to optimize the code in the future.

The Magnetic Field service, which is extensively used, contained the only function where a significant amount of time has been spent. This function did not show up in the CPU time analysis, because almost all of the time was spent waiting for memory. The introduction of a cache dramatically reduced the time spent here, leading to an 20% speed improvement for reconstruction under Run 2 workloads.

7.1.2 Hardware Usage Analysis

While the CPU may be the limiting factor in that it never waits for data, it does not mean the CPU is performing as much work as it could (without saying anything about if the work could be avoided), because modern CPUs have many parallel resources. An analysis of the CPU's hardware counters revealed that the code ran at 1 instruction per CPU cycle (IPC), while the CPU in the test setup could reach a maximum of 4 IPC. An analysis of the assembly showed there were very few vector operations, which are capable of executing up to 8 instructions at once on the tested CPU.

The autovectorization feature of recent compilers did not lead to any speedup after the introduction of the new libraries described in 7.1.1, due to the types of operation and complicated data structures. As compilers get better, this may change in the future, but large gains are unlikely due to the data being scattered in memory. Before the library change, vectorization was not possible because of unaligned memory returned from the allocator.

The xAOD, a new data format which was introduced during LS1 in parts of the reconstruction, stores data internally as structure of array, which facilitates vectorization and can reduce the number of cache misses. The IPC and autovectorization did not improve, but it increases the probability future compilers will be able to exploit the data alignment. Introducing the format to further areas e.g. in the tracking can enable developers to write vectorizing code. The new data structure can be read directly by the analysis software, making a previously needed costly conversion step after reconstruction unnecessary, such that significant computational cost outside of reconstruction was saved.

Another technical improvement was the update of the Event Data Model (EDM) used in the tracking code, which is used to communicate between algorithms. Heavily utilized

classes were rewritten to reduce dynamic memory allocation and templated to reduce their codebase by almost 10.000 lines of code. Both the xAOD and the EDM implementation have happened concurrently with other optimizations, which is why their individual impact on speedup cannot be quantified easily. All concurrent optimizations have in total improved reconstruction runtime by a factor of 1.54 for Run 2 workloads.

7.1.3 Algorithm Level Analysis

For algorithmic improvements, hot spots and hardware usage are not as important as finding out which data processing step is the most computing cost intensive. To this end, I analyzed the runtime per subdetector, because the software is divided into domains analogous to the subdetectors of ATLAS, and per algorithm. This analysis showed the Inner Detector (ID) domain required 56% of the total runtime, under Run 1 workloads, and 68% under Run 2 workloads. Within the ID, 40% (Run 1 workload) respectively 46% (Run 2 workload) of this time was spent by just one of the 32 ID algorithms. This was the ID silicon tracking, a combinatorial algorithm. It consists of many individual steps, but by reducing the number of combinations processed, all of these steps will take less time. The combinations to be removed had to be chosen carefully, because by removing combinations, tracks can become unfindable.

To find secondary particles, which can originate anywhere within the ID, all combinations of measurements were considered, causing this processing step to be very expensive. Not all secondary particles are equally important for analysis. The most important secondary particles are electron-positron pairs. Electrons and positrons have an identifiable signature within the Electromagnetic Calorimeter. The so-called backtracking improvement restricted the particle search to regions where such signatures were found by the calorimeter and extended them from the outer ID regions inwards towards the center of the detector. This required the calorimeter reconstruction to take place before the ID reconstruction, such that this information was available, whereas the previous order was to start reconstruction from the innermost subdetector system going outwards. This improvement alone sped up the overall reconstruction by a factor of 1.19 (Run 1 workloads) respectively 1.83 (Run 2 workloads).

The finding of primary particles, which originate very close to the interaction region, starts with the creation of seeds, which are tuples of 3 measurements which define the direction used to search for particle track candidates. Around 60 seeds are found for each track. My analysis shows that the number of seeds is linear with the effort of the subsequent, computationally much more expensive tracking step. Two different optimizations have been applied to the seeding. One optimization favors seeds which have approximately the same direction as another seed and rejects others if too many are found. When a track candidate is found, the seeds whose measurements have already been included in a track are discarded. The second optimization is the restriction of the region in which particle tracks can originate after a subset of the measurements has been processed. A third optimization allows deciding that a track cannot be found earlier, by decreasing the number of allowed missing measurements for a track. All optimizations combined lead to a 1.54 factor reconstruction speed improvement. These improvements did not negatively affect the physics performance.

7.1.4 Parallelizability Study

Parallelizing code does not in all cases require understanding of the underlying problem. The ATLAS reconstruction is an embarrassingly parallel problem, as each collision event can be processed independently. The reason to increase parallelism is that the reconstruction needs more memory than is available per core. The solution for Run 2 was using multi-processing, by starting one process and forking multiple child processes. All child process share memory regions that are unchanged. While this saved 40% memory, it

will not be sufficient for future workloads and future computing hardware. This is why parallelism within an event is introduced, because less memory is needed if more cores can be kept busy with less events processed in parallel.

Running multiple algorithms of the same event in parallel is the idea behind the framework AthenaMT. To be able to run algorithms within this framework, they have to be updated to be thread-safe, because it also runs multiple events in parallel in a single instance. This requires a significant effort, as most algorithms in the ATLAS reconstruction are not const correct. With a dependency study analyzing the Inner Detector algorithms I showed that the algorithms within the ID spent at least 95% of the ID reconstruction runtime on the critical path, i.e. a sequence of algorithms that has a linear dependency. This means, algorithm parallelism can only occupy more than one CPU per event during around 5% of the ID reconstruction runtime. Independently of these findings and because const-correctness has other advantages, currently algorithms are adapted to work in AthenaMT.

Another approach to make use of multiple CPUs while processing one event is to parallelize within algorithms. Parallelizing a loop with many independent iterations can be trivial. Unfortunately, the ATLAS code does not have such loops which need a significant amount of time. This is why it is necessary to understand each task to identify areas where parallelization is possible. I suggested a way to parallelize each of the main steps of the ID reconstruction, which could be implemented after the algorithms are made thread safe for AthenaMT. For the tracking step, which has a bookkeeping layer that prevents parallelization, my analysis concludes that for future workloads, the bookkeeping improves performance of the ID tracking only by a few percent and is even counterproductive in future high pileup scenarios. It can therefore be abandoned without losing performance such that the ID tracking can be efficiently parallelized for workloads expected in future runs.

7.1.5 Development Process in ATLAS

The challenge to find algorithmic improvements is posed by the complex interplay of the algorithms and the responsibility of different people over different parts of the code. Typically, one or a few persons are the only ones maintaining a part of the software, and if they leave ATLAS the code can remain untouched for years. This is a particular problem for high energy physics experiments because they have a huge turnover of developers due to the many limited duration contracts. The software quality assurance in ATLAS' development process, which has had very few safeguards against low-quality or abandoned code, is currently being improved by a peer-review, comment and discussion system, in part due to results from this thesis. Involving more people in the development can lead to a broader understanding of how different parts of the software influence each other, and as we have seen, the understanding of how algorithms can influence one another can lead to very large gains in CPU time.

The presented algorithmic improvements also led to a loss of information in that fewer particle tracks are found, which had to be carefully selected such that no important information was lost. This is another problem the developers face, as they have to be aware which information is vital and which is not. This requires the developer to be proficient in both the affected physics and programming.

7.2 New Algorithms and Proposed Optimizations

Additional to the analysis of the existing implementations and their optimization, I tested a number of new methods to reduce the tracking runtime. Besides studies of the achievable speedup with different parallel implementations of the tracking, I implemented and analyzed the usability of a transform-based algorithm that has the potential to reduce the work required by tracking to a small fraction of what it is now because of its lower

complexity. It is based on the combination of two algorithms used in the ATLAS Trigger during Run 1.

The algorithms can be used to solve two different problems. The first is finding primary vertex locations and identifying the signal vertex. For one particular event type, a 40 pileup Z to μμ decay event, the algorithm finds and identifies the signal vertex in 93% of the cases with a metric I developed. While this is not enough to restrict reconstruction to the area of the found vertex because 7% of the vertices would be lost, it may be usable for the Trigger because the Trigger works only on smaller regions of interest within which the algorithm would be much more likely to find the vertex.

To avoid losing all tracks from a non-identified vertex, I extended the algorithm to solve a second problem; to identify combinations of measurements that may be part of a track. The algorithm can find 94.8% of tracks with p_T above 1GeV/c. The found measurements would still need to be processed by a tracking algorithm. Therefore, its result can be compared to the seeding algorithm's output. The new algorithm is more resilient to higher pileup than the original algorithms it was developed from, because it produces much less combinations, also much less than the seeding algorithm. The new algorithm has much simpler operations than the seeding and its complexity is $O(n^2)$ compared to $O(n^3)$, which leads to the new algorithm being between 8 and 17 times faster than the current ATLAS seeding. Though having a lower complexity, the algorithm filters out many more fake seeds, producing 6 times less combinations. As my analysis shows, this directly corresponds to a factor 6 improvement in tracking speed. The algorithm could be parallelized easily but the benefit of doing so for an algorithm that takes up less than 1% of the runtime for current workloads is dismissible. Parallelization may become interesting for very high pileup scenarios when the algorithm takes more time.

The application of the algorithm depends on the number of real tracks it can find, which is with 94.8% lower than the number of tracks the seeding finds. Applying the algorithm only to pileup interactions may be acceptable, as the physics efficiency is not as important here. This has the potential to reduce the tracking runtime to a minor contributor to the overall runtime. The algorithm could also be reintroduced in the Trigger, where the previous version was retired because it was not robust with current pileup scenarios.

7.3 Outlook

The ATLAS code quality has been addressed by various means during this thesis and continues to receive more attention than before. Recognizing the importance of this topic has the potential to reduce the number of bugs, increase productivity, speed up the software and avoid losing knowledge of the code by distributing it over multiple people, to name only a few. To achieve this, the consequent application of the planned peer review process is required. Collaboration of developers from different areas should be encouraged and incentivized.

The reconstruction workloads will continue to increase and my analyses have shown that the current algorithms will become unsustainably slow with expected reconstruction workloads. The current state of high-end multi-purpose CPU cores must be seen as the pinnacle of this CPU architecture's development, increasing sequential processing speed of single cores cannot be expected. If predictions of many-core computers or heterogeneous computer systems becoming the norm turn out to be true, more parallelism needs to be introduced into the ATLAS software or even more resources will lie idle. The ACTS project [97] is currently working on rewriting central parts of the tracking such that it is experiment independent and thread safe.

The higher energy efficiency of specialized hardware may force ATLAS to maintain different hardware platforms. Maintaining multiple versions of the ATLAS software is not practical in order to support heterogeneous hardware, such that either compilers supporting the translation of C++ to other platforms need to be found or created or parts of the

software need to be rewritten in a language that supports multiple platforms. Because parallelized code tends to become less maintainable, outsourcing parallelism into libraries should be considered wherever possible.

BIBLIOGRAPHY

[1] L. Evans, "The Large Hadron Collider," *Annu. Rev. Nucl. Part. Sci.*, vol. 61, p. 435–466. 32 p, 2011.

[2] ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider," *JINST*, vol. 3, p. S08003, 2008.

[3] ATLAS Collaboration, *ATLAS Computing Technical Design Report*, no. March. Geneva, 2005.

[4] R. Seuster, "Source Lines of Code." [Online]. Available: http://seuster.web.cern.ch/seuster/SLOC/. [Accessed: 27-Apr-2016].

[5] "WLCG public website." [Online]. Available: http://wlcg-public.web.cern.ch/. [Accessed: 27-Apr-2016].

[6] S. Chekanov, C. Chen, J. Cochran, F. De Lorenzi, A. Ruiz, S. Prell, K. Yamamoto, P. Bernat, and M. Campanelli, "Tracking Performance of the Proposed Inner Tracker Layout for the ATLAS Phase-II Upgrade," 2013.

[7] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's J.*, pp. 1–9, 2005.

[8] R. R. Schaller, "Moore's law: past, present and future," *Spectrum, IEEE*, vol. 34, no. 6, pp. 52–59, 1997.

[9] J. Shalf, J. Bashor, D. Patterson, K. Asanovic, K. Yelick, K. Keutzer, and T. Mattson, "The Manycore Revolution: Will HPC Lead or Follow," *Scidac*, vol. Fall, pp. 40–49, 2009.

[10] I. Bird, P. Buncic, F. Carminati, M. Cattaneo, P. Clarke, I. Fisk, M. Girone, J. Harvey, B. Kersevan, P. Mato, R. Mount, and B. Panzer-Steindel, "Update of the Computing Models of the WLCG and the LHC Experiments," no. CERN-LHCC-2014-014. LCG-TDR-002, 2014.

[11] R. Langenberg, A. Morley, A. Salzburger, M. Elsing, N. van Eldik, R. Mashinistov, and the Atlas Collaboration, "Preparing the track reconstruction in ATLAS for a high multiplicity future," *J. Phys. Conf. Ser.*, vol. 513, no. 2, p. 22018, 2014.

[12] N. Chauhan, G. Kabra, T. Kittelmann, R. Langenberg, R. Mandrysch, A. Salzburger, R. Seuster, E. Ritsch, G. Stewart, N. van Eldik, and R. Vitillo, "ATLAS Offline Software Performance Monitoring and Optimization," 2013.

[13] *Convention for the establishment of a European organization for nuclear research: Paris, 1st July, 1953. Convention pour l'établissement d'une Organisation européenne pour la Recherche nucléaire : Paris, le 1er juillet 1953*. Geneva: CERN, 1953.

[14] "CERN member states." .

[15] CMS Collaboration, "The CMS experiment at the CERN LHC," *JINST*, vol. 3, p. S08004, 2008.
[16] N. Neri, "The LHCb experiment," *2003 IEEE Nucl. Sci. Symp. Conf. Rec. (IEEE Cat. No.03CH37515)*, vol. 3, 2003.
[17] The ALICE Collaboration, "The ALICE experiment at the CERN LHC," *J. Instrum.*, vol. 3, no. 8, p. S08002, 2008.
[18] ATLAS Collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC," *Phys. Lett. B*, vol. 716, p. 1, 2012.
[19] CMS Collaboration, "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC," *Phys. Lett. B*, vol. 716, p. 30, 2012.
[20] O. S. Brüning, P. Collier, P. Lebrun, S. Myers, R. Ostojic, Poole, and P. Proudlock, "LHC Design Report," Geneva, 2004.
[21] "ATLAS Luminosity Public Results." [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/LuminosityPublicResults. [Accessed: 24-Aug-2016].
[22] "ATLAS Data Summary 2012." [Online]. Available: https://atlas.web.cern.ch/Atlas/GROUPS/DATAPREPARATION/DataSummary/2012/run-summary.html. [Accessed: 27-Apr-2016].
[23] ATLAS Collaboration, "Pile-up subtraction and suppression for jets in ATLAS." 2013.
[24] TOTEM Collaboration, "Luminosity-Independent Measurement of the Proton-Proton Total Cross Section at $\sqrt{s}$ = 8 TeV," *Phys. Rev. Lett.*, vol. 111, no. TOTEM-2012-005. CERN-PH-EP-2012-354, p. 012001. 8 p, Nov. 2012.
[25] N. Kimura, "Fast Tracker: a hardware real time track finder for the ATLAS trigger system," *J. Instrum.*, vol. 9, no. 4, p. C04012, 2014.
[26] M. Capeans, T. Flick, R. Vuillermet, G. Darbo, H. Pernegger, K. Einsweiller, M. Garcia-Sciveres, M. Elsing, O. Rohne, and C. Gemme, "ATLAS Insertable B-Layer Technical Design Report," *Cern. Atlas-Tdr-19*, no. September, 2010.
[27] ATLAS Collaboration, "Basic ATLAS TRT performance studies of Run 1." 2014.
[28] ATLAS Collaboration, "Physics at a High-Luminosity LHC with ATLAS." 2013.
[29] T. Cornelissen, M. Elsing, S. Fleischmann, W. Liebig, E. Moyse, and A. Salzburger, "Concepts, Design and Implementation of the ATLAS New Tracking (NEWT)," Geneva, Mar. 2007.
[30] ATLAS Collaboration, "ATLAS Inner Detector: Technical Design Report, Vols. 1,2," Geneva, 1997.
[31] R. E. Kalman and others, "A new approach to linear filtering and prediction problems," *J. basic Eng.*, vol. 82, no. 1, pp. 35–45, 1960.
[32] R. Frühwirth, "Application of Kalman filtering to track and vertex fitting," *Nucl. Instrum. Methods Phys. Res., A*, vol. 262, no. HEPHY-PUB-503, p. 444. 19 p, Jun. 1987.
[33] A. Lund, L. Bugge, I. Gavrilenko, and A. Strandlie, "Track parameter propagation through the application of a new adaptive Runge-Kutta-Nyström method in the ATLAS experiment," *J. Instrum.*, vol. 4, 2009.
[34] H. M. Gray and E. W. Hughes, "The Charged Particle Multiplicity at Center of Mass Energies from 900 GeV to 7 TeV measured with the ATLAS Experiment at the Large Hadron Collider," Pasadena, USA, California Institute of Technology, Pasadena, USA, 2010.
[35] ATLAS Collaboration, "Charged-particle multiplicities in pp interactions measured with the ATLAS detector at the LHC," *New J. Phys.*, vol. 13, p. 53033, 2011.
[36] F. Winklmeier, "The ATLAS High Level Trigger infrastructure, performance and future developments," in *Real Time Conference, 2009. RT '09. 16th IEEE-NPSS*, 2009, pp. 183–188.
[37] "ATLAS Computing Software Releases." [Online]. Available: http://atlas-computing.web.cern.ch/atlas-computing/projects/releases/status/. [Accessed: 27-Apr-2016].

[38] P. Calafiura, C. Leggett, D. R. Quarrie, H. Ma, and S. Rajagopalan, "The StoreGate: a Data Model for the Atlas Software Architecture," Berkeley, CA, Jun. 2003.
[39] C. Leggett, S. Binet, K. Jackson, D. Levinthal, M. Tatarkhanov, and Y. Yao, "Parallelizing ATLAS Reconstruction and Simulation: Issues and Optimization Solutions for Scaling on Multi- and Many-CPU Platforms," *Int. Conf. Comput. High Energy Nucl. Phys. (Chep 2010)*, vol. 331, 2011.
[40] J. Knobloch, L. Robertson, and others, *LHC computing Grid technical design report*, vol. 24, no. June. 2005.
[41] N. Garelli, M. T. Morar, and W. Vandelli, "Balancing the resources of the High Level Trigger farm of the ATLAS experiment," Geneva, Jun. 2012.
[42] "European Middleware Initiative." [Online]. Available: http://www.eu-emi.eu/. [Accessed: 25-Oct-2016].
[43] I. Bird, "Status of the WLCG project, including Financial Status," Geneva, Sep. 2015.
[44] "SPEC 2006 Benchmark Website." [Online]. Available: http://www.spec.org/cpu2006/. [Accessed: 27-Apr-2016].
[45] "HEP-SPEC 2006 Benchmark Website." [Online]. Available: https://w3.hepix.org/processors/dokuwiki/doku.php?id=benchmarks:introduction. [Accessed: 27-Apr-2016].
[46] ATLAS Collaboration, "Charged-particle distributions in $\sqrt{s} = 13\;\mbox{TeV}$ pp interactions measured with the ATLAS detector at the LHC." 2015.
[47] H. Sakamoto, "The ATLAS Distributed Computing: The Challenges Of The Future," in *The International Symposium on Grids and Clouds (ISGC)*, 2013, p. 10.
[48] "Root Homepage." [Online]. Available: http://root.cern.ch/. [Accessed: 27-Apr-2016].
[49] "ATLAS Dashboard Homepage." [Online]. Available: http://dashb-atlas-job.cern.ch/dashboard/request.py/dailysummary. [Accessed: 27-Apr-2016].
[50] P. Prabhu, Y. Zhang, S. Ghosh, D. I. August, J. Huang, S. Beard, H. Kim, T. Oh, T. B. Jablin, N. P. Johnson, M. Zoufaly, A. Raman, F. Liu, and D. Walker, "A survey of the practice of computational science," *State Pract. Reports - SC '11*, p. 1, 2011.
[51] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?," in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, SECSE 2009*, 2009, pp. 1–8.
[52] "Human Resources Required for ATLAS Operation." [Online]. Available: http://atlas-service-enews.web.cern.ch/atlas-service-enews/2007-8/features_07-8/features_otsmou.php. [Accessed: 27-Apr-2016].
[53] "Jira." [Online]. Available: https://www.atlassian.com/software/jira. [Accessed: 14-Jul-2016].
[54] "Coverity." [Online]. Available: http://www.coverity.com/. [Accessed: 14-Jul-2016].
[55] S. Jarp, A. Lazzaro, J. Leduc, and A. Nowak, "How to harness the performance potential of current multi-core processors," *J. Phys. Conf. Ser.*, vol. 331, no. 1, p. 12003, 2011.
[56] "Haswell Architecture Detailed Analysis." [Online]. Available: http://www.realworldtech.com/haswell-cpu. [Accessed: 28-Apr-2016].
[57] J. Mattmann and C. Schmitt, "Track finding in ATLAS using GPUs," Geneva, Jun. 2012.
[58] "Valgrind Homepage." [Online]. Available: valgrind.org. [Accessed: 28-Apr-2016].
[59] "Gperftools CPU Profiler Homepage." [Online]. Available: https://gperftools.googlecode.com/git/doc/cpuprofile.html. [Accessed: 28-Apr-2016].
[60] "ATLAS Performance Monitoring Board." [Online]. Available: http://atlas-pmb.web.cern.ch/atlas-pmb/. [Accessed: 12-Aug-2016].
[61] "CUDA Homepage." [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed: 28-Apr-2016].

[62] "OpenCL Homepage." [Online]. Available: https://www.khronos.org/opencl/. [Accessed: 28-Apr-2016].
[63] "Cilk Homepage." [Online]. Available: www.cilk.com. [Accessed: 28-Apr-2016].
[64] "Threading Building Blocks Homepage." [Online]. Available: https://www.threadingbuildingblocks.org/. [Accessed: 28-Apr-2016].
[65] W. Kutta, "Beitrag zur näherungsweisen Integration totaler Differentialgleichungen," *Z. Math. Phys.*, no. 46, pp. 435–453, 1901.
[66] "perf." [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page. [Accessed: 14-Jul-2016].
[67] "Intel VTune Homepage." [Online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe. [Accessed: 28-Apr-2016].
[68] P. Calafiura, S. Eranian, D. Levinthal, S. Kama, and R. a Vitillo, "GOoDA: The Generic Optimization Data Analyzer," *J. Phys. Conf. Ser.*, vol. 396, no. 5, p. 52072, 2012.
[69] "Intel Architecture CPU Optimization Manual." [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf. [Accessed: 28-Apr-2016].
[70] "Intel CPI Interpretation Website." [Online]. Available: https://software.intel.com/en-us/node/544403. [Accessed: 28-Apr-2016].
[71] S. Kama, R. Seuster, A. G. Stewart, and A. R. Vitillo, "Optimizing ATLAS code with different profilers," Geneva, Sep. 2013.
[72] M. S. N. Rauschmayr, "Optimisation of LHCb Applications for Multi- and Manycore Job Submission," 2014.
[73] A. Limosani, "Private Communication." .
[74] "CLHEP Homepage." [Online]. Available: http://proj-clhep.web.cern.ch/proj-clhep/. [Accessed: 28-Apr-2016].
[75] "Intel Pintools Homepage." [Online]. Available: www.pintool.org. [Accessed: 28-Apr-2016].
[76] P. Mato, "GAUDI-Architecture design document," Geneva, Nov. 1998.
[77] "VDT Library Homepage." [Online]. Available: https://svnweb.cern.ch/trac/vdt. [Accessed: 28-Apr-2016].
[78] R. Seuster, "Private Communication." .
[79] "Jemalloc Homepage." [Online]. Available: http://www.canonware.com/jemalloc/. [Accessed: 26-Apr-2016].
[80] "Intel Math Kernel Library Homepage." [Online]. Available: https://software.intel.com/en-us/intel-mkl. [Accessed: 28-Apr-2016].
[81] "Eigen Library Homepage." [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page. [Accessed: 28-Apr-2016].
[82] R. M. Bianchi and R. A. Vitillo, "Private Communication." .
[83] N. "Cornelissen, T G and Van Eldik, M. Elsing, W. Liebig, E. Moyse, N. Piacquadio, K. Prokofiev, A. Salzburger, and A. Wildauer, "Updates of the ATLAS Tracking Event Data Model (Release 13)," 2007.
[84] M. Gibbs and B. Stroustrup, "Fast dynamic casting," *Softw. Pract. Exp.*, vol. 36, no. 2, p. 18, 2006.
[85] S. Snyder, "Private Communication." .
[86] A. Salzburger, "Optimisation of the ATLAS Track Reconstruction Software for Run-2," Geneva, May 2015.
[87] I. Gavrilenko, "Private Communication." .
[88] P. V. C. Hough, "Method and means for recognizing complex patterns," *US Pat. 3,069,654*, vol. 21, pp. 225–231, 1962.
[89] A. Salzburger and D. Kuhn, "A Parametrization for Fast Simulation of Muon Tracks in the ATLAS Inner Detector and Muon System," Innsbruck U., Innsbruck, 2003.
[90] A. Salzburger, "Private Communication." .
[91] R. Langenberg, "Study of Event Topology for a new Fast Primary Vertex Finder for the ATLAS Trigger," Geneva, Dec. 2015.

[92] Particle Data Group, "Particle Physics Booklet," *Phys. Rev. D*, vol. 86, no. 1, p. 10001, 2012.
[93] ATLAS Collaboration, "Reconstruction of primary vertices at the ATLAS experiment in Run 1 proton--proton collisions at the LHC," Geneva, Nov. 2016.
[94] S. Artz, "Private Communication." .
[95] D. Emeliyanov and J. Howard, "GPU-Based Tracking Algorithms for the ATLAS High-Level Trigger," Geneva, May 2012.
[96] A. Tavares Delgado, P. Conde Muiño, T. Bold, J. Augusto, S. Kama, M. Negrini, A. Sidoti, L. Rinaldi, S. Tupputi, J. Baines, M. Bauce, A. Messina, D. Emeliyanov, A. K. Elliott, Z. D. Greenwood, and S. Laosooksathit, "An evaluation of GPUs for use in an upgraded ATLAS High Level Trigger," Geneva, Nov. 2015.
[97] "ACTS website." [Online]. Available: https://gitlab.cern.ch/acts/a-common-tracking-sw. [Accessed: 12-Apr-2017].
[98] F. Marcastel, "Public CERN Accelerator Complex Graphic," 2013. [Online]. Available: http://cds.cern.ch/record/1621583/files/. [Accessed: 11-Aug-2016].
[99] the Atlas Collaboration, "ATLAS Public Results - Standard Model Results." [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/StandardModelPublicResults. [Accessed: 09-May-2016].

Appendix A - APPROXIMATION ERROR OF BENDING IN THE φ-PLANE

The formula of the approximation of a track is very simple. It exploits that the sine of an angle close to zero is not very different from the angle itself. The circle segments defining the angle are within two pixel layers which in the worst case means combining the innermost pixel layer with the outermost, so in the order of 10cm. which is the case for the helix radii of even the lowest momentum the ATLAS detector is designed to measure.
Indices denote variables belonging to different layers. Index 1 and index 2 belong to the two seeding layers. The index T denotes the target layer. φ denotes the angle around the beam axis. ρ denotes the radius of a layer. The abstraction of having only one radius for a layer doesn't play a role here as both the exact formula and the approximation use this value. All values of variables with index one and two are known, as is the radius of the target.

$$\Delta\boldsymbol{\varphi} = \boldsymbol{\varphi_2} - \boldsymbol{\varphi_1} \tag{1}$$

$$\boldsymbol{\Delta\rho} = \boldsymbol{\rho_2} - \boldsymbol{\rho_1} \tag{2}$$

$$\boldsymbol{\varphi_T} = \boldsymbol{\varphi_1} + \frac{\boldsymbol{\Delta\varphi}}{\boldsymbol{\Delta\rho}}(\boldsymbol{\rho_T} - \boldsymbol{\rho_1}) \tag{3}$$

The exact formula to calculate the φ region in the detector requires calculating the intersection of the circle of the ATLAS detector layer and the helix circle. In addition to the two seeding layers, the beamspot can be used to define the helix circle in the algorithm. The details are explained here. The beamspot is here assumed to be at x=y=0 and is indexed by 0. The center of the circle defined by the target detector layer is also at 0,0. First the center of the circle defined by the helix is calculated. Note that this circle and the circle defined by the layers are on the same 2D plane because the helix axis is parallel to the beam axis. The index h denotes the helix. The Radius and center of the helix are calculated from the three points by constructing two chords through the three points:

$$\boldsymbol{y_{0,1}} = \frac{\boldsymbol{y_1} - \boldsymbol{y_0}}{\boldsymbol{x_1} - \boldsymbol{x_0}}(\boldsymbol{x} - \boldsymbol{x_1}) + \boldsymbol{y_1} \tag{4}$$

$$\boldsymbol{y_{1,2}} = \frac{\boldsymbol{y_2} - \boldsymbol{y_1}}{\boldsymbol{x_2} - \boldsymbol{x_1}}(\boldsymbol{x} - \boldsymbol{x_2}) + \boldsymbol{y_2} \tag{5}$$

For simplicity, the factor is called b

$$\mathbf{b_{0,1}} = \frac{\boldsymbol{y_1} - \boldsymbol{y_0}}{\boldsymbol{x_1} - \boldsymbol{x_0}} \tag{6}$$

$$\mathbf{b_{1,2}} = \frac{\boldsymbol{y_2} - \boldsymbol{y_1}}{\boldsymbol{x_2} - \boldsymbol{x_1}} \tag{7}$$

The two lines perpendicular to these two chords passing through their centers will intersect at the circle center. Establishing the chord centers m with:

$$\mathbf{m_{0,1}} = \left(\frac{\boldsymbol{x_1} - \boldsymbol{x_0}}{\mathbf{2}}, \frac{\boldsymbol{y_1} - \boldsymbol{y_0}}{\mathbf{2}}\right) \tag{8}$$

$$\mathbf{m_{1,2}} = \left(\frac{\boldsymbol{x_2} - \boldsymbol{x_1}}{\mathbf{2}}, \frac{\boldsymbol{y_2} - \boldsymbol{y_1}}{\mathbf{2}}\right) \tag{9}$$

And the perpendicular lines through the chord center with index p:

$$\boldsymbol{y_{0,1,p} = -\frac{1}{b_{0,1}}\left(x - \frac{x_1 + x_0}{2}\right) + \frac{y_1 + y_0}{2}} \tag{10}$$

$$\boldsymbol{y_{1,2,p} = -\frac{1}{b_{1,2}}\left(x - \frac{x_2 - x_1}{2}\right) + \frac{y_2 + y_1}{2}} \tag{11}$$

Solving $y_{1,2,p} = y_{0,1,p}$ for x:

$$\boldsymbol{x = \frac{b_{0,1}(x_2 + x_1) + b_{1,2}b_{0,1}(y_2 - y_0) - b_{1,2}(x_1 - x_0)}{2(b_{0,1} - b_{1,2})}} \tag{12}$$

After finding x, the y coordinate can be found substituting x in either of the two equations for the perpendicular lines $y_{0,1,p}$ or $y_{1,2,p}$. The helix circle radius is easily calculated, as we know that 0,0 is on the circle. The helix circle center coordinates are henceforth called x_h, y_h and the circle radius r_h:

$$\boldsymbol{r_\mathrm{h} = \sqrt{x_\mathrm{h}^2 + y_\mathrm{h}^2}} \tag{13}$$

We can now construct both circle equations. The circle equation for the helix:

$$\boldsymbol{(x - x_\mathrm{h})^2 + (y - y_\mathrm{h})^2 = r_\mathrm{h}^2} \tag{14}$$

As the circle center for the detector layer is at 0, 0 the formula for this circle is simple:

$$\boldsymbol{x^2 + y^2 = \rho_T^2} \tag{15}$$

Intersecting both circles, we obtain (within the design energy always) two intersection points. By subtracting both equations and expanding, we get a linear equation for x and y.

$$\boldsymbol{(x - x_\mathrm{h})^2 + (y - y_\mathrm{h})^2 - x^2 - y^2 = r_\mathrm{h}^2 - \rho_T^2} \tag{16}$$

$$\boldsymbol{y = \frac{\rho_T^2 - r_\mathrm{h}^2 + x_\mathrm{h}^2 + y_\mathrm{h}^2 - 2xx_\mathrm{h}}{2y_\mathrm{h}}} \tag{17}$$

Substituting y in (14) or (15) with (17) yields x. Using x in (14) or (15) returns y. Now one can replace the variables with values generating the highest error. For the ρ_T variable, this is the outermost barrel layer because the particle travels the longest distance in r through the magnetic field. Here is a particle's maximum bending in φ measured in a silicon detector.

$$\boldsymbol{\rho_T = 514\mathrm{mm}} \tag{18}$$

The smaller the size of a helix, the larger deviation of a circle section from a straight line in the r-z plane. The larger this deviation is, the larger the inaccuracy of the approximation. The lower transverse momentum limit with the smallest helix radius for this algorithm is 1GeV/c. The following variables and units are used: A perfectly homogeneous 2Tesla magnetic field denoted B, expressed in the formula as kg per Coulomb per second. The transverse momentum converted from eV/c to kg*m/s denoted by p_T. The helix radius in meters is denoted by r_h and the charge q of one electron in Coulomb [http://physics.nist.gov/cuu/Units/units.html].

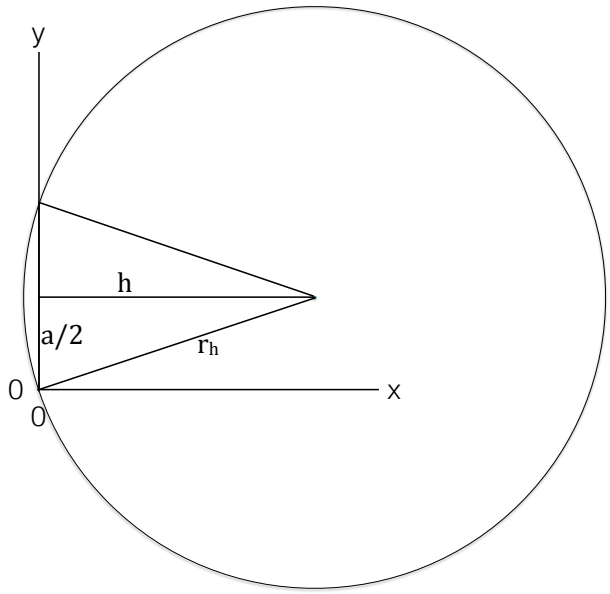


Figure 61: The helix circle always passes through (0, 0). Given (a, 0) also lies on the circle, the circle center is at (a/2, h).

$$\boldsymbol{r_h = \frac{p_T}{B * q}} \tag{19}$$

$$\boldsymbol{r_h = 1667.82047599mm} \tag{20}$$

Defining a helix with zero forward momentum simplifies the calculation. Given a particle with p_T of 1GeV/c originating at (0, 0, 0) in ATLAS coordinates (in mm) passing through the outermost SCT detector layer at (514, 0, 0). Then the center can be calculated as shown in Figure 61.

$$\boldsymbol{a = 514mm} \tag{21}$$

$$\boldsymbol{h = \sqrt{1667.82047599^2 - (\frac{514}{2})^2} = 1647.9005249} \tag{22}$$

So, the center of the helix is at (257, 1647.90). Using (17) and (15) returns x, here for the layer with ρ = 23:

$$\boldsymbol{y = \frac{23^2 - 1667.82047599^2 + 257^2 + 1647.9005249^2 - 514x}{2 * 1647.9005249}} \tag{23}$$

$$\boldsymbol{x^2 + (\frac{23^2 - 1667.82047599^2 + 257^2 + 1647.9005249^2 - 514x}{2 * 1647.9005249})^2 = 23^2} \tag{24}$$

$$\boldsymbol{x_1 = -22.70032} \tag{25}$$

$$\boldsymbol{x_2 = 22.74919} \tag{26}$$

The result closer to the input layer is correct, so x_2

$$\boldsymbol{22.74919^2 + y^2 = 23^2} \tag{27}$$

$$\boldsymbol{y_1 = -3.38738} \tag{28}$$

$$\boldsymbol{y_2 = 3.38738} \tag{29}$$

Using y_1 assumes a positive bending in the φ-plane. This corresponds to an azimuthal angle of -0.147815 in radians or -8.46918°. Similarly, the values for ρ = 50.5 are (50.0089, -7.02566) and -0.1396rad or -7.997042° and for ρ = 122.5 they are (121.648, -14.4227) and -0.1180rad or -6.761°. Using the approximation in (3) with the two layers at ρ = 23 and ρ = 122.5 returns $-0.040°$ (i.e. an absolute error of 0.040). Using the approximation to the target layer at 514mm with the two layers closest to each other results in $-0.039°$ (i.e. an absolute error of 0.039).

Appendix B - APPROXIMATION ERROR OF EXTRAPOLATION IN ρ-Z-PLANE

The approximation assumes a straight line for the extrapolation in z. The relative error in z increases for smaller helix radii and larger ρ. The absolute error increases with flatter angle (higher pseudorapidity) at the largest ρ with a track at the lower energy bound of 1GeV/c. The approximation is therefore for barrel:

$$\boldsymbol{z_T = z_0 + \frac{z_1 - z_0}{\rho_1 - \rho_0} * \rho_T} \tag{30}$$

Equivalently for endcaps the calculation of ρ for known z:

$$\boldsymbol{\rho_T = \rho_0 + \frac{\rho_1 - \rho_0}{z_1 - z_0} * z_T} \tag{31}$$

Using the detector geometry from [2], it can be calculated that the absolute error is larger in the barrel than on the endcaps due to the steeper track angle. The flattest angle in the outermost SCT barrel detector layer is 34.5° while the flattest angle on the innermost SCT endcap is 56.7°. To calculate the maximum absolute error, a track with 1GeV/c is assumed, passing through (514,0,749) which is a position on the outermost barrel layer in the barrel region on one of the forward ends of the barrel. Using calculated accurate positions of the track on the two innermost layers, the approximation is then used to extrapolate back to this position exhibiting the maximum error.

The helix equation in a moved coordinate system is

$$\boldsymbol{\gamma(\varphi) = \begin{pmatrix} Cx + R * cos(\varphi - \varphi_0) \\ Cy + R * sin(\varphi - \varphi_0) \\ h * \frac{\varphi}{2\pi} \end{pmatrix}} \tag{32}$$

With $\boldsymbol{Cx}$ and $\boldsymbol{Cy}$ denoting the position of the center of the helix in the coordinate system. φ denotes the current angle of the helix while φ_0 denotes the starting angle. The z axis is assumed to be parallel to the helix axis, as is the case in ATLAS.
Because we want the helix to intersect with the z axis of our coordinate system, we can replace $\boldsymbol{R}$ with $\sqrt{(Cx^2 + Cy^2}$. The starting point is also on the z axis, such that the starting angle φ_0 can be replaced by $\boldsymbol{\pi - \tan^{-1}\left(\frac{Cy}{Cx}\right)}$:

$$\boldsymbol{\gamma(\varphi) = \begin{pmatrix} Cx + \sqrt{(Cx^2 + Cy^2} * cos(\varphi - (\pi - \tan^{-1}\left(\frac{Cy}{Cx}\right))) \\ Cy + \sqrt{(Cx^2 + Cy^2} * sin(\varphi - (\pi - \tan^{-1}(\frac{Cy}{Cx}))) \\ h * \frac{\varphi}{2\pi} \end{pmatrix}} \tag{33}$$

For readability, I will keep the old denominations in the formula. The formula for the distance in ρ in the coordinate system is:

$$= \sqrt{(\boldsymbol{Cx} + \boldsymbol{R} * \boldsymbol{cos}(\boldsymbol{\varphi} - \boldsymbol{\varphi_0}))^2 + (\boldsymbol{Cy} + \boldsymbol{R} * \boldsymbol{sin}(\boldsymbol{\varphi} - \boldsymbol{\varphi_0}))^2} \quad (34)$$

A helix described by a charged 1GeV/c particle within a perfectly homogenous magnetic field of 2 Tesla has a radius of 1.667m as calculated in Appendix A. Note that the position around z does not change $\boldsymbol{\rho}$. To avoid division by 0 we assume the helix center to be at $\boldsymbol{Cx} = \boldsymbol{Cy} = \sqrt{\mathbf{1.667}}$. Replacing all variables by their values we get:

$$\boldsymbol{\rho} \approx \sqrt{\mathbf{6.6667} - \mathbf{6.6667} * \boldsymbol{cos}(\boldsymbol{\varphi})} \quad (35)$$

Because we know $\boldsymbol{\rho}$ and want to find out $\boldsymbol{\varphi}$ because $\mathbf{z}$ is linearly dependent on $\boldsymbol{\varphi}$, we invert this formula for positive $\boldsymbol{\rho}$ and $\boldsymbol{\varphi}$ up to π. This excludes particles with transverse momentum so low they do not leave the ID, which is fine as these are not reconstructed anyway:

$$\boldsymbol{\varphi} \approx \mathbf{cos}^{-1}(\mathbf{1} - \mathbf{0.15}\boldsymbol{\rho}^2) \quad (36)$$

Which means for 0.514m in $\boldsymbol{\rho}$ the helix travelled 0.282rad or 16.18°. The rotation length in z denoted $\boldsymbol{h}$ must therefore be 0.749m/0.282 * 2π. With all variables defined, we can now calculate the impact points in ρ and z of a perfect helix on the innermost detecting layers which will be used to calculate the straight line to the outermost layer. This will show the difference in z of the estimation and an ideal helix. The innermost layers are at $\boldsymbol{\rho}$ of 0.0325m and 0.0505m, until which the helix travelled in φ:

$$\mathbf{cos}^{-1}\left(\mathbf{1} - \mathbf{0.15} * \mathbf{0.0325}^2\right) \approx \mathbf{0.0178} \quad (37)$$

$$\mathbf{cos}^{-1}\left(\mathbf{1} - \mathbf{0.15} * \mathbf{0.0505}^2\right) \approx \mathbf{0.0277} \quad (38)$$

With these φ values, we can calculate z at these layers:

$$\mathbf{0.749m/0.282°} * \mathbf{2\pi} * \frac{\mathbf{0.0178°}}{\mathbf{2\pi}} \approx \mathbf{0.0473}\boldsymbol{m} \quad (39)$$

$$\mathbf{0.749m/0.282°} * \mathbf{2\pi} * \frac{\mathbf{0.0277°}}{\mathbf{2\pi}} \approx \mathbf{0.0736}\boldsymbol{m} \quad (40)$$

Using these two points $(\boldsymbol{\rho}, \boldsymbol{z})$: (0.0325,0.0473) and (0.0505,0.0736) to calculate a straight line.

$$\boldsymbol{z} \approx \mathbf{1.461}\boldsymbol{\rho} - \mathbf{0.00019} \quad (41)$$

I added the 0.00019 with higher precision than the rest just to show that this straight line does not intersect z = ρ = 0, although it gets very close. The approximated z value for ρ = 0.514 is therefore 0.7510m, so a delta of 2mm from the exact value. The bin size used in this thesis is 4.8mm in z for the outermost barrel layer, showing that even this extreme case can be found using the approximation in the absence of other effects.