IEKP-KA/2013-17

Karlsruhe Institute of Technology

Parallel Triplet Finding for Particle Track Reconstruction

Zur Erlangung des akademischen Grades eines
MASTER OF SCIENCE
von der Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)
genehmigte

MASTER THESIS

von

Daniel Funke

Referent: Prof. Dr. G. Quast
Institut für Experimentelle Kernphysik

Korreferent: Prof. Dr. P. Sanders
Institut für Theoretische Informatik

Betreuer: Dipl.-phys. H. Hauth
CERN, Genève, CH

Betreuer: Dipl.-phys. Dipl.-inform. D. Schieferdecker
Institut für Theoretische Informatik

Deutsche Zusammenfassung

Das Compact Muon Solenoid (CMS) Experiment am Teilchenbeschleuniger Large Hadron Collider (LHC) der Organisation Européenne pour la Recherche Nucléaire (CERN) nahe Genf, Schweiz, führte bereits zu zahlreichen Erkenntnissen, die unser Verständnis über die Interaktion der elementarsten Teilchen des Universums beträchtlich erweiterten. Insbesondere die Entdeckung eines Higgs-ähnlichen Bosons im Jahr 2012 [43] gilt bereits jetzt als einer der Meilensteine der modernen Physik. Diese Studien wurden durch die präzise Rekonstruktion der Spuren von Teilchen, die in den Proton-Proton-Kollisionen mit einer Schwerpunktenergie von 7 TeV entstanden sind, ermöglicht. Die Flugbahnen der Teilchen durch den CMS Spurdetektor – *Tracker* – werden von 16 588 Detektormodulen auf 75 Millionen Kanälen erfasst [44]. Die Rekonstruktion der Bahnen aus dieser Vielzahl an Messpunkten, genannt *Hits*, stellt eine enorme algorithmische Herausforderung dar. Wenn der LHC im Jahr 2015 seinen Dienst mit erhöhter Schwerpunktenergie (14 TeV) und Strahlintensität wieder aufnimmt, wird diese Herausforderung nochmals gesteigert, da mehr Spuren durch eine erhöhte Anzahl simultaner Proton-Proton-Interaktionen entstehen. Hierdurch erhöht sich der kombinatorische Aufwand der Teilchenspurrekonstruktion und damit die Laufzeit der involvierten Algorithmen beträchtlich.

Verschärfend kommt hinzu, dass die Taktfrequenz von Prozessoren die Grenzen des physikalisch Möglichen erreicht hat und daher Leistungssteigerungen bei CPUs in den letzten Jahren vor allem durch Einführung neuer Technologien wie Mehrkernprozessoren und Vektoreinheiten erzielt wurden. Die Nutzung dieser Technologien ist unabdingbar um den zukünftigen, steigenden Anforderungen gerecht zu werden, erfordert jedoch die Anpassung der zur Spurrekonstruktion eingesetzten Verfahren. Parallele und daten-lokale Algorithmen sind am besten geeignet, das Potenzial moderner Mehrkernprozessoren auszunutzen. Als Alternative zum momentan verwendeten iterativen Kalman-Filter-Verfahren [59] empfiehlt sich Spurrekonstruktion auf Basis eines *Zellularautomaten* [6, 80, 107]. Hierbei werden Teilchenspuren aus geeigneten Tripeln von Hits in benachbarten Detektorlagen gebildet, die aufgrund von simplen und lokalen Eigenschaften auf Kompatibilität geprüft werden.

Die vorliegende Arbeit befasst sich mit dem *Entwurf* eines parallelen Algorithmus zum Finden *gültiger* Hit-Tripel, d.h. Tripel von Messpunkten, die von ein und demselben Teilchen bei dessen Weg durch den Detektor erzeugt wurden. Die Anzahl an Tripeln, die sich aus Hits von verschiedenen Teilchen zusammensetzen – *fakes* – soll dabei gleichzeitig reduziert werden. Um die Kombinatorik beim Erzeugen der Tripel einzuschränken, soll der mögliche Detektorbereich für einen weiteren Hit mit Hilfe der Informationen gegeben

durch einen einzelnen Hit oder ein Hitpaar prädiziert werden. Sowohl für die Prädiktion als auch für die Identifizierung gültiger Tripel sollen einfache und lokale geometrische Berechnungen verwendet werden, um das Potenzial moderner CPUs *und* GPUs durch Parallelisierung auszuschöpfen. Die Daten des CMS-Detektors werden im heterogenen Worldwide LHC Computing Grid (WLCG) verarbeitet. Daher muss das verwendete Parallelisierungsframework eine Vielzahl an CPU- und GPU-Plattformen unterstützen können. Um schnell auf Hits innerhalb des prädizierten Bereiches zugreifen zu können, sollen diese in einer geeigneten geometrischen Datenstruktur organisiert werden.

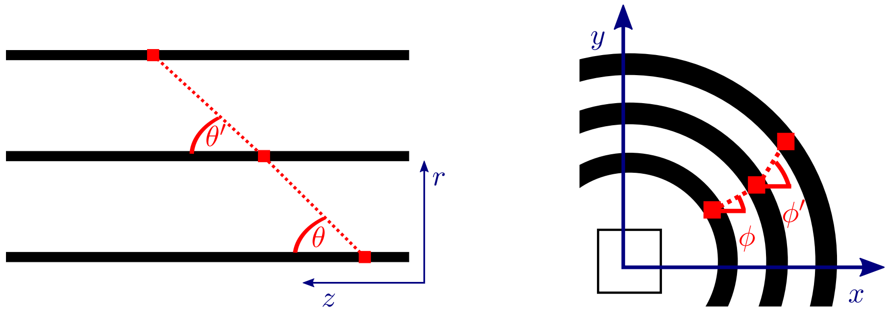
Der nächste Abschnitt stellt die entworfenen Algorithmen zur Hitpaarbildung, Tripeltprädiktion und Tripeltfilterung vor. Untersuchungen sowohl zur erzielten Datenqualität als auch zu Laufzeitmessungen werden im darauffolgenden Abschnitt dargelegt.

Algorithmen

Vor der Beschreibung der entwickelten Algorithmen sollen zunächst deren technologischen und physikalischen Grundlagen umrissen werden.

Um der Anforderung nach portabler Ausnutzung von Parallelismus auf Prozessoren und Grafikkarten verschiedener Hersteller gerecht zu werden, bildet die Open Computing Language (OpenCL), als offenes Framework für die Entwicklung massiv-paralleler Algorithmen [102], die Grundlage der präsentierten Implementierung. OpenCL ermöglicht es, einen Code sowohl auf CPU als auch GPU zur Ausführung zu bringen, erfordert jedoch die Berücksichtigung einiger Eigenheiten und Einschränkungen im Design der Algorithmen. Für maximalen Durchsatz auf dem *Compute Device* ist eine fein-granulare Parallelisierung ohne Verzweigungen im ausgeführten *Kernel* notwendig. Weiterhin erfordert das Fehlen von dynamischer Speicherallokation innerhalb eines Kernels, dass sämtlicher benötigter Speicher vor Kernelausführung vom *Host* alloziert wird. Daher verwenden alle im folgenden dargelegten Algorithmen einen Zwei-Phasen-Ansatz [152]: In einem ersten Durchlauf – der *Count*-Phase – wird nur die Anzahl der gefunden Ausgaben, z.B. Hitpaare, gezählt. Jeder Thread führt einen eigenen Zähler, über die nach Ablauf der ersten Phase eine Präfixsumme [159] gebildet wird. Der Host kann nun für die bekannte Anzahl von Ausgaben Speicher allozieren und den zweiten Durchlauf – die *Store*-Phase – initiieren, in dem die Ausgaben tatsächlich geschrieben werden. Die Berechnungsergebnisse des ersten Schrittes werden dem Zweiten hierbei durch einen „Orakel“-Bitstring zur Verfügung gestellt.

Die physikalischen Grundlagen sind durch die Detektorgeometrie und das Verhalten geladener Teilchen im Magnetfeld gegeben. Im Inneren des zylinderförmigen CMS-Detektors erzeugt eine supraleitende Spule ein 3.8 T starkes Magnetfeld, das die Flugbahn geladener Teilchen in der Ebene senkrecht zum Strahlrohr – transversale Ebene – krümmt, jedoch entlang des Strahlrohrs – longitudinale Ebene – unberührt lässt. Diese Eigenschaften können sowohl zur Prädiktion von Suchfenstern als auch Filterungen von Hittripeln herangezogen werden, wie in Abbildung 0.1 verdeutlicht. Ein weiteres Kriterium für die Identifikation valider Tripel, neben $d\phi$ und $d\theta$, ist der minimale Abstand zum nominalen Interaktionspunkt, sowohl in der transversalen (d_0) als auch longitudinalen (z_0) Ebene.



(a) In der longitudinalen Ebene bleibt die Flugbahn eines geladenen Teilchens unberührt vom Magnetfeld des Detektors. Die Bahn kann deshalb durch eine Gerade angenähert werden. Abweichungen zwischen den Winkeln θ und θ' sind auf Streuung und Messungenauigkeiten zurückzuführen. Daher kann des Verhältnis zwischen beiden Winkeln für gültige Tripel durch $\frac{\theta'}{\theta} \leq d\theta$ beschränkt werden.

(b) Die Krümmung der Spur in der transversalen Ebene verhält sich umgekehrt proportional zum Transversalimpuls des Teilchens. Daher kann die maximale Krümmung durch $|\phi' - \phi| \leq d\phi$ beschränkt werden.

Abbildung 0.1.: Schematische Abbildung der geometrischen Eigenschaften von gültigen Hittripeln.

Damit Hittripel in den Detektordaten effizient gefunden werden können, wird für jede Detektorlage eine *uniforme Grid-Datenstruktur* aufgebaut. Diese partitioniert die Lage in gleich große Zellen in z (longitudinale Ebene) und ϕ (transversale Ebene). Auf eine Gridzelle kann in $\mathcal{O}(1)$ Zeit zugegriffen werden; die Granularität des Grids bestimmt hierbei, wie dicht jede Zelle mit Hits besetzt ist. Die Grid-Datenstruktur wird in einem Zwei-Phasen-Algorithmus auf dem Compute Device erzeugt.

Der eigentliche Tripelfindungsprozess verläuft in drei Schritten:

1. **Paarfindung** generiert aus Hits in zwei gegebenen Detektorlagen Paare, die zu einer plausiblen Teilchenspur führen können. Das Suchfenster für den zweiten Hit wird basierend auf minimalen Transversalimpuls p_T und (d_0, z_0) eingeschränkt, um die Anzahl zu inspizierender Hits in der zweiten Detektorlage zu minimieren.

2. **Tripelprädiktion** erstellt Tripelkandidaten aus einem Hitpaar und Hits in einer gegebenen dritten Detektorlage. Der Pfad des Teilchens wird basierend auf dem Hitpaar und dem minimalen p_T auf die dritte Lage extrapoliert, um den kombinatorischen Aufwand zu begrenzen.

3. **Tripelfilterung** prüft Tripelkandidaten anhand der vorgestellten Kriterien auf Validität. Die erforderlichen Werte für das $d\phi$- und $d\theta$-Kriterium können direkt aus den Hitpositionen berechnet werden. Um den transversalen Abstand d_0 zum Strahlrohr zu ermitteln, müssen die Parameter des durch das Tripel beschriebenen

Kreises in transversaler Ebene berechnet werden. Hierfür wird das *Riemann-Fit*-Verfahren von Strandlie et al. [173] verwendet.

Alle Schritte werden durch Zwei-Phasen-Algorithmen auf dem Compute Device realisiert, die mit besonderem Hinblick auf Verzweigungsfreiheit und Ausnutzung der vorhanden Speicherhierarchie entworfen werden.

Resultate

Die entworfenen Algorithmen werden unter Verwendung simulierter Kollisionsereignisse *validiert* und *evaluiert*. Sowohl die erzielte Datenqualität als auch die Laufzeit hängen maßgeblich von den gewählten Parametern für die Filterkriterien $d\phi$, $d\theta$ und d_0 ab. Zu stark selektierende Werte senken die *Effizienz*, d.h. die Anzahl gefundener simulierter Spuren, während zu großzügige Werte die *Fake-Rate*, d.h. die Zahl der Tripel die sich aus Hits von unterschiedlichen simulierten Teilchen zusammensetzen, erhöht. Parameter, die beide Kriterien abwägen, werden durch die Analyse von $t\bar{t}$-Ereignissen, die sich durch eine komplexe Topologie auszeichnen, bestimmt. Abbildung 0.2 zeigt die mit den ausgewählten Parametern erzielte Datenqualität.

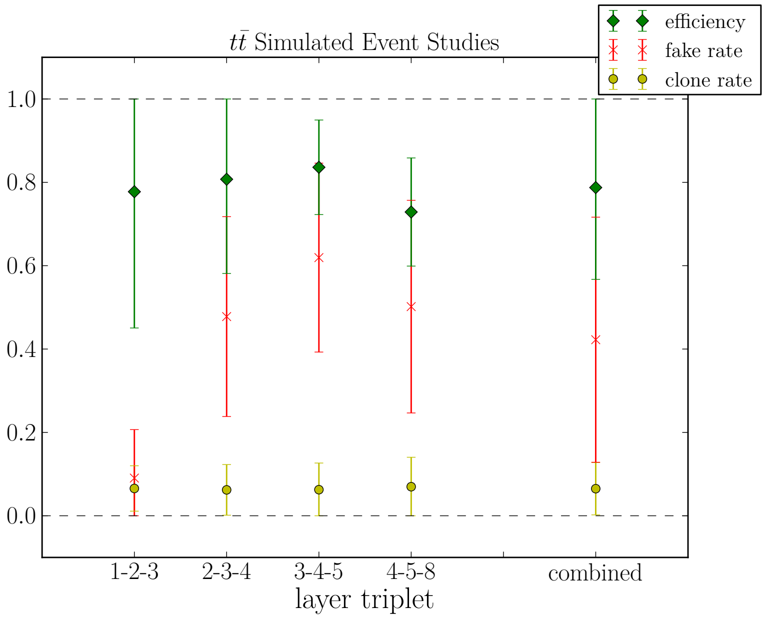


Abbildung 0.2.: Übersicht der erzielten Datenqualität für unterschiedliche Detektorlagenkombination in $t\bar{t}$-Ereignissen. Die ersten drei Lagen befinden sich im hochpräzisen Pixeldetektor und erlauben daher sehr stark selektierende Filterparameter. Ab der vierten Lage kommen weniger präzise Siliziumstreifen zum Einsatz. Daher müssen großzügigere Filterwerte gewählt werden, um eine ähnliche Effizienz wie im Pixeldetektor zu erzielen, die aber auch eine höhere Fake-Rate bedingen. Die *Clone-Rate* gibt den Anteil der Spuren an für die mehr als ein Tripel gefunden wurde, z.B. aufgrund überlappender Detektormodule.

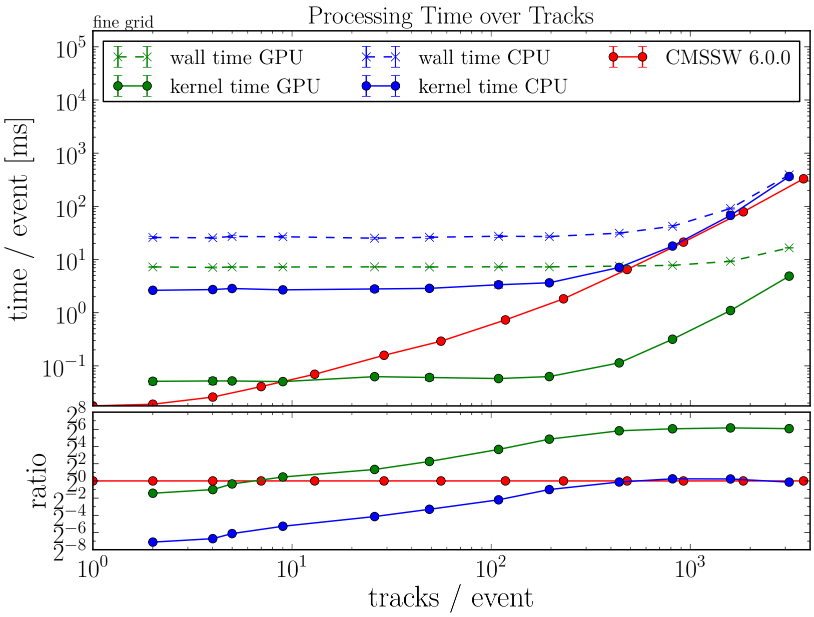


Abbildung 0.3.: Laufzeitverhalten des entworfenen Algorithmus auf CPU und GPU in Abhängigkeit von der Anzahl Spuren im Ereignis. Die *Kernel Time* beinhaltet die vom Compute Device benötigte Zeit zum Ausführen der Algorithmen. Die *Wall Time* umfasst neben der Kernel Time die Zeit für Datentransfers zwischen Host und Compute Device, sowie den Overhead der OpenCL Laufzeitumgebung. Um diese Fixkosten zu amortisieren werden 30 Ereignisse *simultan* auf dem Compute Device prozessiert. Die Laufzeit für CMSSW umfasst das Finden von Tripeln im *ersten* von insgesamt sieben spezialisierten Rekonstruktionsschritten.

Die Laufzeit wird maßgeblich von der Anzahl der Spuren im Kollisionsereignis bestimmt. In Abbildung 0.3 ist das Laufzeitverhalten der entworfenen Algorithmen im Vergleich zu den aktuell verwendeten Rekonstruktionsalgorithmen aus dem CMS Software Framework (CMSSW) dargestellt,

Schlussfolgerungen

Der präsentierte Algorithmus zeigt, dass trotz einfacher geometrischer Berechnungen eine hohe Effizienz bei der Findung von Hittripeln erzielt werden kann. Durch die Implementierung mit OpenCL ist es möglich sowohl CPUs als auch GPUs zur Ausführung zu nutzen. Die Vorteile der GPU zeigen sich vor allem bei der *gleichzeitigen* Verarbeitung vieler Ereignisse mit *starker* Aktivität. Ist dies gegeben, so erzielt die Grafikkarte gegenüber der CPU einen Geschwindigkeitsvorteil von einem Faktor 64. Wie Abbildung 0.3 zeigt, wird dieser Faktor auch gegenüber der aktuellen Experimentsoftware erreicht.

Daher eignet sich der entworfene Algorithmus, den bevorstehenden Herausforderungen bei der Rekonstruktion von LHC-Daten ab 2015 entgegenzutreten.

Motivation

Scientific endeavor is one of mankind's noblest traits. For millennia, curiosity drove humans to ponder upon the workings of the universe, explore even the most remote places and devise intricate experiments to scrutinize their hypotheses about nature. The Large Hadron Collider (LHC) at the Organisation Européenne pour la Recherche Nucléaire (CERN) is one of the largest scientific apparatus ever built. As the world's most-powerful particle accelerator, it allows to study the fundamental constituents of matter at unprecedented energies. The proton-proton collisions are observed with the most sophisticated particle detectors conceived to date. The Compact Muon Solenoid (CMS) detector – one of the two general-purpose experiments at the LHC – allows the tracking of particles to highest precisions. The interactions of traversing particles with the detector material are measured by 16 588 detector modules in 75 million individual readout channels. Reconstructing paths of physical objects from this immense amount of data poses a tremendous computational challenge. Operations were suspended at the LHC in early 2013 to upgrade the machine to even higher energies and beam intensities. When experiments resume in 2015, this will result in more particle interactions, hence further increasing the computational load.

The situation is aggravated by the stagnating processor clock speeds of recent years. Instead, increases in computing capability have been due to multiple cores and vector units. Additionally, graphics cards, suitable for general-purpose computation, have been introduced to high performance computing. These new technologies can only be exploited by algorithms specifically tailored towards massive data-parallelism. To cope with the data volumes anticipated for 2015, track reconstruction methods adhering to these requirements must be examined. In Cellular Automaton-based track finding, particle tracks are reconstructed by identifying compatible triplets of measurements through simple and local computations and therefore being well-suited for data-parallel execution.

The present thesis examines suitable methods to produce *valid* triplets of measurements, i. e. measurements originating from the same particle's interaction with the detector material. To lower the combinatoric complexity of finding triplets, the physical properties of a particle's path should be exploited to predict search ranges based upon a given individual measurement or pair of measurements. Furthermore, efficient spatial data structures should be used, pertinent to the required range queries and parallel execution. The triplet finding should exploit *all* available parallelism in a *portable* manner, due to the diversity of the LHC computing infrastructure.

Outline The thesis is structured as follows: Part I introduces the LHC particle accelerator and CMS detector (Chapter 1) as well as the currently employed algorithms for particle track reconstruction (Chapter 2). The following Chapters 3 through 5 outline the foundations of parallel computing, Cellular Automaton-based track finding and spatial data structures. The devised parallel triplet finding algorithm is presented in Part II, Chapter 6 et seq. Following the discussion of properties of valid triplets of measurements in Chapter 7, the employed data structures and algorithms are detailed in Chapters 8 and 9, respectively. The second part is concluded by an evaluation of the achieved data quality and runtime (Chapter 10) and a summary of the findings (Chapter 11).

Contents

I Foundations **7**

1 The Large Hadron Collider and CMS Detector at CERN **9**
- 1.1 Large Hadron Collider . . . 10
- 1.2 The CMS Detector . . . 12
  - 1.2.1 Coordinate System . . . 12
  - 1.2.2 Inner Tracking System . . . 13
  - 1.2.3 Calorimeters and Muon Chambers . . . 15
- 1.3 Luminosity and Pile-Up Events . . . 17

2 CMS Event Reconstruction **19**
- 2.1 The World-wide LHC Computing Grid . . . 19
- 2.2 Software Framework CMSSW . . . 21
- 2.3 Event Processing . . . 22
  - 2.3.1 Trigger and Data Acquisition . . . 23
  - 2.3.2 Reconstruction of Physical Objects . . . 24
  - 2.3.3 Iterative Kalman Filter-based Track Finding . . . 26
  - 2.3.4 Triplet Seeding . . . 29
- 2.4 Event Generation . . . 33

3 Parallel Computing **35**
- 3.1 CPU Technologies . . . 36
- 3.2 General-Purpose-GPUs . . . 38
- 3.3 OpenCL . . . 40
- 3.4 Performance Metrics . . . 44

4 Cellular Automata **47**
- 4.1 Cellular Automata for Track Finding . . . 48
- 4.2 CA-based Track Finding in CMS . . . 52

5 Spatial Data Structures **55**
- 5.1 CMSSW Triplet Seeding – ϕ-sorted List . . . 55
- 5.2 k-d Tree . . . 56
- 5.3 Quadtree . . . 57
- 5.4 R-Tree . . . 58
- 5.5 Uniform Grid . . . 59

II Parallel Triplet Finding with OpenCL 61

6 Overview 63

7 Filter Criteria 67
7.1 Angular Constraints . . . 67
7.2 Transverse Impact Parameter Constraint . . . 69

8 Data Structures 73
8.1 Detector Geometry . . . 73
8.2 Event Hit Data . . . 75

9 Algorithms 81
9.1 Two-pass Algorithms . . . 81
9.2 Prefix Sum Algorithm . . . 83
9.3 Pair Building . . . 86
    9.3.1 Grid-based Pair Building . . . 87
    9.3.2 Prediction-based Pair Building . . . 89
    9.3.3 Implementation Details . . . 91
9.4 Triplet Prediction . . . 93
    9.4.1 Angular-based Prediction . . . 93
    9.4.2 Extrapolation-based Prediction . . . 95
    9.4.3 Implementation Details . . . 97
9.5 Triplet Filtering . . . 99

10 Evaluation 101
10.1 Evaluation Setup . . . 101
    10.1.1 Simulated Events . . . 101
    10.1.2 Hardware and Software Configuration . . . 102
10.2 Physics Performance . . . 104
    10.2.1 Determination of Cutoff Values for Filter Criteria . . . 104
    10.2.2 Physics Performance for QCD Events . . . 108
    10.2.3 Physics Performance for $t\bar{t}$ Events . . . 110
    10.2.4 Physics Performance for Muon Events . . . 110
10.3 Algorithmic properties . . . 112
    10.3.1 Work-Group Size . . . 113
    10.3.2 Concurrent Events . . . 115
    10.3.3 Grid Granularity . . . 116
    10.3.4 Runtime Composition . . . 117
    10.3.5 Tracks per Event . . . 118

11 Conclusion 121

III Appendices **123**

A Supplement to Physics Performance 125

B Supplement to Algorithmic Performance 129

C Configuration Parameters 141

D List of Figures 143

E List of Tables 147

F Acronyms 151

G Bibliography 155

Part I.

Foundations

Chapter 1

The Large Hadron Collider and CMS Detector at CERN

The Organisation Européenne pour la Recherche Nucléaire (CERN) was founded in 1954 to foster the research of the fundamental constituents of matter in Europe [142]. Originally established by 12 European countries, the number of member states grew to 20 by 1999 [65]. Several particle accelerators were installed at CERN's site over the years, including the original linear accelerator LINAC 1[1], the Low Energy Antiproton Ring (LEAR)[2] and the Large Electron-Positron Collider (LEP)[3], which was installed in the tunnel now occupied by the Large Hadron Collider (LHC). Today, six accelerators are operating at CERN in an interlinked manner, each increasing the energy of the accelerated particles before delivering them to either experiments or the next acceleration stage, refer to Figure 1.1. The numerous experiments operating at CERN's acceleration facilities made large contributions to mankind's knowledge about the particles of the universe. The discovery of W and Z bosons by the UA1 and UA2 experiments in 1983 [176, 177] greatly advanced the understanding of the weak force, one of the four fundamental forces governing the behavior of matter. The Nobel Prize in physics was awarded to the discovery in 1984. In 1989, experiments at LEP established the existence of three lepton families [71] and in 1995 antihydrogen atoms were first created at the PS210 experiment [18] and have been further studied ever since [8]. In 2009, the LHC started its operation as the world's largest and most powerful particle accelerator [38]. Section 1.1 gives an overview of the accelerator, its scientific program and encompassed experiments. One of the experiments, the Compact Muon Solenoid (CMS), is described in more detail in Section 1.2.

Besides contributions to the physics community, scientists at CERN also contributed to computer science. By proposing a project called ENQUIRE, Berners-Lee [25] started the development of the World Wide Web, which greatly influences everybody's life today. Furthermore, CERN became a center for grid computing to cope with the tremendous amount of data produces by the LHC. The data processing at the LHC is discussed in Chapter 2.

---

[1] 1958-1992

[2] 1982-1996

[3] 1989-2000

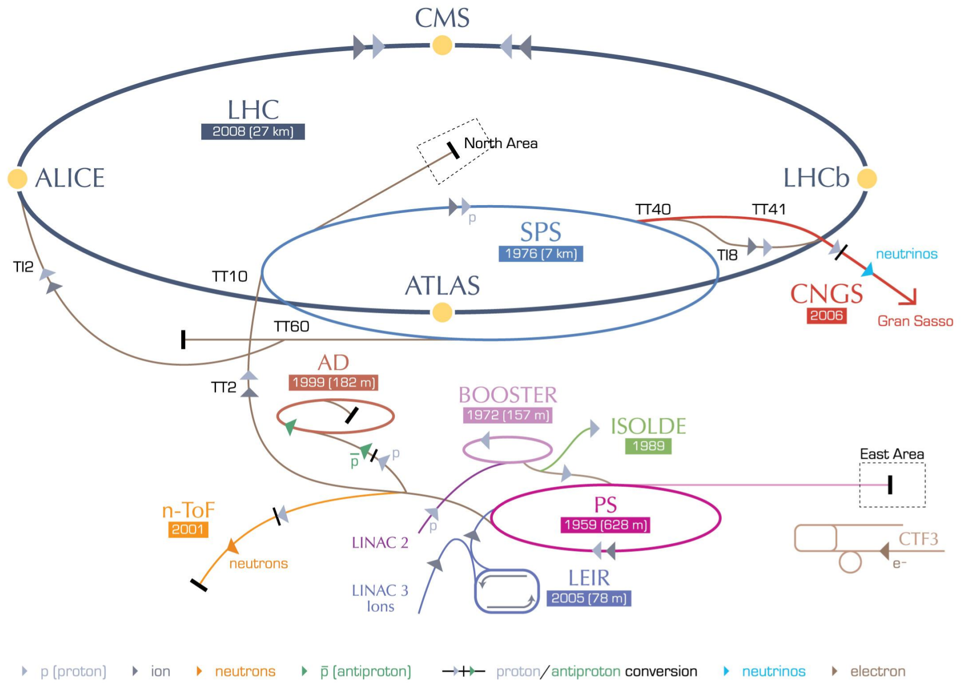

CMS
LHC
2008 (27 km)
North Area
ALICE
LHCb
TT40
TT41
p
SPS
1976 (7 km)
TI8
neutrinos
TI2
TT10
ATLAS
CNGS
2006
Gran Sasso
TT60
AD
1999 (182 m)
TT2
BOOSTER
1972 (157 m)
ISOLDE
1989
p̄
p
East Area
n-ToF
2001
PS
1959 (628 m)
p
CTF3
e–
LINAC 2
neutrons
LINAC 3
Ions
LEIR
2005 (78 m)
p (proton)
ion
neutrons
p̄ (antiproton)
proton/antiproton conversion
neutrinos
electron

Figure 1.1.: Overview of particle acceleration facilities and experiments at LHC [116].

1.1. Large Hadron Collider

The LHC was designed with a diverse scientific program in mind [38, 113], including

- validation and refined measurement of the standard model of particle physics,
- search for the Higgs boson, which lets particles interact with the Higgs field, thus giving them mass,
- unification of gravity with the other fundamental forces via supersymmetric particles,
- exploration of dark matter and dark energy, accounting for 94 % of the universe
- scrutinizing the matter-antimatter imbalance in the universe,
- study of the quark-gluon plasma to gain insights into the first microseconds after the universe's birth.

To be able to address this program 10 000 scientists and engineers from over 100 countries built an particle accelerator ring of 27 km circumference near Geneva, Switzerland. Located in a tunnel buried as deep as 175 m beneath the Franco-Swiss border, the LHC accelerates two proton beams in opposing direction of travel via 16 high frequency cavity resonators to up to 14 TeV center-of-mass energy [51]. The beams are controlled via 9 593 superconducting magnets to keep the protons on the ring trajectory and focus them as tightly as possible – 16.7 µm nominal transverse beam size [31] – for a high chance of head-on collisions at the four intersection points of the beam pipes. Due to space constraints in the tunnel, the magnets adopt a twin-bore design, accommodating both beam pipes within the same cold mass and cryostat [51]. Protons circulate the ring in 2 808 *bunches* of 1.15×10^{11} protons each with a design spacing of 25 ns between bunches. Thus, bunches collide at a rate of 40 MHz at the beam pipe's crossing points, referred to as bunch crossing. Each interaction point is equipped with a highly sophisticated particle detector, refer to Figure 1.1. Two of them, A Toroidal LHC Apparatus (ATLAS) and CMS, are large, general-purpose detectors designed for a broad range of physics analyses. In a major breakthrough in 2012, both collaborations announced the discovery of a new „higgs-like“ boson [14, 43]. The further detectors, LHC-beauty (LHCb) and A Large Ion Collider Experiment (ALICE), are specifically tailored towards the study of bottom quarks physics and the quark-gluon plasma, respectively. The latter is studied in the Heavy Ion (HI) mode of the LHC, colliding stripped lead ions instead of protons at 2.76 TeV center-of-mass energy [51].

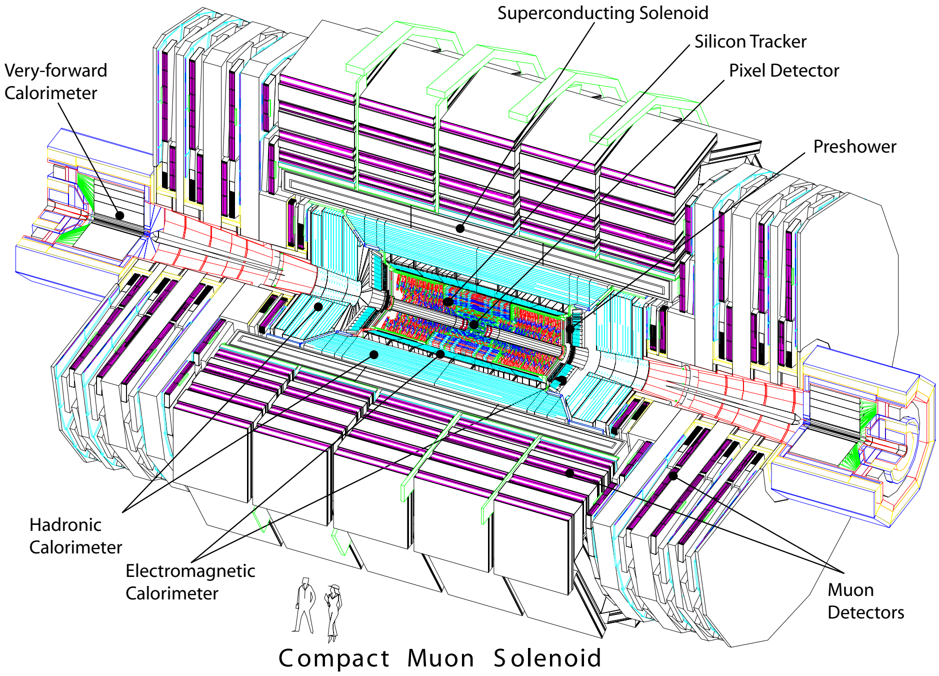

Superconducting Solenoid
Silicon Tracker
Pixel Detector
Very-forward Calorimeter
Preshower
Hadronic Calorimeter
Electromagnetic Calorimeter
Muon Detectors
Compact Muon Solenoid

Figure 1.2.: Schematic of the CMS detector [44].

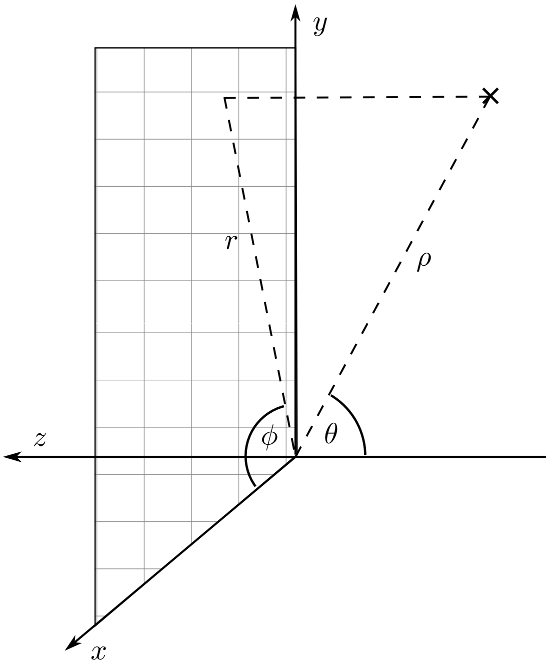


Figure 1.3.: Coordinate system as employed by the CMS collaboration.

1.2. The CMS Detector

More than 3 000 scientists from 38 countries worldwide participate in the CMS collaboration, responsible for the design, manufacturing and operation of the CMS detector [115]. As a general-purpose detector it must cater to a wide range of physical analyses, hence a multitude of performance requirements have been identified, such as high precision mass, momentum and missing energy resolution [44]. To cope with these requirements, a 21.6 m long, cylindrical detector has been constructed with 14.6 m diameter and a total weight of 12 500 t. It is located in a cavern 100 m beneath the French village of Cessy. Figure 1.2 illustrates the major components of the CMS detector, which are built around the central, 12.5 m long, superconducting solenoid with an inner-diameter of 6 m (ibid.). The solenoid creates a magnetic field of 3.8 T in its inner volume, which is required to bend the path of charged particles for precise charge and momentum measurements. The 2158 turns of niobium-titanium wire are cooled to 4 K to exploit their superconducting properties. The solenoid is surrounded by a 10 000 t iron yoke, which returns the magnetic flux to the inner detector. The particle detector components of CMS are detailed in the subsequent sections, with a focus on the inner tracking system. The following descriptions are based on the elaborate account by the CMS Collaboration [44] if not otherwise stated.

1.2.1. Coordinate System

The nominal Interaction Point (IP) corresponds to the origin of the CMS coordinate system and the center of the detector. The z-axis is defined along the the beam line

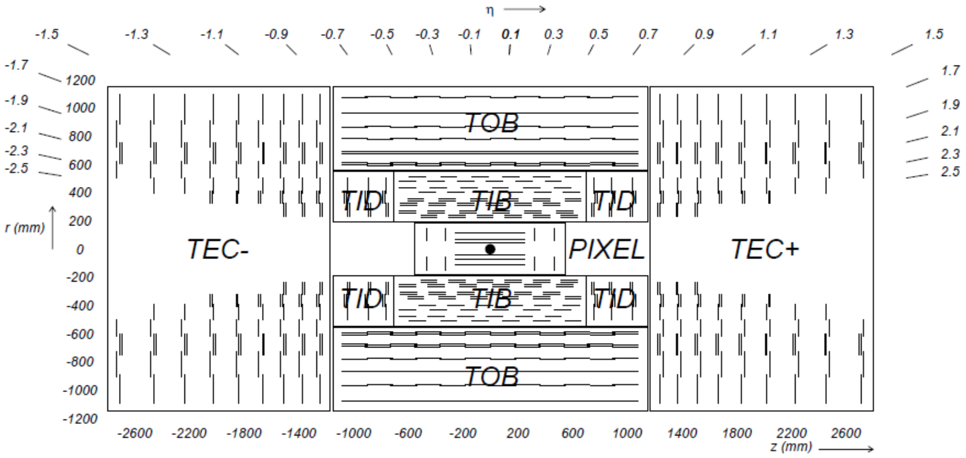

η
-1.5
-1.3
-1.1
-0.9
-0.7
-0.5
-0.3
-0.1
0.1
0.3
0.5
0.7
0.9
1.1
1.3
1.5
-1.7
-1.9
-2.1
-2.3
-2.5
1.7
1.9
2.1
2.3
2.5
r (mm)
1200
1000
800
600
400
200
0
-200
-400
-600
-800
-1000
-1200
TOB
TID
TIB
TID
TEC-
PIXEL
TEC+
TID
TIB
TID
TOB
-2600
-2200
-1800
-1400
-1000
-600
-200
200
600
1000
1400
1800
2200
2600
z (mm)

Figure 1.4.: Major components of the CMS silicon tracker [44]. The pixel barrel and endcap detectors are not labeled individually.

in anti-clockwise direction, with the x- and y-axis spanning the perpendicular plane in a right-handed coordinate system, i. e. y points towards the surface and x towards the center of the LHC ring, refer to Figure 1.3. In the spherical coordinate system, the xy-plane is referred to as transverse plane, or r-ϕ-plane, and defines the azimuthal angle ϕ with respect to the positive x-axis. The polar angle θ is defined with respect to the positive z-axis in the longitudinal plane, also referred to as r-z-plane. The radius of a point is defined twofold: its radius in the sphere is referred to as ρ, whereas r denotes the radius in the transverse plane. Instead of the polar angle θ, the *pseudorapidity* η is often used to describe a particles relation to the beam pipe:

$$\eta = -\ln \left[\tan \frac{\theta}{2} \right] .$$

1.2.2. Inner Tracking System

As Figure 1.2 depicts, the tracker is the inner-most sub-detector and thus experiences the highest particle flux of up to 1 000 particles per interaction. This entails the need for high precision spatial measurements and radiation-resistant components. The tracker is entirely silicon-based and extends to a length of 5.8 m and a radius of 1.25 m, covering an area of $200\,\mathrm{m}^2$, rendering it the largest silicon tracker built to date [154]. These dimensions allow for the precise tracking of charged particles with $|\eta| < 2.5$.

A charged particle traversing the tracker induces small ionization currents in the silicon-based semi-conductors, which are amplified and measured by read-out chips. The point of interaction between the particle and the detector module is referred to as *hit*. As the tracker is contained within the solenoid, charged particle tracks are bent due to the magnetic field. The curvature of the track is a direct measure for the transverse momentum p_T of the particle.

Barrel			**Endcap**		
Sub-Detector	**Layer**	**r** [cm]	**Sub-Detector**	**Layer**	**±z** [cm]
Pixel Barrel (PXB)	1	4.4	Pixel Forward (PXF)	1	34.5
	2	7.3		2	46.5
	3	10.2			
Tracker Inner Barrel (TIB)	4	25.5	Tracker Inner Disk (TID)	3	78.8
	5	39.9		4	91.8
	6	41.9		5	104.7
	7	49.8			
Tracker Outer Barrel (TOB)	8	60.8	Tracker End Cap (TEC)	6	136.0
	9	69.2		7	150.0
	10	78.0		8	164.0
	11	86.8		9	178.0
	12	96.5		10	192.0
	13	108.0		11	209.5
				12	228.5
				13	249.0
				14	270.5

Table 1.1.: Sub-detectors of the the CMS tracker with their abbreviations and comprised layers [44, 169]. The given measurements are averages of each layer.

The tracker is partitioned into six sub-detectors, each containing several layers of detector modules, and is presented pictorially in Figure 1.4. Table 1.1 introduces the abbreviations used to refer to the sub-detectors and provides details of the layer configuration. The three innermost barrel (PXB) and two innermost endcap (PXF) layers consist of 1440 silicon pixel detector modules with a total of 6.6×10^7 pixels, covering an area of $1\,\mathrm{m}^2$. Pixel detectors provide three-dimensional measurements with a spatial resolution of 10 µm in the transverse plane and 20 µm in the longitudinal plane.

Adjacent to the pixel detectors are 10 and 12 silicon strip detector layers in the barrel and endcap, respectively. Strips vary in length between 85.2 mm to 201.8 mm and are mounted parallel to the z-axis in the barrel and perpendicular to it in the endcaps. Thus, individual silicon strip detectors can only provide measurements of ϕ, with a resolution of 23 µm to 35 µm in the TIB/TID, 10 µm to 40 µm in the TOB and at least 53 µm in the TEC. To mitigate this shortcoming, double-sided modules are employed in the first two TIB and TOB layers, as well as the inner rings of TID and TEC, as depicted in Figure 1.5. Two silicon strip modules are mounted back-to-back with a stereo angle of 100 mrad, enabling the measurement of z with a resolution of 230 µm in the TIB and 530 µm in the TOB, as well as the measurement of r in the TID and TEC with varying resolution. In total 9.3×10^6 strips provide an active silicon area of $198\,\mathrm{m}^2$.

The measurements need to be amplified, buffered and digitized before further processing. In the pixel detector, 16 000 read-out chips, directly mounted to the detector modules, process the data of 53 rows · 52 columns of pixels each [100]. The read-out rate of 40 MHz corresponds to the design collision rate with the designated bunch spacing of 25 ns. In the strip detector, amplification is performed by 73 000 APV25 Application-specific

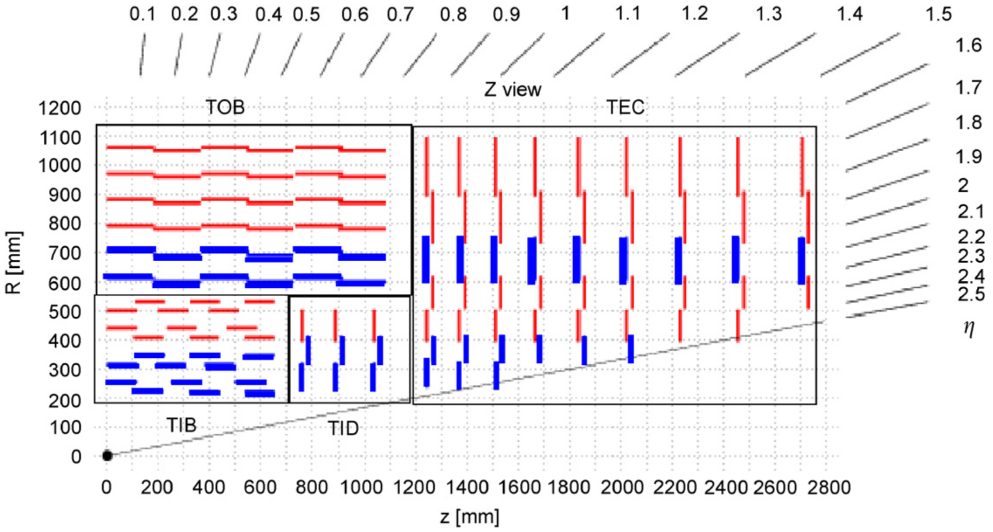


Figure 1.5.: One quarter of the CMS silicon strip tracker [108]. Red and blue lines represent single- and double-sided silicon strip modules, respectively.

Integrated Circuits (ASICs) utilizing radiation-hard 0.25 µm CMOS technology [56]. Each APV is capable of reading 128 silicon strips at 40 MHz sampling rate. Digitization is performed by Front End Driver (FED) modules, located in a separated room in the cavern. The data acquisition system of CMS and the further processing of the gathered data is described in Section 2.3.

1.2.3. Calorimeters and Muon Chambers

After traversing the tracker, a particle passes through further detection elements, as shown in Figure 1.6. The Electromagnetic Calorimeter (ECAL) is contiguous to the tracker and measures the energy of electrons and photons. It is a scintillation calorimeter, composed of 75 848 lead tungstate ($PbWO_4$) crystals, used as both absorber and scintillation material. An incoming electromagnetic particle creates a cascading shower of electrons, positrons and photons within the crystals, with the number of created photons being a direct measure for the incident particle's energy. The scintillation light is measured by avalanche photodiodes in the barrel and vacuum phototriodes in the endcap, the former being insensitive to axial magnetic fields and the latter withstanding the high radiation exposure. Lead tungstate was chosen due its short scintillation decay time. Within 25 ns, 80 % of all scintillation photons are emitted, aiding the separation of consecutive collisions.

Similar to the tracker, barrel and endcap are treated by different sub-detectors. The ECAL Barrel (EB) covers the pseudorapidity range $|\eta| < 1.479$, the ECAL Endcaps (EEs) detect particles with $|\eta| \in [1.479, 3.0]$. In addition, pre-shower detectors are mounted in front of the EE, consisting of lead absorbers and silicon detectors, to separate high energetic single photons from photon pairs originating in π^0 decays.

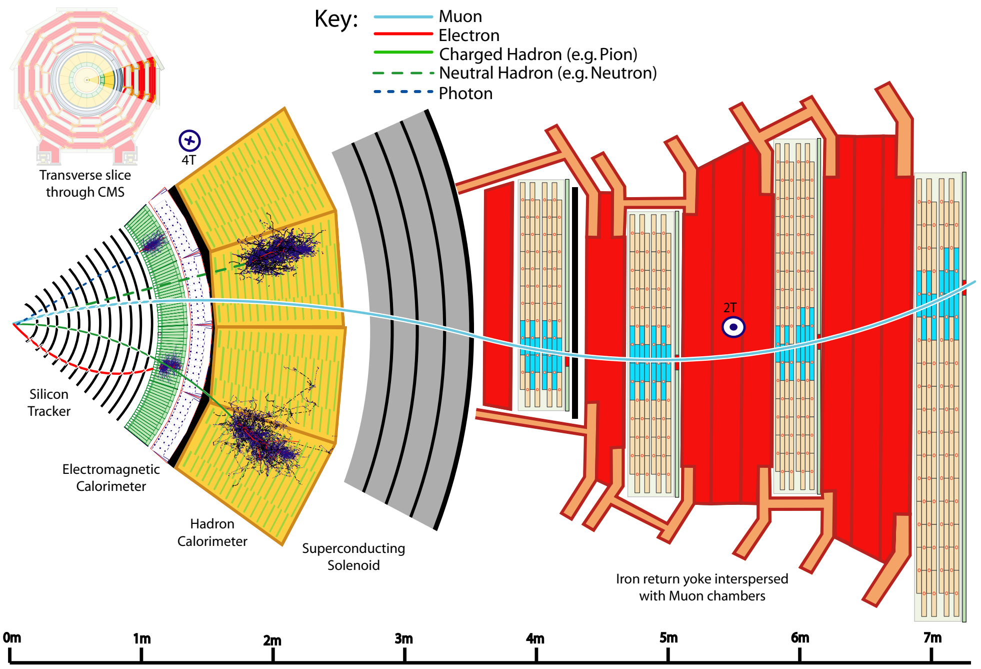

Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
Transverse slice through CMS
4T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
2T
Iron return yoke interspersed with Muon chambers
0m
1m
2m
3m
4m
5m
6m
7m

Figure 1.6.: Schematic CMS detector slice with passing particles [17]

The Hadronic Calorimeter (HCAL) surrounds the electromagnetic one and determines the energy of passing hadrons, such as protons and pions. As sampling calorimeter it is constructed of alternating brass absorber layers and plastic scintillators. An incident hadron induces a cascade of particle showers, which are mainly absorbed in the brass material and only a fraction being used for detection in the scintillation layers. Hybrid photodiodes, insensitive to axial magnetic fields, measure the scintillation light, which has been gathered by wavelength-shifting fibres.

Four sub-detectors constitute the HCAL. The HCAL Barrel (HB) and HCAL Endcaps (HEs) cover the pseudorapidity ranges of $|\eta| < 1.3$ and $|\eta| \in [1.3, 3.0]$, respectively. Both are contained within CMS's solenoid. To extend the absorbing capabilities of the calorimeter, an additional Outer HCAL (HO) is installed outside the solenoid as „tail-catcher“, covering the same η-range as the HB. Two Forward HCALs (HFs) complete the system, which are installed at $z = \pm 11.2\,\mathrm{m}$ outside the solenoid, accepting particles with $|\eta| \in [3.0, 5.0]$.

The muon detection system is integrated into the iron return yoke. Muon identification and momenta measurement are crucial for a wealth of physical analyses, including the study of the Higgs boson. Three sub-detectors constitute the muon system of CMS. In the barrel, covering $|\eta| < 1.2$, Drift Tubes (DTs) filled with 85 % argon and 15 % CO_2 detect passing muons via 172 000 wires in the gaseous volume. Cathode Strip Chambers (CSCs) are installed in the endcaps, accepting particles of $|\eta| \in [0.9, 2.4]$. The 468 CSCs are filled with a mixture of argon, CO_2 and CF_4 and contain 2×10^6 wires. Both DTs and CSCs provide a good spatial resolution, with CSCs operating reliably even in a spatially varying magnetic field. To compensate for the lower time resolution of both detector types compared to scintillators, Resistive Plate Chambers (RPCs) are installed in both barrel and endcap. Due to the fast response time of RPCs, their signal is used in the Level-1 triggering of CMS, refer to Section 2.3.

1.3. Luminosity and Pile-Up Events

The instantaneous luminosity is a characteristic quantity of a particle collider and is given by

$$L = \frac{N_a N_b f_{\text{rev}}}{A_{\text{eff}}} = \frac{N_a N_b f_{\text{rev}}}{4\pi \sigma_{T,a} \sigma_{T,b}},$$

with two beams of N_a and N_b particles respectively, colliding at a frequency of f_{rev} with an effective collision area A_{eff} [51]. Assuming a Gaussian beam distribution, the effective area is determined by the mean transverse beam sizes $\sigma_{T,\{a,b\}}$ of the beams. As the beams are not colliding precisely head-on but at an small crossing angle θ_c of about 200 µrad (ibid.), the luminosity must be reduced by

$$F = \frac{1}{\sqrt{1 + \frac{\theta_c \sigma_z^2}{2\sigma_T^2}}},$$

with mean longitudinal beam size σ_z and mean transverse beam size σ_T, assumed to be equal for both beams. Taking into account the relativistic gamma factor γ_{rev}, the instantaneous luminosity is given by

$$L = \frac{N_a N_b f_{\text{rev}} \gamma_{\text{rev}}}{4\pi \sigma_T^2} F.^{\dagger}$$

The probability for a specific physical process, e. g. a Higgs boson decaying into four muons, to happen during a proton-proton collision is referred to as production cross section σ_{prod}. Given the instantaneous luminosity L and cross section σ_{prod}, the number of generated events per second equals

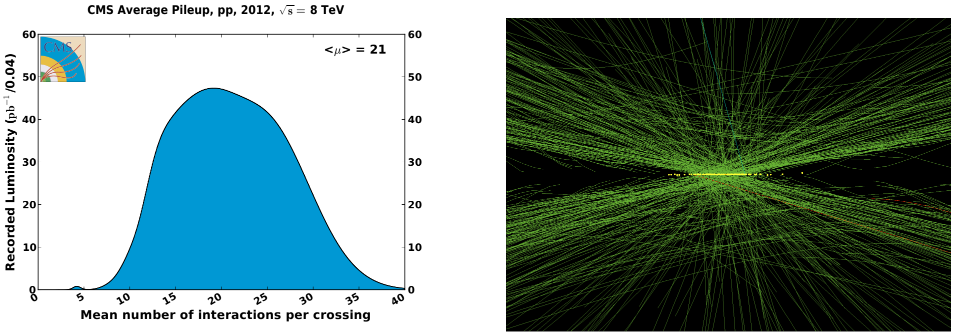
$$\dot{N}_{\text{event}} = L \cdot \sigma_{\text{prod}}.$$

Integrating the instantaneous luminosity over time yields the integrated luminosity $\mathcal{L}$, denoting the amount of accumulated collision data

$$N_{\text{event}} = \int L \sigma_{\text{prod}} dt = \mathcal{L} \cdot \sigma_{\text{prod}}.$$

[†] Evans and Bryant [51] characterize the quadratic mean transverse beam size σ_T^2 by the normalized transverse beam emittance ϵ_n and the beta function at the collision point β^*

$$\pi \sigma_T^2 = \epsilon_n \beta^*.$$



(a) Distribution of number of interactions per bunch crossing in 2012 [81].

(b) Event from 2012 high pile-up run with 78 reconstructed vertices [86].[5]

Figure 1.7.: Pile-up events in the CMS detector.

The instantaneous luminosity is *not* constant over time. During the time between injection and extraction of the proton beam – referred to as *fill* – the intensity and emittance of the beam deteriorates. Therefore, the instantaneous luminosity is measured every 23 s with four rings of the Forward HCAL [41]. During this time, denoted *luminosity section*, L is assumed to be constant. Data taking with a specific detector configuration is labeled *run*. A *fill* spans several *runs*, each containing many *luminosity sections* with approximately 9.2×10^8 *events* per luminosity section at design luminosity.

One recorded *event*, may contain energy deposits and interactions with the detector material of several collisions, denoted as *pile-up events*. Two kinds of pile-up need to be distinguished:

In-time pile-up is due to interactions within the same bunch crossing, hence it increases with the number of protons per bunch.

Out-of-time pile-up is caused by the finite time resolution of the CMS detector components. Energy deposits and observed interactions with the detector material from prior bunch crossings are still present in the detector at the time of the next event recording. Therefore, a smaller bunch spacing increases this effect.

In 2012, the bunches were separated by 50 ns, so out-of-time pile-up was minimized. As shown in Figure 1.7a, on average 21 interactions took place during one bunch crossing. The beam parameters foreseen for the upcoming data-taking period in 2015 will double the beam intensity and reduce the spacing between bunches to the design value of 25 ns [37]. During a high pile-up run in 2012, some of the resulting effects could be studied. Figure 1.7b depicts an event with 78 reconstructed vertices, which are proportional to the number of interactions in the event [72]. Such high occupancy does not only pose a challenge to the detector components but also to the entire event processing chain, particularly the particle track reconstruction.

---

[5] Run: 198609, Lumi: 56, Event: 3565522.

Chapter 2

CMS Event Reconstruction

The experiments at the LHC produce an enormous amount of data. At design luminosity, proton-proton collisions occur at a rate of 40 MHz in each of the detectors, generating 40 TByte s^{-1} of data [26]. It is neither feasible nor desirable to process every event as only a minuscule fraction of events, approximately 0.01 ‰, contain interesting physical processes [44]. To reduce the data rate, CMS employs a two-stage triggering system to identify events worth studying. The trigger system and the subsequent reconstruction of physical objects from the raw detector data is described in Section 2.3. The preceding Sections 2.1 and 2.2 elucidate the basis for event processing, the computing infrastructure and application framework, respectively. Section 2.4 concludes the chapter by discussing the simulation of particle interactions.

2.1. The World-wide LHC Computing Grid

All experiments at the LHC rely on an amply scaled, reliable computing infrastructure, catering to the following requirements [26]:

- storage of large data volumes, delivered at high data rates,
- long-term data archiving in a robust manner,
- data access for thousands of users and
- provision of raw computing capacity.

Facing 25 PByte of data a year [36] and 4 000 users from ATLAS and CMS alone, the multi-tier Worldwide LHC Computing Grid (WLCG) [49] was devised to cope with this tremendous challenge.

As Figure 2.1 illustrates, CERN's own Data Centre (DC) acts as *Tier 0*, primarily responsible for immediately archiving RAW data from the experiments at full data rate, ≈ 1.3 GByte s^{-1}, on tape [26]. To ensure reliability, the data is distributed to the *Tier 1* centers via the dedicated LHC Optical Private Network (LHCOPN) with 10 GByte s^{-1} bandwidth (ibid.). Furthermore, the Tier 0 is responsible for the prompt reconstruction of event data for data taking guidance as well as detector alignment and calibration. Coordinating the overall operation of the grid also falls within the Tier 0's responsibilities.

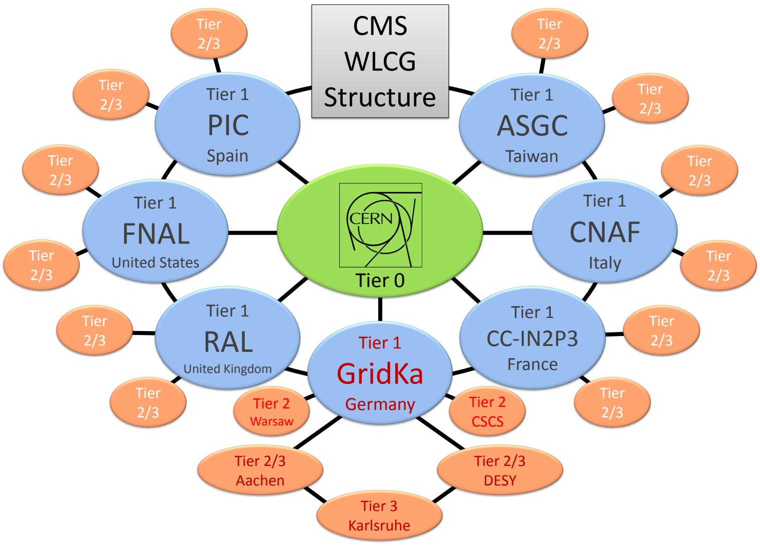

CMS
WLCG
Structure
Tier 2/3
Tier 1
PIC
Spain
Tier 1
ASGC
Taiwan
Tier 1
FNAL
United States
CERN
Tier 0
Tier 1
CNAF
Italy
Tier 1
RAL
United Kingdom
Tier 1
GridKa
Germany
Tier 1
CC-IN2P3
France
Tier 2
Warsaw
Tier 2
CSCS
Tier 2/3
Aachen
Tier 2/3
DESY
Tier 3
Karlsruhe

Figure 2.1.: Schematic view of the WLCG grid tier hierarchy including CMS Tier 1 sites and German CMS Tier 2 centers [156].

Currently, there are 30 PByte of disk and 100 PByte of tape storage as well as 65 000 processing cores installed at CERN, which will be augmented by 5.5 PByte disk storage and 20 000 cores at a secondary Tier 0 location at the Wigner Research Centre for Physics in Budapest, Hungary [35].

Large-scale organized analyses are mainly performed at Tier 1 sites [26]. Data archiving aside, these centers distribute data to the connected Tier 2 sites. Tier 1 centers are primarily large-scale national computing centers supporting several experiments. Seven of the 11 Tier 1 sites provide services to the CMS collaboration. *Tier 2* sites provide storage and computing resources and are accessible to regular grid users for job submission. In addition to the reconstruction of data, these sites also serve the generation and simulation of Monte Carlo (MC) events, which are crucial to relate the obtained experimental measurements to the underlying theory under scrutiny [130]. End-user analysis is performed on *Tier 3* sites, which are connected to a Tier 2 center for data access. These sites cover a wide range of data center sizes, ranging from institute local clusters to national facilities.

A grid, as a federation of computer resources, is, unlike other forms of distributed systems such as clusters, *not* centrally administered [55]. Thus, it features a heterogeneous computing environment. The WLCG is governed by a Memorandum of Understanding (MoU) [181] between CERN and the national funding agencies. It prescribes which services a computing center participating in the WLCG has to offer and defines service levels to be fulfilled. However, the MoU does *not* regulate hardware specifics, hence a variety of components can be found in different WLCG sites, ranging from hardware featuring the latest technology to rather geriatric equipment. Software for the WLCG needs to be designed with this heterogeneity in mind, being easily *deployable* and *performant* on diverse computer systems. As of 2013, the lowest common denominator for all WLCG sites is the use of x86-based processors operating under Scientific Linux CERN (SLC) 5.0 [124].

2.2. Software Framework CMSSW

The CMS Software Framework (CMSSW) is the primary tool for the processing and analysis of events recorded with the CMS detector [44]. The modular C++ framework is centered around the Event Data Model (EDM) and is used for both online and offline processing [19]. The former addresses the High Level Trigger (HLT) and prompt event reconstruction (refer to Section 2.3), whereas the latter is devoted to the meticulous re-reconstruction of recorded data. The development of the application was guided by the following design goals [93]:

Utilization of a clear data model Modules communicate only through a central `Event` container. The container stores data produced by a module and provides access to it for subsequent ones. Each datum is uniquely identified by its data type and a combination of module label and instance label, as defined by the producing module.

Provenance tracking for all data All algorithms involved in the production of a particular datum must be traceable, including their specific configuration parameters.

Clear separation of concerns Modules should address a specific task and store all their data in the `Event` to allow for individual testing in isolation.

Provision of browsable ROOT files ROOT is the standard tool in High-Energy Physics (HEP) for histogram production, data fitting and visualization [30]. Providing output files directly usable in ROOT eases the process of output validation for CMSSW modules.

CMSSW defines five module types [93]. *Sources* provide the initial `Event` instance to the process, either reading event data from disk or directly accessing the Data Acquisition (DAQ) system. *Producers* generate new data to be stored in the `Event`, based upon already existing information. For instance, compatible hits in adjacent detector layers, as stored in the `Event`, are combined to form triplets, which are later used as seeds for the track finding routines. *Filters* decide whether a given event should be further processed or discarded, based upon specific criteria such as contained particle types or minimum p_T requirements. *Analyzers* are read-only modules, studying particular properties of the event, e. g. p_T distribution for specific particles. The product of the analysis is usually written to disk in a separate ROOT file. *Output modules* store the data collected in the `Event` container on disk.

The individual modules are orchestrated into a process via a Python-based configuration file [179]. Figure 2.2 shows a sample process with a execution schedule comprising two sequences, each containing several modules. A module specification consists of a C++

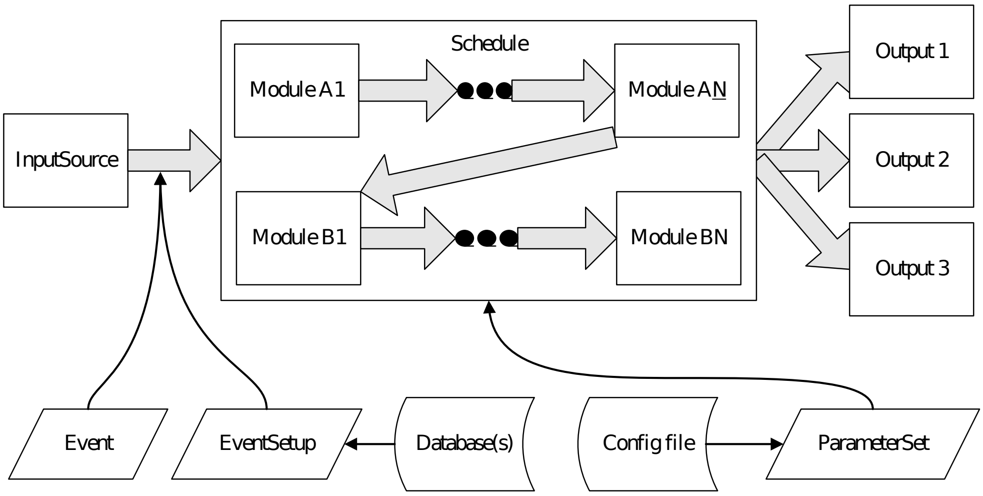


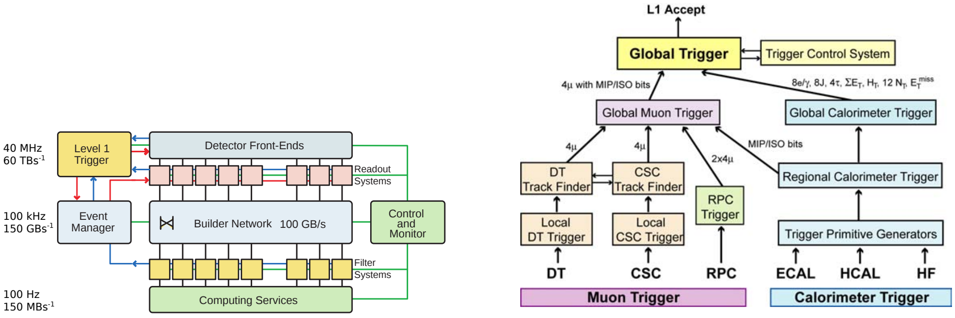
Figure 2.2.: CMSSW process containing a path with two sequences, each containing several modules [44].

class, that implements the desired function and inherits from an appropriate CMSSW superclass, e. g. `edm::EDAnalyzer`, and a `ParameterSet`, configuring the parameters of the algorithm, e. g. minimum p_T. These module specifications can be arranged in `Sequences` and `Paths`. Whereas `Sequences` are merely an organizational tool to ease the reuse of predefined sequences, `Paths` define the actual execution order – `Schedule` – of the process. Even though dependencies between modules may be stated in the `Path` specification, there is only a sequential processing stream in CMSSW at the moment. The mitigation of this shortcoming is currently under active research and is introduced in Section 3.1.

There is a wealth of predefined sequences in CMSSW, which can be reused and customized by users in their own analyses. The following section introduces the standard reconstruction sequence.

2.3. Event Processing

The reconstruction of physical objects from the raw detector data is a most intricate task. Hundreds of steps are required to transition from individual energy deposits in the detector material to the comprehensive description of a particle's trajectory, energy and momenta. First, Section 2.3.1 elaborates on the CMS trigger system, which reduces the event rate from 40 MHz down to manageable 100 Hz. Sections 2.3.2 to 2.3.4 outline the reconstruction sequence with a focus on Kalman filter-based track finding and triplet seeding. A comprehensive account of the entire reconstruction process is given by Acosta et al. [2].


40 MHz
60 TBs^{-1}
Level 1 Trigger
Detector Front-Ends
Readout Systems
100 kHz
150 GBs^{-1}
Event Manager
Builder Network 100 GB/s
Control and Monitor
Filter Systems
100 Hz
150 MBs^{-1}
Computing Services
L1 Accept
Global Trigger
Trigger Control System
4μ with MIP/ISO bits
8e/γ, 8J, 4τ, ΣE_T, H_T, 12 N_T, E_T^{miss}
Global Muon Trigger
Global Calorimeter Trigger
4μ
4μ
2x4μ
MIP/ISO bits
DT Track Finder
CSC Track Finder
RPC Trigger
Regional Calorimeter Trigger
Local DT Trigger
Local CSC Trigger
Trigger Primitive Generators
DT
CSC
RPC
ECAL
HCAL
HF
Muon Trigger
Calorimeter Trigger

(a) Components of the DAQ system with design event and data rates.

(b) Level-1 Trigger decision based upon calorimeter and muon triggers.

Figure 2.3.: Schematics of the CMS trigger and data acquisition system [44].

2.3.1. Trigger and Data Acquisition

The trigger system comprises two levels [44]. The Level-1 trigger analyzes *every* bunch crossing. At an event rate of 40 MHz, a complete reconstruction is not possible. Thus, the decision whether an event contains interesting characteristics must be based on a subset of the available information. As depicted in Figure 2.3b, the calorimeter and muon system are responsible for the L1 triggering. The complete detector readout is buffered in the FED, before a L1 decision is reached. As the L1 must determine the relevance of an event within 3.2 µs, L1 algorithms are implemented in hardware. Field-programmable Gate Arrays (FPGAs) are the preferred hardware technology, due to their flexibility, but speed and radiation requirements demand for some components to be implemented as ASICs (ibid.).

Given a positive L1 decision, the detector readout is moved from the FED buffers to the High Level Trigger (HLT), as depicted in Figure 2.3a. The design event rate for the Level-1 trigger is 100 kHz, which equals a data rate of 150 GByte s^{-1}, given an event size of approximately 1.5 MByte/event (ibid.). Whereas the L1 trigger is implemented in hardware and located in a control room next to the detector cavern, the HLT is performed on a dedicated commodity computer farm located on the surface with currently 13 200 Central Processing Unit (CPU) cores [21]. The HLT algorithms are similar to the ones employed in the offline reconstruction, therefore trigger decisions can be made upon physical objects in the event. Being implemented in software, the HLT offers great flexibility to treat different kind of physical processes. A set of trigger criteria is referred to as HLT menu and can be adapted to the needs of manifold analyses [4]. About 400 individual trigger algorithms contribute to the HLT decision, requiring a total runtime of $\approx$ 165 ms/event [21]. Due to runtime limitations, some algorithms are simplified in comparison with the offline reconstruction or use tighter constraints and boundary conditions [2].

The design event rate for the HLT was originally set to 100 Hz. Due to advances in storage technology, it currently operates with 300 Hz to 350 Hz event rate, resulting in approximately 500 MByte s^{-1} data rate.

For underlying event studies, trigger calibration and impartial physical exploration, CMS additionally records zero- and minimum-bias events, with no or only a minimal selection of L1 and HLT triggers, respectively [39, 167].

2.3.2. Reconstruction of Physical Objects

In the reconstruction, RAW detector readouts obtained from the DAQ, called `Digis`, are incrementally processed into higher-level physical objects [2]. Each step builds upon the objects produced by previous ones and is further influenced by

EventSetup Non-event data such as geometrical description of the detector, alignment and calibration data, as well as the magnetic field conditions. These information are not tied to an event but have a specific Interval of Validity (IOV), spanning many events. The `EventSetup` provides the appropriate data for a given `Event` as non-event data can have varying IOVs;

ParameterSet As described in Section 2.2, this data provides the configuration parameters of a module's algorithm.

The reconstruction consists of three coarse-grained steps

1. *local* reconstruction within a detector module, e. g. a pixel detector module,
2. *global* reconstruction within a sub-detector, e. g. the tracker or ECAL,[1]
3. *combined* reconstruction, aggregating the information from all sub-detectors.

The following outline of these three steps, with a focus on the tracker, is based on the elaborate description by Acosta et al. [2] if not otherwise stated. Figure 2.4 illustrates the process and the products of each step.

Local Reconstruction uses only information from one detector module and processes the digitized readout, `Digis`, to reconstructed hits, `RecHits`, including spatial information about a particle's interaction with the detector and the amount of deposited energy. The specifics vary between the sub-detectors, e. g. local reconstruction already produces track segments within the superlayers of the muon DTs. In the tracker, different algorithms are employed for pixel and silicon strip detector.

The FEDs of the pixel detector produce a zero-suppressed readout with adjustable threshold charge – currently $3\,200\,e$ [162]. In the local reconstruction, adjacent pixels are formed to clusters with a minimum charge of $4\,000\,e$. The cluster's position and uncertainty is then determined in the local coordinate frame of the sensor (u, v) by the cluster's centroid, corrected for the Lorentz shift. In a second approach, the observed cluster charge distribution is compared to *templates*, expected cluster shapes for particles passing the detector module at various angles of incidence, obtained from MC studies.

---

[1] The name is somewhat misleading, as it suggests that information from the *entire* detector is used.

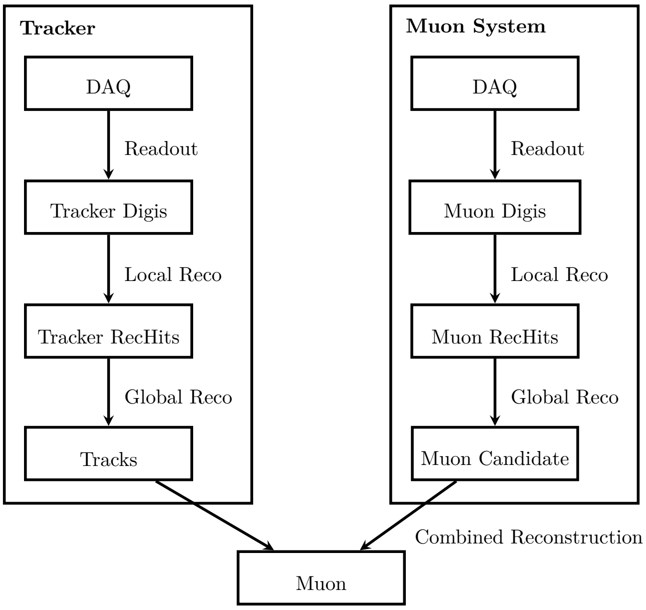


Figure 2.4.: Muon reconstruction steps in CMS [adapted from 2]. The particle is treated independently in the tracker and muon system during the local and global reconstruction. Only in the combined reconstruction, high-level physical objects with aggregated information from all sub-detectors are created.

In the silicon strip detector, strips pass the zero-suppression if their charge exceeds five times the channel noise or if a strip and one of its neighbors both exceed two times the channel noise. Clusters are formed starting from `Digis` exceeding three times the channel noise. Neighboring strips are added to the cluster, if they exceed two times the strip noise. The final cluster must exceed five times the cluster noise to be accepted for further processing. The charge-weighted average of the strip positions, corrected for the Lorentz shift, yields the cluster's position (ibid.). For double-sided modules, the two two-dimensional hits from the r-ϕ-plane and the stereo-plane, rotated by 100 mrad, are matched to three-dimensional `MatchedRecHits`.

Global Reconstruction is performed on all reconstructed hits from the same sub-detector to form higher-level physical objects. In the tracker, compatible hits are combined to `Tracks` in an iterative approach. The discussion of track finding is deferred to Section 2.3.3. Further objects of interest in the tracker's global reconstruction are the primary vertices of the event, which correspond to the locations of proton-proton interactions in the bunch crossing. The number of primary vertices is thus proportional to the pile-up of the event. The primary vertices are reconstructed from tracks originating close to the beam line in the transverse plane, i. e. with a small Transverse Impact Parameter (TIP). The selected tracks are clustered according to their z-coordinate at the point of closest approach to the Interaction Point (IP) by a deterministic annealing algorithm [148]. An adaptive vertex fitter is applied to the resulting clusters to obtain the vertex position and associated uncertainty [178].

In the calorimeters, matching clusters in the ECAL and HCAL are linked to form calorimetric towers – `CaloTowers` – with a definite position in the η-ϕ-plane.

Reconstruction in the muon system produces `MuonCandidates`, also referred to as „standalone muons“, since they are not linked to tracker hits. The algorithm uses locally reconstructed track segments of the innermost muon chambers as seeds and builds trajectories via a Kalman filter-based method in the radially-increasing direction. The propagation uses a detailed material and magnetic field map to account for the energy loss and bending of the muon's path. When the outermost muon chamber layer is reached, the track is propagated backwards to the inner layers and finally extrapolated to the nominal IP. Kalman filter-based track building is discussed with the tracker's global reconstruction in Section 2.3.3.

Combined Reconstruction combines the global tracking results from all sub-detectors of CMS to produce a number of high-level physics objects. Standalone muons are paired with a compatible `Track` in the tracker and refitted to obtain the final track parameters. If `CaloTowers` can be matched to tracks in the pixel detector they are identified as electron, otherwise they are flagged as photon. The hadronization of free quarks and gluons produces narrow cones of hadrons, referred to as jets. A wealth of algorithms for jet reconstruction exist [160], most relying on clustering calorimetric towers which are close in (η, ϕ) to a high transverse energy E_T tower.

2.3.3. Iterative Kalman Filter-based Track Finding

Goal of the track finding is to identify a sequence of hits, belonging to the trajectory of a charged particle among the set of all `RecHits` reconstructed in the tracker. The identified hit sequence is used to fit the track's parameters

$$\boldsymbol{\tau} = (d_0, z_0, \phi, \cot\theta, p_T)$$

at the point of closest approach to the beam line, referred to as impact point. The impact point is defined by the transverse coordinates (x_0, y_0), given by $d_0 = y_0 \cos\phi - x_0 \sin\phi$, and the longitudinal coordinate z_0. The momentum vector at the impact point is described by azimuthal angle ϕ and polar angle θ and its transverse magnitude p_T [2].

CMS uses a Combinatorial Track Finder (CTF). Given k detector layers with n_i, $i \in [1, k]$, hits in layer i, the number of possible hit combinations is given by

$$\prod_{i\in[1,k]} n_i \in \mathcal{O}(n^k).^{\dagger}$$

Restricting the physical properties of a track, such as transverse momentum p_T or transverse distance to the interaction point TIP, drastically reduces the number of possible hit combinations. In *iterative tracking*, early iterations search for tracks with high p_T and small TIP, which can be found with tight constraints on possible hit

---

[†] $f(n) \in \mathcal{O}(g(n)) \equiv f(n) \leq k \cdot g(n)$ for some positive k and sufficiently large n.

candidates [144]. Hits associated with a track are masked and no longer considered in subsequent iterations, thus reducing the combinatorial complexity and allowing the search for more difficult to find tracks. This class includes tracks with low p_T and tracks originating further away from the beam line, thus possibly not traversing all three pixel layers. Seven iterations are currently used in CMSSW track finding. As the parameters of an iteration mainly influence the seeding procedure, they are presented in Section 2.3.4.

Each iteration proceeds as follows [2]:

Seeding generates pairs or triplets of hits in the inner detector to provide an initial estimate of the track parameters.

Track Finding extrapolates the seed along the predicted path of the particle to assign additional hits to the track.

Track Fitting aims to provide the best possible estimate of the track parameters, based upon all assigned hits.

Track Selection examines quality criteria for the produced tracks, discarding insufficient ones and flagging tracks with particularly low errors on the track parameters as *high purity.*

Seeding is discussed in Section 2.3.4. Track finding and fitting are based on the *Kalman filter* technique [96, 97] used for estimating the state of a discrete linear dynamic system via a series of measurements. Frühwirth [59] is generally credited with establishing the Kalman filter for track finding and fitting in the HEP community. A track in space is regarded as dynamic system, described by a state vector $\boldsymbol{\tau}$, i. e. the track parameters. At each point in space $\mathbf{x}$, $\boldsymbol{\tau}(\mathbf{x})$ uniquely describes the trajectory of a particle. Considering the state vector only at discrete points $\mathbf{x}_k$, the intersection points of the particle's trajectory and the detector layers, leads to the following system equation

$$\boldsymbol{\tau}(\mathbf{x}_k) := \boldsymbol{\tau}_k = f_{k-1}(\boldsymbol{\tau}_{k-1}) + w_{k-1},$$

with f_{k-1} denoting the track propagator from detector layer $k-1$ to k and w_{k-1} describing disturbances during the traversal, such as *multiple scattering.* Multiple scattering is a stochastic effect due to non-uniformities in the traversed medium between two detector layers, repeatedly deflecting a particle from its original trajectory. The state vector can only be observed via measurements $\mathbf{m}_k$, suffering from measurement noise $\boldsymbol{\epsilon}_k$,

$$\mathbf{m}_k = h_k(\boldsymbol{\tau}_k) + \boldsymbol{\epsilon}_k.$$

If both f_k and h_k are linear, the Kalman filter is the optimal recursive state estimator [96]. However, in the presence of a magnetic field, the track propagator is non-linear. Frühwirth [59] notes, that the measurement equation can be linearized by an appropriate choice of state vector and the non-linear propagator f_k can be approximated by its first order Taylor expansion

$$\hat{f}_k(\boldsymbol{\tau}^*_k) = f_k(\boldsymbol{\tau}_k) + \frac{\partial f_k}{\partial \boldsymbol{\tau}_k}(\boldsymbol{\tau}^*_k - \boldsymbol{\tau}_k).$$

Given this system model, the following estimation operations can be applied:

- *Prediction* of the future state $\boldsymbol{\tau}_{k+1}$ given the current one $\boldsymbol{\tau}_k$.
- *Filtering* the present state $\boldsymbol{\tau}_k$ given all measurements $\mathbf{m}_1 \dots \mathbf{m}_k$.
- *Smoothing* a past state $\boldsymbol{\tau}_j$, $j < k$, with all measurements $\mathbf{m}_1 \dots \mathbf{m}_k$.

An elaborate presentation of the mathematical formulation of these operations, in particular the calculation of the covariance matrix and residuals, is given by Frühwirth and Regler [61].

Track Finding is initialized with the coarse track parameter estimation from the trajectory seed [162]. The trajectory is extrapolated from the seed's outermost hit to determine compatible adjacent detector layers, taking into account the current uncertainty of the track and the detector material to be crossed. While adjacency is well-defined in the barrel, the process is more involved in the endcap and barrel-endcap transition region. Given a (set of) compatible layer(s), suitable detector elements within these layers are searched. If the trajectory intercepts the detector surface within a configurable number of standard deviations, the detector element is considered compatible (ibid.). The positions and uncertainties of the hits on the selected detector elements are refined with the trajectory's direction on the detector surface to more accurately reflect the Lorentz shift. A hit's compatibility with the track is determined by a χ^2 test. The trajectory state is updated for *every* hit with χ^2 value below a configurable threshold. To limit the increase in track candidates, only a limited number of new trajectory states are chosen for further processing, determined by the best normalized χ^2 values. To account for detector inefficiencies, *invalid hits* may be introduced to the track if no suitable hit is found on a compatible detector element, resulting in a penalty to the χ^2 value.

The process is continued until the outermost layer is reached or a configurable number of hits have been added by the outward propagation. All assigned hits – except the seed – are used to fit the trajectory's parameters. Thereafter, starting at the innermost *non*-seed hit, the track is propagated inwards towards the beam line. Hereby, additional compatible hits in the seeding layers can be found and if not the innermost layers are used for seeding the track may be extended further inwards.

Track Fitting is a two-phase process to obtain the best possible track parameter estimation and eliminate any bias introduced to the track parameters by the constraints of the seeding and track building algorithms (ibid.). In the filtering phase, the Kalman filter is initialized at the innermost hit with the trajectory parameters obtained from seeding. The algorithm proceeds outwards, updating the trajectory estimate at each hit. Again, the hit positions and uncertainties are corrected for the Lorentz shift with the current track parameters. At a given detector layer i, the state estimate incorporates all measurements $\mathbf{m}_1 \dots \mathbf{m}_i$. In the smoothing phase, the Kalman filter is initialized with the result of the first phase at the outermost hit and propagated inwards to the beam line. The smoothed trajectory estimate at layer i can be obtained by averaging the track parameters from the first and second phase, as the second phase yields a state estimate at i with measurements $\mathbf{m}_i \dots \mathbf{m}_k$ for a k layer detector.

#	**Step**	**Seeding Layers**	$\mathbf{p_t}$ [GeV]	$\mathbf{d_0}$ [cm]	$\mathbf{z_0}$ [cm]
1	initial triplets	PXB, PXF	0.6	0.02	4.0 σ
2	low p_T triplets	PXB, PXF	0.2	0.02	4.0 σ
3	pixel pairs	PXB, PXF	0.6	0.015	0.09
4	detached triplets	PXB, PXF	0.3	1.5	15
5	mixed triplets	PXB, PXF, TIB (1,2), TEC (1,2)	0.4-0.6	1.5	10
6	pixel-less pairs	TIB (1,2), TID/TEC (1,2)	0.7	2.0	10
7	TOB/TEC pairs	TOB (1,2), TEC (5)	0.6	6.0	30

Table 2.1.: Configuration of the CMS seeding in the iterative tracking steps [162, 170]. Either pairs or triplets are used, as indicated by the second column. For sub-detectors other than pixel, the used layers are indicated in parentheses. Minimum p_T, maximum transverse, d_0, and longitudinal, z_0, impact parameter are given by the following columns. The Gaussian longitudinal width of the beamspot is denoted by σ.

Track Selection applies a set of quality measures to reduce the number of *fake tracks*, i. e. tracks not associated with a charged particle (ibid.). These criteria include the number of associated hits and traversed layers, number of consecutive invalid hits, normalized χ^2 value and compatibility with the reconstructed primary vertices. Tracks excelling in some or all of the above criteria are tagged as high purity tracks, which are the subject of most physics studies.

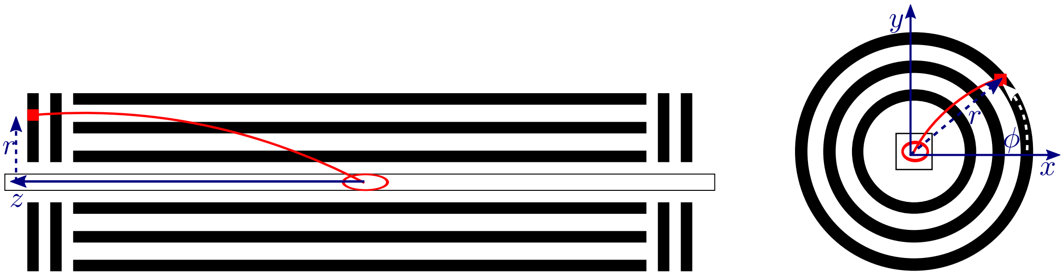
2.3.4. Triplet Seeding

The seeding procedure, requiring about 16 % of the overall reconstruction runtime [79], provides the first estimate of a track's parameters and is the basis for the subsequent Kalman filter-based track finding [162]. To fit the track parameters $\boldsymbol{\tau}$, at least three hits are required or two hits and a beamspot constraint. To avoid the full combinatorics of all hits in the seeding layers, seeds must exhibit a minimum transverse momentum, maximum transverse impact parameter and be compatible with the beamspot position, i. e. the luminous region of the proton-proton collision. These parameters vary with the iterations of the iterative tracking and are presented in Table 2.1. The first, second and forth iteration require the track to produce hits in each of the three pixel layers. As Sguazzoni et al. [162] state, 85 % of the charged particles produced within the geometrical acceptance of the tracker, $|\eta| < 2.5$, fulfill this requirement. To account for pixel detector inefficiencies, such as gaps in the coverage, non-functioning modules and readout saturation, the third iteration is seeded by pairs of hits. Later iterations are targeted towards particles not produced directly in the proton-proton collision and include double-sided layers of the silicon strip detector in the seeding.

Triplet finding can be considered as pattern recognition problem in a three-dimensional volume.[3] Exploiting the layered construction of the CMS detector, the problem can be reduced to two dimensions, by treating barrel and endcap layers separately. As Figure 2.5 illustrates, the detector geometry defines the radius r or z position of a barrel or endcap

---

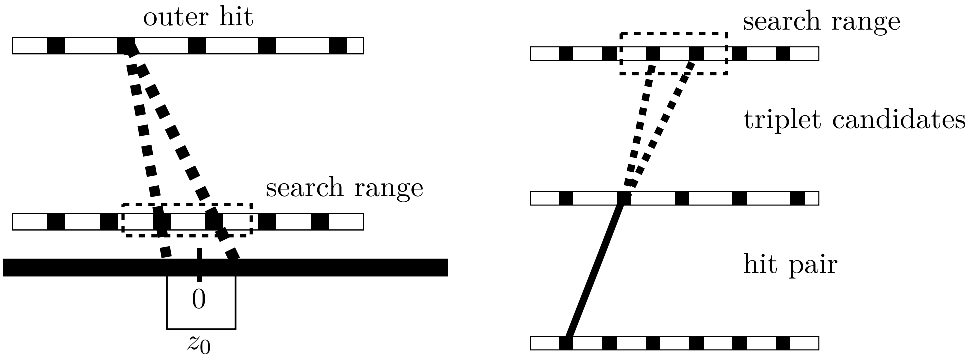
[3] The same applies to pair finding.



(a) In the endcap, the detector geometry defines z_{layer}, therefore a hit is completely defined by (ϕ, r).

(b) Detector geometry defines r_{layer}, hit described by (ϕ, z).

Figure 2.5.: Distinction of seeding in the barrel and the endcap region. Using information given by the detector geometry, the three-dimensional problem can be reduced to two dimensions.



(a) For pair finding, the outer hit is used in conjunction with the track parameter constraints to restrict the search range for the inner hit.

(b) In triplet finding, the hit pair is extrapolated following the track parameter constraints to limit the search range for the outer hit.

Figure 2.6.: Search range restriction for pair and triplet finding in CMSSW.

layer, respectively. This quantity is known with high precision. Given that,

$$x = r \cdot \sin \phi \quad \text{and} \quad y = r \cdot \cos \phi,$$

a hit in a barrel layer is fully defined by its ϕ and z coordinate. Accordingly, endcap hits are completely described by ϕ and r.

If no restrictions were applied, triplet finding would be in $\mathcal{O}(n^3)$. Considering the high pile-up event pictured in Figure 1.7b with 3 627 reconstructed tracks, this would result in approximately 5×10^{10} triplet candidates. Given a minimum transverse momentum p_T and maximal transverse and longitudinal displacement d_0 and z_0, respectively, the number of candidates can be reduced drastically. In the considered event, about 1×10^7 triplet candidates are considered as seeds by the CMSSW seeding step.

Algorithm 2.1 CMSSW pair finding algorithm for the barrel region.

```
Require: seeding layer pair, min $p_T$, max $d_0$ and $z_0$
for all outer hits $\in$ outer layer do
    $\phi$-range $\leftarrow$ $\mathit{compatible}_\phi(\texttt{EventSetup}, \text{inner layer}, p_T, d_0, z_0, \text{outer hit}_{\{\phi, p_T\}})$
    for all inner hits $\in$ { inner layer $\cap$ $\phi$-range } do
        $z$-range $\leftarrow$ $\mathit{compatible}_z(\texttt{EventSetup}, \text{inner layer}, p_T, d_0, z_0, \text{inner hit}_r)$
        if $z$-range $\cap \{\text{inner hit}_z \pm \text{error}\} \neq \varnothing$ then
            valid pair
        end if
    end for
end for
```

Pair Finding is used as sole seeding procedure in iterations three, six and seven and as sub-operation in the other steps. Algorithm 2.1 describes the employed procedure for the barrel region. Given a pair of seeding layers, *all* hits of the outer layer are inspected by the algorithm. As depicted in Figure 2.6a, the range of feasible ϕ-values is computed based upon the outer hit's ϕ and p_T, taking into account the track's bending due to the magnetic field and the effects of multiple scattering. The latter is approximated using the multiple scattering parametrization of Highland [83]. Hits of a detector layer are stored in a ϕ-sorted manner, thus all inner hits inside the predicted ϕ-range can be retrieved quickly. For a given outer hit - inner hit pairing, the feasible z-range is computed, considering the track parameter constraints and the radius r of the inner hit. The radius is defined by the position of the hit's detector element in the barrel and therefore known with little uncertainty. If the inner hit is inside the predicted z-range, the pair is regarded as valid seed. In the endcap, instead of predicting the z-range based on r, the r-range is determined using the detector geometry-defined z coordinate.

Triplet Finding generates the seeds for most reconstructed tracks [162]. The procedure is given by Algorithm 2.2 and visualized in Figure 2.6b. Initially, hit pairs are constructed according to the algorithm outlined in the preceding paragraph. Multiple layers for the third hit of a triplet can be specified, refer to Table 2.1. Thus, for each generated hit pair, all defined outer layers are inspected for third hit candidates. Again considering the barrel, the feasible z-range of the third hit is extrapolated on a straight line based upon the hit pair and the detector geometry of the outer layer. The prediction is then corrected for effects due to track bending and multiple scattering, using the same parametrization as in pair finding [83]. Given the z-range in conjunction with the imposed track parameters, the feasible detector region can be constrained. To determine the ϕ-range within that region, an approximation of the helical path of a charged particle is employed. In the transverse plane, the particle's helix can be described by a circle

$$(x-a)^2 + (y-b)^2 = R^2,$$

with circle center (a, b) and radius R. A mapping

$$f : U \to V \qquad \text{with } U, V \subset \mathbb{R}^n$$

Algorithm 2.2 CMSSW triplet finding algorithm for the barrel region.

```
Require: seeding layer triplet, min $p_T$, max $d_0$ and $z_0$
pairs ← hit pairs in inner two seeding layers
for all pairs do
    for all outer layers do
        $z$-range ← $predictLine_z$(outer layer, pair)
        $z$-range ← $correctMS_z$(EventSetup, $p_T$, outer layer, pair, $z$-range)
        $\phi$-range ← $predictHelix_\phi$(outer layer, pair, $p_T$, $d_0$, $z_0$, $z$-range)
        $\phi$-range ← $correctMS_\phi$(EventSetup, $p_T$, outer layer, pair, $\phi$-range)
        for all outer hits ∈ { outer layer ∩ $\phi$-range } do
            $z$-range ← $predictLine_z$(outer layer, pair, outer hit$_r$)
            $z$-range ← $correctMS_z$(EventSetup, $p_T$, outer layer, pair, $z$-range)
            $\phi$-range ← $predictHelix_\phi$(outer layer, pair, $p_T$, $d_0$, $z_0$, outer hit$_r$)
            $\phi$-range ← $correctMS_\phi$(EventSetup, $p_T$, outer layer, pair, $\phi$-range)
            if $z$-range ∩ {outer hit$_z$ ± error} ≠ ∅
            ∧ $\phi$-range ∩ {outer hit$_\phi$ ± error} ≠ ∅ then
                found triplet
            end if
        end for
    end for
end for
```

is conformal at a point u_0, if it preserves the angles between curves passing through u_0. Transforming (x, y) coordinates to the (u, v) plane according to

$$u = \frac{x}{x^2 + y^2}$$
$$v = \frac{y}{x^2 + y^2},$$

translates circles in (x, y) into straight lines in (u, v) [61]. Imposing

$$R^2 = a^2 + b^2, \tag{2.1}$$

yields the straight line equation

$$v = \frac{1}{2b} - u\frac{a}{b}. \tag{2.2}$$

By fitting the transformed hit pair and the borders of the feasible detector region to this line, the center and curvature of the track's path can be determined. However, since Equation (2.1) forces the circle to pass through the origin, the transverse impact parameter d_0 is lost. Hansroul et al. [77] mitigate this shortcoming by introducing a δ to Equation (2.1)

$$R^2 = a^2 + b^2 + \delta.$$

For $\delta \ll R^2$, the circle in (x, y) can be approximated by a parabola with very small curvature

$$v = \frac{1}{2b} - u\frac{a}{b} - u^2 d_0 \left(\frac{R}{b}\right)^3,$$

with $d_0 = R - \sqrt{a^2 + b^2} \approx \frac{\delta}{2R}$, thus preserving the transverse impact parameter (ibid.). The obtained ϕ-range is then, after being corrected for bending and multiple scattering effects, used to retrieve possible third hits for the pair from the outer layer. For each third hit candidate, the feasible z- and ϕ-range is re-evaluated using the outer hit's radius r. If the outer hit is within these extrapolated ranges, the triplet is accepted as valid seed. The treatment of the endcap region differs only in the interchanged roles of z and r.

2.4. Event Generation

Probing a physical theory requires the verification or falsification of theoretical predictions about natural processes. By simulating the interaction of particles according to the theory under scrutiny, the response of the CMS detector can be anticipated and compared to the measurements of real particle collisions. These interactions can only be modeled in a non-deterministic manner, therefore Monte Carlo (MC) methods are required for their simulation [50, 130]. Several established tools exist for collision simulation and are used by the CMS collaboration, e. g. PYTHIA [168], MadGraph [9] and Herwig++ [16]. PYTHIA is one of CMS's main MC generators, as it is specifically designed to simulate the hard proton-proton interaction. More than 300 Standard Model, minimal supersymmetric and non-standard physical processes are implemented in the framework.

The interaction of the particles produced by the MC generator with the detector is simulated by the Geant4 software package [7]. A detailed detector description, including geometrical layout and material composition, has been modeled in Geant4 to accurately simulate particles' energy depositions in the detector elements and the influence of the material and magnetic field on their trajectories. The simulation yields digitized detector readouts, equal to the `Digis` of the Data Acquisition system. Hence, the identical reconstruction software can be employed for the simulated collisions.

Therefore, simulated events are of crucial importance for algorithmic studies as well. Comparing the output of a reconstruction algorithm to the simulated particles, referred to as *MC truth*, gives insights into the physical performance of the algorithm. The ability to reconstruct the tracks of simulated charged particles is measured by the *efficiency*

$$\text{Eff} = \frac{n_\text{valid reconstructed}}{n_\text{simulated}}, \tag{2.3}$$

for a simulated event with $n_\text{simulated}$ particle tracks, out of which $n_\text{valid reconstructed}$ were successfully reconstructed. The number of reconstructed tracks not associated to a simulated particle – *fake tracks* – defines the *fake rate*

$$\text{FR} = \frac{n_\text{fake tracks}}{n_\text{total reconstructed}}. \tag{2.4}$$

The trajectory of one simulated particle might result in several reconstructed tracks – denoted *clones*. The number of produced clones yields the *clone rate*

$$\mathrm{CR} = \frac{n_{\text{clones}}}{n_{\text{total reconstructed}}}. \tag{2.5}$$

Furthermore, MC generators can be configured to produce a specific number of tracks within one event, enabling the study of the runtime behavior of the algorithm.

Chapter 3

Parallel Computing

For many years software designers could rely on ever increasing clock speeds of CPUs to speed up their single-threaded applications with minimal effort on their part. Driven by Moore's Law [126] of an exponentially growing number of transistors on integrated circuits in conjunction with Pollack's Rule [29] of computing performance increasing roughly proportional to the square root of the increase in complexity, developers could rest assured that their software would automatically meet increasing computational demands[1]. However, clock speeds have been approaching the limits of the physically possible and large performance gains can no longer be expected from increasing operation frequency. Instead, new technologies have been introduced to modern CPU architectures, such as multiple cores and vector units. These technologies need to be *actively* exploited by software developers, dawning an end to the „free lunch“ era [175].

Additionally, the computational power of Graphical Processing Units (GPUs) has attracted much research attention in the past decade [139]. Outperforming CPUs in peak performance by an order of magnitude – refer to Figure 3.1 – they have been successfully utilized for compute-intense applications such as DNA sequencing [33], simulation of molecular dynamics [11] or climate modeling [46]. In particular, GPUs have been used to execute the Game of Life Cellular Automaton by Harris et al. [78], who also coined the term General Purpose Graphical Processing Unit (GPGPU) for non-graphical computations performed on a GPU. While GPU programming using low level shader programming languages is an arduous, error-prone task, the introduction of high level Software Development Kits (SDKs) such as NVIDIA's CUDA in 2007 [137] and Open Computing Language (OpenCL) in 2008 [12] greatly eased the process of using GPUs for general purpose computations and helped GPGPUs attain the popularity they enjoy today. The latter is maintained by the non-profit technology consortium Khronos Group [101] and allows the development of programs executing across heterogeneous platforms such as CPUs as well as GPUs of different vendors. As platform independence is crucial for applications developed for the Worldwide LHC Computing Grid, OpenCL was chosen as basis for this work and is described in more detail in Section 3.3. The prior Sections 3.1 and 3.2 give an overview of current CPU and GPU technologies, respectively. Section 3.4 introduces metrics used to evaluate the quality of parallel algorithms.

---

[1] e. g. increased data volume, growing user base, etc.

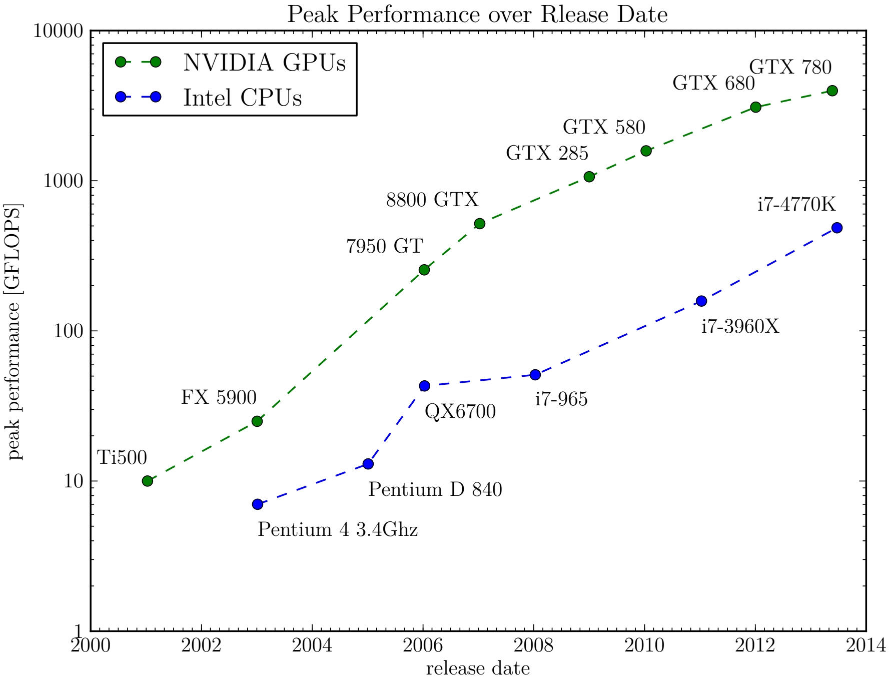


Figure 3.1.: Comparison of peak performance of CPUs and GPUs.

3.1. CPU Technologies

Multi-core processors have been the predominant trend in CPU technology [66]. In 2001, IBM released the POWER4 [87], the first non-embedded microprocessor to integrate two independent processing units, *cores*, on a single die. With the introduction of the first dual-core consumer CPU, AMD's Athlon 64 X2, in 2005 [3] multi-core processors also became the standard in consumer hardware.[2] Today, up to 12 cores can be found within one processor. To exploit this trend, existing applications need to be redesigned in a multi-threaded manner.

Fortunately, reconstructing particle tracks from independent collisions is embarrassingly parallel on an event level [125]. However, CMSSW was *not* designed with parallelism in mind, thus processing multiple events concurrently within one instance of the application is currently not possible. This limitation is circumvented by running several CMSSW instances in parallel on the same machine, each processing its own set of events. With a memory footprint of over $\approx 1.5\,\text{GByte/instance}$ for events with ten pile-up collisions, this approach quickly reaches the limits of memory capacity and bandwidth. Therefore, reducing the memory footprint would allow to process more CMSSW instances concurrently. Running isolated instances of CMSSW in parallel to concurrently process events suffers from the lack of sharing of common data between the processes. A multi-threaded framework, using shared detector geometry and magnetic field data, allows for a higher level of concurrency due to a lower memory footprint and is under active research e. g. by Hegner et al. [82]. Still, with many-cores on the horizon, the *one core per event* approach does not scale with memory requirements, thus intra-event-parallelism is required to fully exploit the future hardware's capabilities. The CMS collaboration pursues two approaches

---

[2] Intel's Pentium D, although released earlier, was not an actual dual-core processor but rather a traditional Symmetric Multiprocessing (SMP) system within one package.

1. **Module-level parallelism** As described in Section 2.3, the event's processing is already decomposed into a multitude of modules. Identifying dependencies between those modules allows for concurrent execution of independent ones. This concept is scrutinized by Jones [94] as well as Hegner et al. [82].

2. **Algorithm-level parallelism** Using parallel algorithms instead of sequential ones within a module is the lowest level of exploitable parallelism. For instance, several hits, tracks or detector layers could be processed concurrently by multiple cores, in the CPU's vector units or on a GPGPU. This level of parallelism is the focus of Hauth et al. [79] and this thesis.

According to Flynn [54], parallelization using multiple concurrent threads is characterized as Multiple Instruction Multiple Data (MIMD) as threads may execute different operations on different data. In another processing scheme – Single Instruction Multiple Data (SIMD) (ibid.) – the same instruction is performed on multiple data points, e. g. addition of two vectors, hence the term vector processing. Specialized vector supercomputers of the 1970s and 1980s processed up to 64 000-dimensional vectors [58, 112] but were superseded by commodity hardware-based supercomputers. Vector processing regained interest with the introduction of SIMD extensions to the Instruction Sets (ISs) to commodity processors. Intel's first SIMD extension, „MMX Technology“ introduced in 1996 [92], was quickly superseded and expanded by the Streaming SIMD Extensions (SSE) in 1999 [88] and finally, after four iterations of SSE, by Advanced Vector Extensions (AVX) in 2011 [90]. Each IS features dedicated SIMD registers[3] and a growing number of instructions to operate on packed integer or floating point values. AVX comprises fifteen 256 Bit registers to hold either 4/8 double/single-precision floating point values, respectively, or 4/8/16/32 **`long`/`int`/`short`/`char`** numbers. The instructions cover the movement of data between main memory and vector registers, with the possibility to specify the stride of successive elements in main memory, value and bitwise comparison as well as basic arithmetic operations, such as add, multiply, divide and square root, which can operate in a predicated manner – refer to [89] for the complete AVX specification.

Exploiting SIMD instructions in algorithm implementations can be a tedious task, involving low level, IS-specific optimizations (intrinsics). Only recently, compilers started offering auto-vectorization features [133]. Through dependency analysis of the innermost loop of an algorithm, the compiler may be able to package independent loop iterations for vector processing. In some circumstances, vectorization is even possible in the presence of control flow instructions in the loop [165]. Still, the process is rather fickle and loop bodies need to be sufficiently simple to be analyzable by the compiler and refrain from calling non-intrinsic functions. Hauth et al. [79] explore this algorithm-level parallelization opportunity by developing a vectorizable math library and applying it to the vertex clustering in CMSSW. Gorbunov et al. [69] present a Kalman filter-based track fitter tailored towards SIMD processing.

---

[3] Except for MMX, which reuses the Floating Point Unit (FPU) registers.

OpenCL allows the software developer to exploit both, multiple cores and SIMD instructions, in an accessible and portable manner, refer to Section 3.3 for further details.

3.2. General-Purpose-GPUs

Graphics hardware is designed to *render* three-dimensional primitives into a two-dimensional image, viewable on a computer monitor [70]. The rendering process encompasses several distinct steps to transform the vertices – points defining corners and intersections of geometric shapes – describing the three-dimensional primitives into colored pixels for display (ibid.):

Vertex Shading Vertices are positioned in the scene and transformed according to their appearance in the two-dimensional image, influencing the shape and lightening of the described geometrical objects.

Rasterization In this step, the vertex-based scene description is transformed into pixel-based fragments. Polygons in the scene are represented by a set of triangles which are determined by three vertices. The rasterizer transforms those vertices into two-dimensional points and fills the resulting two-dimensional triangle appropriately,.

Pixel Shading Textures can be assigned to objects of the scene to display different materials. These textures are stored as flat bitmap images (texture maps), which need to be translated into the correct perspective in three-dimensional space and applied to the pixels of the object in the fragment. This step yields the final image for output.

The hardware components of a GPU closely resemble this processing pipeline and are depicted in Figure 3.2. Historically, the vertex and pixel shader as well as the rasterizer

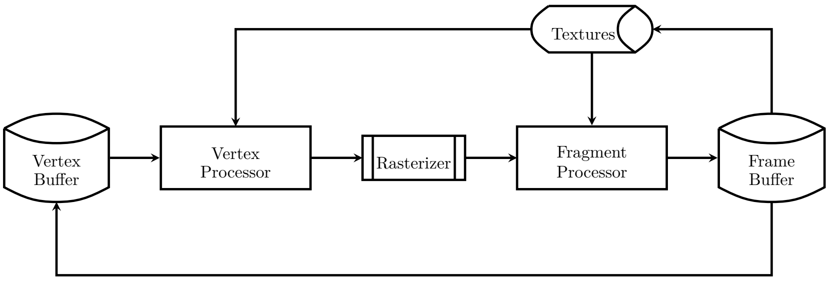
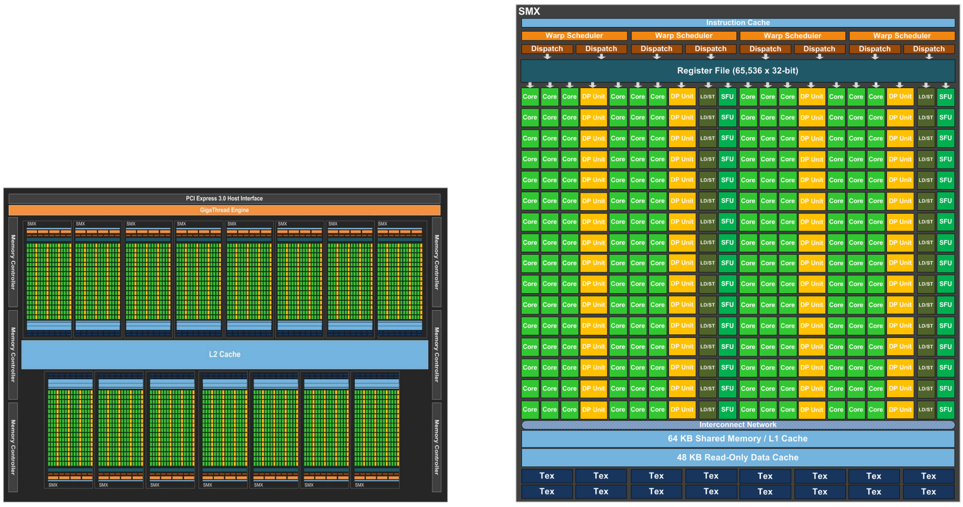


Figure 3.2.: Schematic of the GPU pipeline. In modern GPUs, the vertex and fragment processors are freely programmable and realized as unified shaders.

were fixed-function components [139]. With increasing demands on graphics quality, the pipeline components needed to become more flexible to accommodate more realistic lighting and texturing. Therefore, the vertex and fragment processors transitioned from fixed-function units to programmable components (ibid.). Graphics Application Programming Interfaces (APIs), such as Khronos Group's OpenGL or Microsoft's DirectX [70], supported this development with the introduction of shader programming languages. In 2005, this development culminated with the introduction of Unified Shader Models (USMs) [127], providing one large grid of general data-parallel floating-point processors instead of separate custom processors for vertex and pixel shading [118]. The USM allows for better utilization of the GPU's capabilities, as more resources can be allocated to the compute-intense shader programs, regardless whether they are concerned with vertices or pixels. Furthermore, this paved the way for general-purpose computing on GPUs [139].

Scientists have been using GPUs to accelerate computations as early as 1994 [32], but efforts were rather isolated. Momentum picked up with the development of OpenVIDIA by Fung and Mann [63], a GPU-accelerated framework for computer vision and image analysis, in collaboration with NVIDIA, which later released the CUDA SDK [137]. GPGPU computing exploits the stream processing of the GPU's hardware. Vertices and pixels can be treated individually, therefore GPUs are optimized for high-throughput of massively-parallel data points [118]. A function, called kernel, is applied to each data point independently, which should be arithmetically intense in order to mitigate the memory access latencies. Recent GPU architectures such as NVIDIA's Kepler feature up to 1 536 cores, organized in 15 Streaming Multiprocessors (SMXs), on a single graphics card, as illustrated by Figure 3.3. The GeForce GTX 680 top model, reaches a memory bandwidth of $192\,\mathrm{GByte\,s^{-1}}$ and 3 090 GFLOPS [138], a Intel Core i7 3960x features $51\,\mathrm{GByte\,s^{-1}}$ memory bandwidth and 140 GFLOPS [91]. Care should be taken when comparing those numbers as CPUs are designed for low-latency, control flow intense computations. The principles of GPU programming are closely coupled to the design of respective APIs, thus they will be detailed in the subsequent Section 3.3.

GPUs are exploited to accelerate particle track reconstruction in the HLT of the ALICE experiment [6]. NVIDIA's CUDA SDK is used to implement the Cellular Automaton (CA)-based tracking algorithm described in Chapter 4. Further research on GPUs in the physics community include the work by Lamanna et al. [114], who employ GPUs in the low level trigger at the NA62 experiment, and Perez-Ponce et al. [141] as well as Seiskari et al. [157], who explore GPUs to accelerate the Geant4 simulation toolkit.


PCI Express 3.0 Host Interface
GigaThread Engine
Memory Controller
SMX
L2 Cache
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex

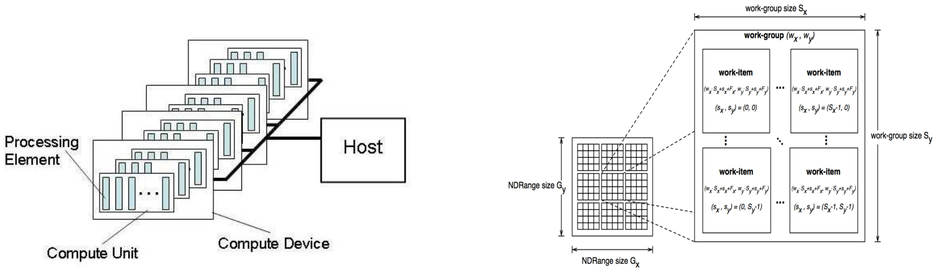
(a) Block diagram of NVIDIA GK110 chip with 15 multiprocessors [138].

(b) Block diagram of NVIDIA SMX. It includes 192 single-precision computing cores, augmented by 64 double-precision units, 32 special function units and 32 load/store units [138].

Figure 3.3.: Schematic of NVIDIA's Kepler architecture.

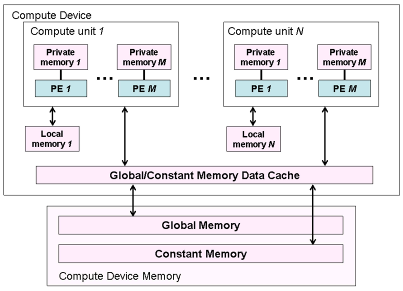
3.3. OpenCL

OpenCL is an open framework maintained by the Khronos Group for developing massively-parallel applications, executing across heterogeneous *compute devices* such as CPUs and GPUs [128]. The framework encompasses a C-derived programming language to specify functions, so-called *kernels*, for execution on the compute device and an API for the *host* application to steer memory movements between host and device as well as schedule kernels for execution. Its platform independence makes OpenCL an ideal candidate for developing parallel algorithms destined for execution in the WLCG. However as a cross-vendor, cross-platform framework it can not leverage the latest hardware developments. Therefore, it stands to reason that OpenCL could be outperformed by platform or vendor specific solutions. For multi-core CPU compute devices Shen et al. [164] show that initial performance gaps between OpenCL and OpenMP, a well established API for shared-memory parallel programming [47], can be mitigated by forgoing GPU-specific optimizations. After tuning their OpenCL implementation towards CPUs, Shen et al. [164] even report OpenCL outperforming OpenMP in 80 % of their test cases. Even though NVIDIA's CUDA is the most advanced framework for GPU computing, offering features such as C++ templates in kernels and dynamic parallelism [134], Fang et al. [52] attest OpenCL a similar performance for a wide range of problems and CUDA outperforming OpenCL by at most 30 % on a few. They attribute performance differences partly to the immaturity of the OpenCL compilers, a point that is also raised by Shen et al. [164]. Improvements by the platform vendors on the compiler quality could thus


Processing Element
Compute Unit
Compute Device
Host
work-group size S_x
work-group (w_x, w_y)
work-item
$(w_x \cdot S_x+s_x+F_x, w_y \cdot S_y+s_y+F_y)$
$(s_x, s_y) = (0, 0)$
$(s_x, s_y) = (S_x-1, 0)$
$(s_x, s_y) = (0, S_y-1)$
$(s_x, s_y) = (S_x-1, S_y-1)$
work-group size S_y
NDRange size G_y
NDRange size G_x

(a) Platform model illustrating relationship between host and its compute device(s) [128].

(b) Execution model of work-items, organized in more coarse-grain work-groups, mapped onto an index space [128].


Compute Device
Compute unit 1
Compute unit N
Private memory 1
Private memory M
PE 1
PE M
Local memory 1
Local memory N
Global/Constant Memory Data Cache
Global Memory
Constant Memory
Compute Device Memory

(c) Memory model of a compute device exhibiting the memory hierarchy of OpenCL [128].

Figure 3.4.: Schematic of the OpenCL platform, execution and memory model.

further alleviate existing performance differences.

In order to provide platform independence, OpenCL needs to present an abstracted model of the underlying hardware to the developer. The key elements of this model are described in the following [128].

Platform Model Figure 3.4a illustrates the platform model of OpenCL. One host is connected to one or more compute devices which are further partitioned into compute units each containing one or several processing elements. Processing elements of one compute unit execute a kernel either as SIMD unit, i. e. in lockstep, or as Single Program Multiple Data (SPMD) unit, i. e. each processing element maintains its own program counter. Table 3.1 summarizes the mapping of OpenCL entities to CPU and GPU components. The host interacts with the compute device via commands submitted to a command queue.

Execution Model Kernels are executed within an OpenCL context. The context comprises a set of compute devices, memory objects – handles to data in the device's

OpenCL	**CPU**	**GPU**
Compute Device	CPU	GPU
Compute Unit	thread	multiprocessor
Processing Element	processor core	computing core
Global Memory	DRAM	DRAM
Constant Memory	DRAM	DRAM
Local Memory	DRAM	shared memory
Private Memory	registers	registers
Caches	L1/L2 per core L3 per CPU	L1 per SMX L2 per GPU

Table 3.1.: Mapping of OpenCL entities to CPU and GPU components [150].

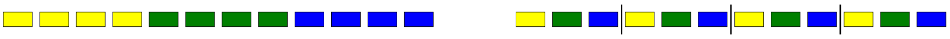
memory – and a command queue. The host can enqueue memory transfers, kernel executions or synchronization points into the command queue, which are then executed by the device either in- or out-of-order. Kernels are executed in a one, two or three dimensional *index space*, refer to Figure 3.4b. Each point of the index space is handled by a *work-item*, an instance of the kernel with its index position (*global ID*) as implicit input. Via the global ID, the symmetry between the work-items is broken and the appropriate memory locations for processing identified. The work-items are organized in *work-groups*. All work-items of one work-group are executed on the processing elements of a single compute unit and are scheduled in batches referred to as *warps*. Work-items of one work-group are able to synchronize at work-group barriers. There are no means to synchronize work-items of different work-groups.

Memory Model OpenCL employs a hierarchical memory model, depicted in Figure 3.4a. Table 3.1 relates the abstract entities to the hardware components of CPUs and GPUs. The state of the memory visible to a work-item might not be consistent across a collection of work-items at all times since a relaxed consistency model is used.

Global Memory is accessible by all work-items of all work-groups and offers read and write access. Depending on device capabilities, accesses might be cached for latency reduction. The host can transfer data to and from global memory. No consistency guarantee is made for work-items of different work-groups accessing global memory. Within a work-group, memory consistency is ensured at work-group barriers placed by the programmer.

Constant Memory is allocated and initialized by the host and remains unchanged during kernel execution. All work-items of all work-groups are able to read from this memory area.

Local Memory can only be accessed by work-items of one work-group. Its consistency is ensured at a work-group barrier for all work-items of that work-group. The host can allocate local memory for work-groups but has no read/write access to it.



(a) The work-items process consecutive data points. Therefore, the concurrently running work-items of a work-group access *non*-consecutive memory segments, resulting in *non*-coalesced memory accesses.

(b) A work-item processes memory segments separated by the work-group size. Therefore, work-items of the work-group concurrently access consecutive segments in a coalesced memory access.

Figure 3.5.: Mapping of index space to memory locations. Boxes represent memory locations, work-items of one work-group are represented by different colors.

Private Memory is exclusively associated to one work-item for local variables of the kernel.

Programming Model Data parallel execution is the primary model of OpenCL. The same sequence of operations is applied to multiple data points. The index space maps work-items to the data to be processed. Parallelism is achieved by processing huge amounts of data. OpenCL is also capable of task parallel processing, i. e. executing a single kernel instance. A multitude of enqueued tasks is the source of parallelism in this model.

Since OpenCL employs a C-derived language, kernel source code should look familiar to most developers. The OpenCL specification augments traditional C by functions to query a work-item's position in the work-group local or global index space, atomic operations on local and global memory as well as synchronization commands. Furthermore, vector versions of various data types are introduced to the language, for instance two-, three-, or four-dimensional floats – `float2`, `float3` and `float4`, respectively. Notwithstanding the familiarity of the OpenCL programming language to most developers, the design of kernels requires the consideration of general GPGPU and specific OpenCL peculiarities [106].

- In general, data transfers between host and device should be minimized [106]. This includes executing computations on the device which would perform equally well on the host in order to avoid data transfers between the two. Overlapping kernel execution and memory transfer of independent data is also an apt approach.

- As of OpenCL 1.2 [128], kernels are not able to allocate memory (global or local) dynamically. Thus, all memory needs to be allocated by the host prior to kernel submission. This aggravates kernel development, where the memory demand depends on the outcome of the kernel computation itself.

- Accessing global memory suffers from high latency, thus local memory should be preferred for read/write operations [136]. Data transfer between both memory areas needs to be explicitly handled by the developer, albeit more recent GPUs feature caches for global memory on each SMX. Accessing consecutive segments of memory results in the highest possible throughput. Therefore, the mapping from index space to data points should be devised accordingly as illustrated by Figure 3.5.

- Control flow instructions, such as branches, are an essential building block of most algorithms. On CPUs, branches incur little overhead if predicted correctly by the instruction unit, otherwise a pipeline flush is required. Threads possess their own program counter, therefore thread divergence cause no penalty other than cache inefficiencies. On GPUs, the threads of one SMX are processed in lockstep, therefore threads, taking different branches of a control flow instruction, must be scheduled sequentially. Even though NVIDIA introduced MIMD processing in its vertex processors in 2005 [103], it is still advised to avoid divergent execution paths within one warp [136]. Techniques to avoid branches include moving flow-control decisions up the pipeline, i. e. treating the interior and the border of a grid in two individual kernels, and predication, computing both sides of a branch and discarding one of the results. Branch avoidance and dynamic warp formation based on taken branches is still a field of active research [64, 76].

The implications of the raised peculiarities of OpenCL become more evident in Chapter 9, when discussing the algorithms designed and implemented in this thesis.

3.4. Performance Metrics

An essential step of developing algorithms for modern hardware platforms is the thorough evaluation of the merit of certain optimizations and technology usages. As real-world hardware more and more deviates from analytically tractable machine models, experiments are of increasing importance in the analysis of the performance of new algorithms. Furthermore, insights gained from experiments can shape the further development of the algorithm, as promoted by the *algorithm engineering* discipline [151].

In order to derive insights from experiments, meaningful metrics need to be employed [15]. For serial algorithms, the CPU time spent on varying input sizes provides an understanding of the efficiency and scalability of the implementation. The runtime of parallel algorithms not only depends on the input but also on the machine size – i. e. the number of used processors – and is influenced by synchronization and message passing between them. Therefore, elapsed wall time is a more meaningful measure than CPU time (ibid.).

To measure the success of an algorithm's parallelization, the *speedup* relates the execution times of the implementation for varying machine sizes p and is defined as

$$S(p) = \frac{T(1)}{T(p)}. \tag{3.1}$$

Two kinds of speedup can be distinguished: the *absolute* speedup relates the processing time $T(p)$ of the parallel algorithm on p processors to the runtime $T(1)$ of the best known sequential algorithm, thus quantifying the merit of using a parallel algorithm, taking into account potential costs due to the parallelization; relating $T(p)$ to the runtime of the parallel algorithm with just one processor – thus $T(1)$ includes the parallelization overhead – yields the *relative* speedup, measuring the scalability of the algorithm with machine size. Ideally, $S(p)$ should be linear in p with $S(p) = p$. The utilization of processors by the algorithm is measured by the *efficiency*

$$E(p) = \frac{S(p)}{p}. \tag{3.2}$$

A scalable algorithm should have an efficiency close to one in order to effectively utilize a large number of processors.

The attainable speedup is governed by Amdahl's law [10]

$$S(p) = \frac{1}{s + \frac{1-s}{p}},$$

with s denoting the proportion of the application which is inherently sequential and does not benefit from parallelization. Even small values of s drastically limit the reachable speedup,

$$\lim_{p\to\infty} S(p) = \frac{1}{s}.$$

For instance, given a 90 % *parallelizable* application, the maximum possible speedup is 10. Hence, careful algorithm design is required to reduce s to the smallest possible value. Karp and Flatt [99] introduce a method to determine s experimentally either instead or in addition to deriving it analytically:

$$s = \frac{\frac{1}{S(p)} - \frac{1}{p}}{1 - \frac{1}{p}}. \tag{3.3}$$

Whereas Amdahl's law considers the speedup for a fixed input size and therefore concludes the domination of the serial portion when $p \to \infty$, Gustafson's law [73] is concerned with a problem size growing monotonically in p, proposing

$$S(p) = p - s(p - 1).$$

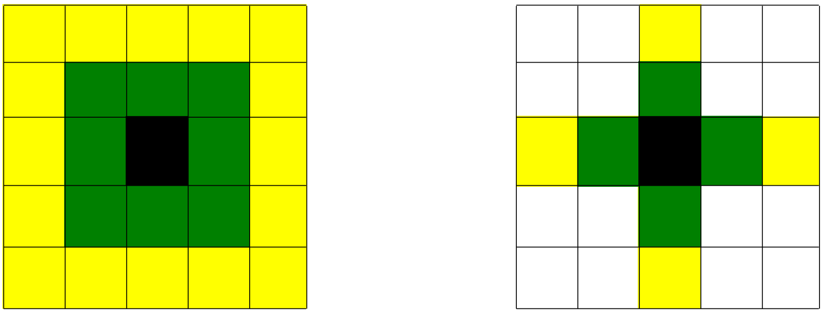
Given a sufficiently large problem and number of processors, the limitations due to inherently sequential computations can be mitigated by increasing the amount of total computations.

Chapter 4

Cellular Automata

Cellular Automata (CA) as a computation model were first studied by Neumann [129] to calculate the motion of liquids. The liquid is partitioned into discrete units and each unit's motion is calculated based upon the behavior of its neighbors in discrete time steps. More formally, consider a regular grid of cells $R = \mathbb{Z}_1 \times \cdots \times \mathbb{Z}_d$ in d dimensions. For each cell $\mathbf{i}$ with $\mathbf{i} = (i_1, \ldots, i_d)$ we define a neighborhood $N = \{\mathbf{n}_1, \ldots, \mathbf{n_k}\} \subset \mathbb{Z}^d$ with $\mathbf{n}_j$ representing coordinate *differences*, thus the jth neighbor of $\mathbf{i}$ is cell $\mathbf{i} + \mathbf{n}_j$. Figure 4.1 illustrates two widely used neighborhoods for CAs, the Moore and von Neumann neighborhood. Every cell is in *one* of a finite number of Q states. For each time step t, the configuration $c^t : R \to Q$ describes the state of cell $\mathbf{i}$, given by $c^t_{\mathbf{i}} \in Q$. A CA is characterized by its transition function $\delta : Q^N \to Q$ determining the new state $c^{t+1}_{\mathbf{i}}$ of a cell $\mathbf{i}$ based on the states of cells in its neighborhood $c^t_{\mathbf{i}+N}$. By definition CA are *homogeneous*, i. e. the same transition function and neighborhood is used for all cells. We consider *synchronous* CAs where all cells simultaneously change their state.

Cellular Automata provide an apt model to design parallel algorithms [120]. Their spatial properties can be well mapped to physical or logical computer and network architectures and their localized communication pattern translates into predictable memory and network accesses. Hence, CA-based parallel algorithms can be applied to a



(a) Moore neighborhood with radius $r = 1$ (green) and $r = 2$ (yellow).

(b) Von Neumann neighborhood with radius $r = 1$ (green) and $r = 2$ (yellow).

Figure 4.1.: Illustration of two widely used neighborhood functions for Cellular Automata.

wealth of problems (ibid.). For physics applications, Glazov et al. [68] were the first to apply a CA to suppress noise hits in the event reconstruction of the ARES experiment. CAs can also be employed for track finding in particle collisions as discussed in the subsequent Section 4.1. Section 4.2 outlines a CA for particle track reconstruction in the CMS detector.

4.1. Cellular Automata for Track Finding

The concept of Cellular Automata is used by several HEP experiments. Abt et al. [1] describe a CA-based track finding algorithm for the HERA-B experiment; Kisel [107] adapts that algorithm to the CBM experiment. At the LHC, CAs are employed by the ALICE collaboration in the High Level Trigger (HLT) [6]. The following deliberations outline a general CA-based track finding algorithm, mainly based on [1, 107]. Subsequently, some peculiarities of ALICE's algorithm are discussed. All cited papers consider a less formal model of computation than presented in the previous section, therefore the applicability of the term „Cellular Automaton“ is scrutinized by evaluating the implementability of the algorithm in the formal CA model.

The cells of a track finding CA are defined by track *segments* – formed by hits in adjacent detector layers. Figure 4.2 illustrates a detector with five layers and the segments defined by the reconstructed particle interactions with the detector material. Each segment (cell) s is defined by its two hits $s = (s^i, s^o)$ where s^i is a hit on the inner layer and s^o is on the outer one. Given the set of segments $\mathbf{S}$, a track can be seen as sequence $t = s_1, \ldots, s_n$ of segments. Following Abt et al. [1], track searching can be seen as an optimization

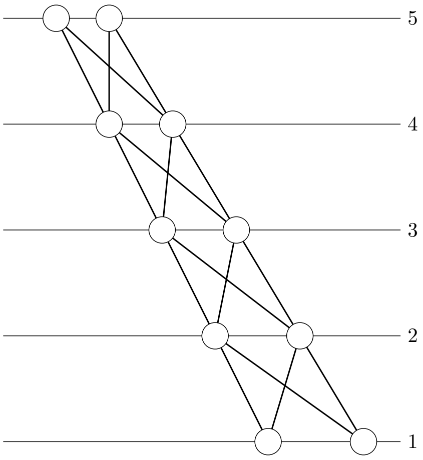


Figure 4.2.: Five detector layers with CA cells defined via segments – two hits in adjacent layers.

problem

$$\underset{t \in \mathcal{P}(\mathbf{S})}{\text{maximize}} \qquad u(t) = n - \gamma \sum_{i=1}^{n-1} \varphi(s_i, s_{i+1}) \tag{4.1}$$

$$\text{subject to} \qquad \forall i \in \{1, \ldots, n-1\} : s_i^o = s_{i+1}^i. \tag{4.2}$$

Equation (4.1) favors long, smooth tracks by introducing a penalty for the breaking angle between two neighboring segments, given by $\varphi(s_i, s_{i+1})$ and weighted by γ (ibid.). Constraint (4.2) ensures a shared hit in the common detector layer of two neighboring segments. Many tracks my be contained in set **S**, hence all *local* maxima to Equation (4.1) are viable track candidates.

With the CA cells being defined by the segments of **S**, a neighborhood function can be obtained from Constraint (4.2)

$$N(s_i) := N^{\downarrow}(s_i) \cup N^{\uparrow}(s_i) = \{s_j \in S : s_j^o = s_i^i\} \cup \{s_j \in S : s_j^i = s_i^o\}.$$

This generic definition needs to be refined to reflect the experimental setup and employed track model of an experiment, for instance by introducing a maximum breaking angle between two neighboring segments or imposing a constraint on the transverse and longitudinal impact parameter when extrapolating a segment to the beam line.

Given the neighborhood definition, the track finding CA proceeds in three steps:

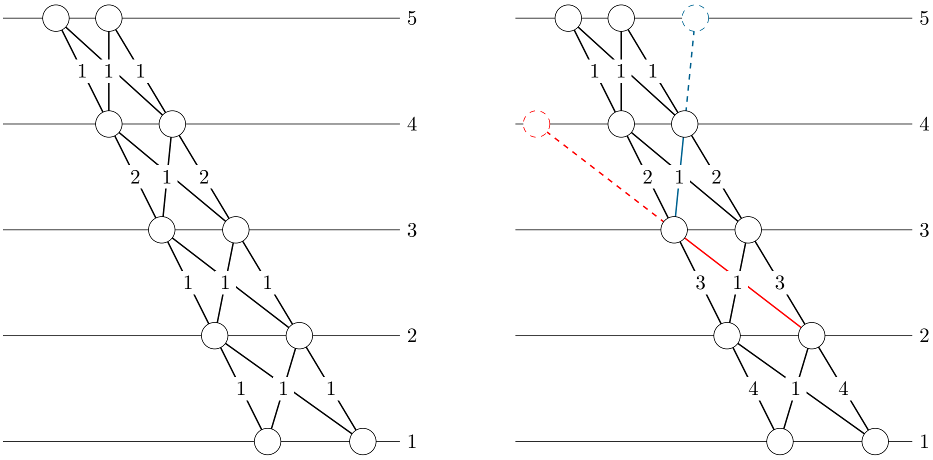
1. **Track formation** joins compatible segments starting at the outermost layer proceeding inwards towards the beam line.
2. **Track selection** determines the longest tracks formed in the first step.
3. **Track collection** gathers the segments belonging to the selected tracks and marks them for further processing.

The following paragraphs elaborate on each step.

Track Formation The state of a cell i represents the length of the track candidate, in track segments, that was successfully formed up to i. Hence, given N detector layers, $Q := \{1, \ldots, N-1\}$. All cells are initialized to unity. The transition function

$$\delta(q) = \max_{\mathbf{n} \in N^{\uparrow}} c_{\mathbf{i+n}}^t + 1$$

realizes a simple counter over the finite set Q. It requires at most $N-1$ time steps to update the states of the cells belonging to the longest track candidate. Figure 4.3 illustrates the configuration of the CA after two and four steps. In the example, two track candidates comprising four segments could be found. The „criss-cross“ segments do not possess neighboring cells in $N^{\uparrow}$ and therefore remain in state 1.



(a) Configuration after two steps.

(b) Configuration after four steps.

Figure 4.3.: CA-based track formation. The cell states represent the number of successfully joined segments from the outermost layer. In Figure (b), the colored dashed circles denote the expected locations of track model compliant hits for the colored criss-cross segments.

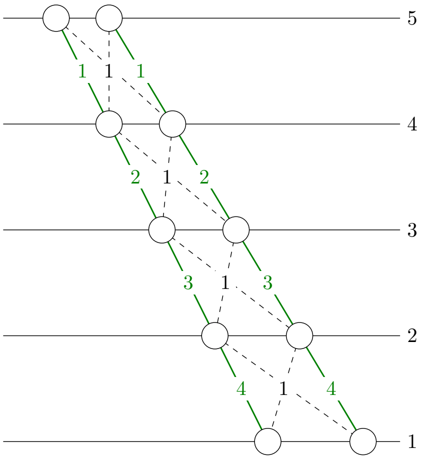


Figure 4.4.: Selection of the longest found tracks by the CA. Starting from the innermost cells with the highest state, collect the cell in the next layer with a state lower by unity until the outermost cell is reached.

Track Selection Subsequent to track candidate forming, the best track candidates need to be identified. In the first phase of the CA, candidates of different lengths have been formed. In order maximize the first part of Equation (4.1), the longest candidates need to be selected. This is trivial when considering the CA merely as an conceptual tool as the longest tracks can be identified from the „outside“ [1, 107].

Within the CA model, maximum finding can be considered a variation of the Queen Bee Problem (QBP) – also known as leader election – introduced by Smith [166]. The QBP asks to elect a unique leader (queen bee) among a connected pattern in a CA. For arbitrary patterns in d dimensions, Stratmann and Worsch [174] describe an algorithm requiring $\Theta(diam \log(diam))$ time, with $diam$ being the diameter of the pattern in number of cells.[1] The adaption to the maximum selection problem is straightforward. The connected pattern is defined by the detector layers and the extent of each layer. A von Neumann neighborhood connects a cell of layer k to its neighbors *within* the detector layer and to the closest – even incompatible in the sense of the track model – cells in layers $k \pm 1$. The borders of the detector are marked by dedicated dead cells, i. e. sentinels. The leader election criterion is the state of a cell. In contrast to the original QBP, the existence of several equally long tracks needs to be taken into account.

Track Collection With the longest track candidate(s) being selected in the preceding phase of the CA, their constituting segments need to be collected and marked for output. This step is again trivial when adapting the notion of a simulated CA.

In the formal CA model, the procedure starts at the end of the track, i. e. the segment with the highest state identified during the previous phase, and proceeds in the opposite direction as the track formation. To mark a segment for output, the state set $Q := \{1, \ldots, N-1\}$ is extended by $\{1^o, \ldots, (N-1)^o\}$. The end segment is immedialty marked for output. The transition function for the further processing is described by

$$\delta(q) = \begin{cases} q^o & \text{if } (q+1)^o \in n^{\downarrow} \\ q & \text{else,} \end{cases}$$

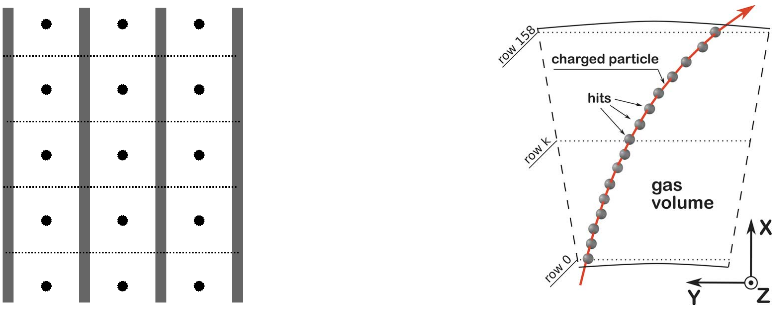
i. e. a cell is marked for output if it observes a cell with a state higher by unity than its own state within its downward neighborhood. Figure 4.4 illustrates this process. All segments between layer four and five are in state 1 and observe a cell with state 2 in their neighborhood. To avoid the criss-cross segments being marked as output track constituents, the transition function can be extended to only mark the segment for output that minimizes the second term of Equation (4.1) – the breaking angle – if several segments in layer k are eligible for output due to the same segment in layer $k-1$ [1]. Capturing this constraint in the formal CA model is rather involved and requires signaling within the layer, thus necessitating an extended neighborhood definition, with subsequent minimum determination.

Phases two and three are repeated, with cells already marked for output not participating in the track selection, thus allowing the second longest track(s) to be identified and marked for output. The process ends when a preconfigured lower threshold for track length is reached.

ALICE's CA To accelerate the track reconstruction in the HLT, ALICE employs a CA-based track finding routine in their TPC detector, implemented on GPGPUs ALICE Collaboration [6]. A TPC is a gas-filled cylindrical chamber with MWPC endplates [123]. A MWPC is a set of thin, parallel, equally spaced and positively charged anode wires between negatively charged cathode planes in a gas volume [155]. A bypassing charged

---

[1] $f(n) \in \Theta(g(n)) \equiv k_1 \cdot g(n) \le f(n) \le k_2 \cdot g(n)$ for some positive k_1, k_2 and sufficiently large n.



(a) View of a MWPC perpendicular to the anode wires. Wires are depicted by black dots, cathode planes by gray thick lines. The dashed lines represent the collection area of each wire mask.

(b) ALICE's TPC endplate configuration with 159 rows of wires [6].

Figure 4.5.: Illustration of a MWPC and its application in the ALICE TPC endplates.

particle causes a localized cascade of ionization in the gas which is collected on the wire due to the external magnetic field, resulting in an electric current proportional to the energy of the detected particle (ibid.). The TPC cylinder is halved along the z-axis by an electric-field induced by a central high-voltage electrode disc. A particle passing through the chamber ionizes the contained gas. The ions drift towards the MWPC endplates due to the electric field, the drift time determines the z-coordinate of the interaction. The wires of the MWPC measure the interaction's x and y coordinate, refer to Figure 4.5b illustrating the TPC of ALICE featuring 159 rows of wires. Due to this abundance of measurement points, the multitude of tracks in Heavy Ion events as well as the homogeneous geometry of a TPC, a simple track model and strong track candidate selection criteria can be employed ALICE Collaboration [6].

ALICE completely eliminates combinatorics in the segment forming. Their cells are based on triplets of hits $\mathbf{t}^k = (h^{k-1}, h^k, h^{k+1})$ in adjacent detector layers $k \pm 1$. For each hit h_i^k in detector layer k only the *best* triplet, i. e. the triplet most closely resembling a straight line , is accepted to form a cell. If two hits h_i^k and h_j^{k+1} are both contained in each other's best triplet they are considered neighbors. Therefore, criss-cross segments as shown in Figure 4.2 are already eliminated at the CA cell level. The further processing of ALICE's algorithms follows the general track finding CA outlined above, with the addition of a Kalman filter-based χ^2 quality check in the track selection and collection.

4.2. CA-based Track Finding in CMS

As the foreseen LHC beam parameters for the next data taking period – starting in 2015 – will double the the number of expected pile-up interactions per event, fast alternatives to Kalman filter-based track finding need to be explored. The CMS collaboration identifies two approaches as viable candidates [161]: Hough transformation- [40] and CA-based track reconstruction techniques.

Hauth et al. [80] examine the suitability of the Cellular Automaton-based approach for CMS. Since CAs are prone for a massively-parallel implementation, Hauth et al. [80] strive to leverage modern CPU and GPU technology in a portable manner as demanded by the heterogeneous nature of the Worldwide LHC Computing Grid. The CMS detector features a complex geometry with barrel, endcap and transition region as well as inhomogeneous detector modules – pixel and silicon strip detectors – with varying precision, thus previous work, particularly [6], can *not* be easily adapted to CMS. Therefore Hauth et al. [80] propose a new algorithm, following the general processing of the track finding CA presented in Section 4.1 but tailored towards the specific features of the CMS detector. The CA cells are defined via triplets of reconstructed hits which are more related to the hit triplets obtained from CMSSW seeding than the triplet notion of [6]. This gives rise to the need for an efficient – both in the physical as well as in the algorithmic sense – triplet finding algorithm as the foundation of the automaton. The design, implementation and evaluation of the OpenCL-based triplet finding algorithm is presented in Part II of this thesis. As the track model of both the triplet finding algorithm and the track reconstruction CA are closely related, its discussion is deferred to Part II.

Chapter 5

Spatial Data Structures

A data structure describes a particular method to organize and store data for efficient access via associated algorithms – such as insertion, search, deletion etc. Spatial objects like points in space, lines, surfaces and volumes do not only possess individual properties, including position and extent, but also feature a spatial relation to one another. Spatial data structures need to capture and exploit the geometric structure and properties of these objects in order to efficiently answer spatial queries, e.g. point location, nearest neighbor or range queries [149].

Range queries for points in space are of particular importance for particle track reconstruction. In general, given a set P of n points in d-dimensional space, a range query asks for all points inside a rectilinearly oriented d-orthotope Q. As presented in Section 2.3.4, the three-dimensional nature of the track reconstruction problem can be reduced to two dimensions by exploiting the layered structure of the CMS detector and treating barrel and endcap layers separately. Furthermore, even though hits are reconstructed from clusters of energy deposits, as discussed in Section 2.3.2, they are defined by a single position plus uncertainty and can thus be treated as point data. Therefore, a query for hits in detector layer k, which are compatible with a particle's trajectory, is defined by a query rectangle Q given by the extrapolation of the trajectory from layer $k-1$ to k. In the barrel region for instance, Q is defined by the predicted ϕ- and z-ranges.

This chapter introduces spatial data structures for two-dimensional range queries. The current CMSSW triplet seeding which is described in Section 5.1, employs a ϕ-sorted list to store the hits of a detector layer. Section 5.2 describes k-d trees, which have been recently introduced to CMSSW in order to decrease the number of retrieved hits during triplet finding. The following sections present quadtrees and r-trees as further spatial indices. While the former tree data structures are constructed in a data-driven manner, Section 5.5 discusses the space-driven uniform grid data structure.

5.1. CMSSW Triplet Seeding – ϕ-sorted List

As shown in Algorithms 2.1 and 2.2, the current pair and triplet finding algorithms use only one-dimensional range queries to retrieve compatible hits for seed building. All hits of a detector layer within the predicted ϕ-range are retrieved for compatibility checking.

Despite computing the admissible z-range – or r-range in the endcap – in the triplet finding algorithm, the information is disregarded for hit retrieval. For pair finding, the feasible z-range is not predicted, nevertheless the reduced number of retrieved hits could outweigh the effort for calculating the appropriate values of z.

Due to the one-dimensionality of the current queries, The hits of a particular detector layer are stored in a simple list, sorted by ϕ. Given a layer with n hits, constructing this data structure requires $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space. A query for hits in $[\phi_l, \phi_u]$ can be answered in $\mathcal{O}(\log n + r)$ time using binary search, with r denoting the number of retrieved hits. A wealth of sorting algorithms suitable for GPU-based processing exist in the literature, for instance odd-even merge sort [184] or sample sort [117]. Therefore this data structure *could* be used in a OpenCL-based triplet finding algorithm. However, the number of retrieved hits and the resulting combinatoric effort can be reduced by adding the z/r dimension to the search queries as facilitated by the data structures described in the following.

5.2. k-d Tree

A k-dimensional tree is a space-partitioning binary tree for points in $\mathbb{R}^k$ introduced by Bentley [23] and depicted in Figure 5.1. Each non-leaf node of the tree splits the space associated to it into two half-spaces by a $k-1$ dimensional hyperplane. The splitting dimension $d \in [1, \ldots, k]$ is chosen cyclically for each level of the tree, with the splitting hyperplane being spanned perpendicular to the d-axis. The hyperplane separates points smaller than a chosen pivot on the d-axis in the left subtree from the points greater than the pivot in the right subtree. To ensure a balanced k-d tree, the median of the points with respect to their coordinate in d is used as pivot. This would either require sorting the points in $\mathcal{O}(n \log n)$ or employing the complex linear time algorithm by Blum et al.

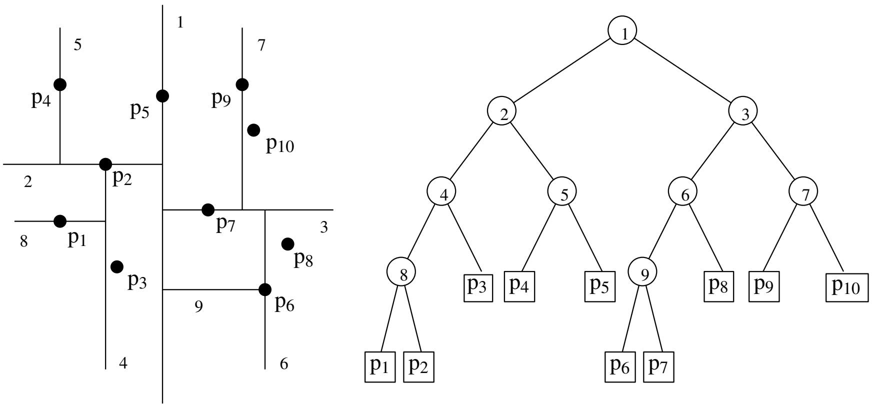


Figure 5.1.: Illustration of a k-d tree for two-dimensional space points [24]. The node numbers in the tree on the right correspond to the splitting lines on the left and exhibit the alternating splitting dimension with each tree level.

[28]. However, sampling-based median selection approaches yield sufficiently balanced trees for most inputs in constant time $\mathcal{O}(1)$ [22]. The splitting continues recursively until a half-space contains only one point, which is then added to the tree as leaf.

A k-d tree can be constructed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space, using either the linear time median selection algorithm by Blum et al. [28] or a sample-based approach with a fixed sample size. Employing a sorting-based median computation results in a runtime of $\mathcal{O}(n \log^2 n)$. Range queries with an axis-parallel query orthotope can be answered in $\mathcal{O}(n^{1-\frac{1}{k}} + r)$ time, with r reported points [23].

In preparation for high pile-up events, k-d trees have been introduced to the *particle flow* algorithm in the combined reconstruction of CMSSW [67]. The algorithm links reconstructed tracks to calorimetric clusters in the (η, ϕ)-plane and thus uses nearest neighbor queries on the data structure, which can be answered in average $\mathcal{O}(\log n)$ time. Recently, Reid [146] adapted CMSSW's triplet seeding algorithm to k-d trees. The modified algorithm performs a two-dimensional range query for compatible hits to a given hit pair. Reid [146] duplicates hits with an offset of 2π in ϕ in order to simplify the treatment of the angular wraparound. Furthermore, he expands the predicted z-range to maintain the algorithms physical efficiency. Construction of the k-d tree incurs some overhead in the triplet finding, nevertheless Reid [146] reports a reduction in overall seeding runtime of 5.6 % to 22 %, due to the reduced number of retrieved hits.

In graphics processing, k-d trees are widely spread, particularly in ray tracing applications. Therefore, several GPU-based algorithms exploiting this data structure exist. Santos et al. [153] review k-d tree traversal algorithms implemented in NVIDIA's CUDA. They propose a new traversal algorithm, minimizing global memory accesses by introducing *ropes*, pointers in each leaf to neighboring nodes. Zhou et al. [185] study the construction of k-d trees on GPUs and report favorable results both in speedup as well as quality of the built tree. Thus, k-d trees are a viable candidate for a triplet finding algorithm based on OpenCL, due to their proven applicability for triplet seeding and GPU suitability.

5.3. Quadtree

A quadtree partitions two-dimensional space by recursively dividing it into four quadrants [53]. Each internal node posses four children, either subtrees further partitioning the associated quadrant or leafs representing a single point in space, as shown in Figure 5.2. Therefore, the height h of a quadtree depends on the smallest distance d between any two points and is bound by $\log \frac{l}{d} + \frac{3}{2}$ for an initial square space of length l. It can be constructed in $\mathcal{O}(hn)$ time in linear space [163]. The worst case bound for range queries is $\mathcal{O}(n)$, for instance if all contained points are spread closely along the sides of the initial square and the inner region is queried.

Due to their simplicity, quadtrees are widely applied in graphics algorithms, e.g. for collision detection or rendering, but also for simulating Cellular Automata [147]. Zhang et al. [183] employ quadtrees to encode geospatial data on GPUs; Yusov and Turlapov [182] present a GPU-optimized quadtree for terrain rendering.

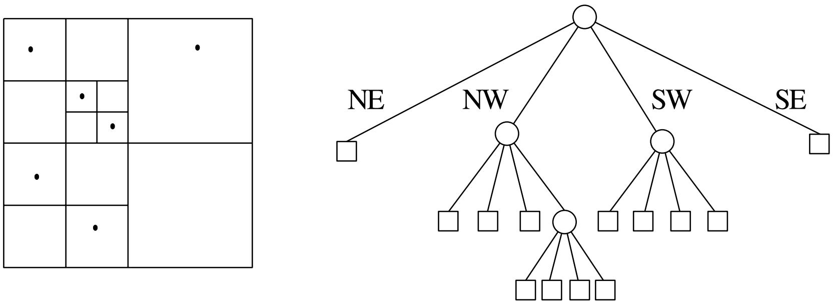


Figure 5.2.: Illustration of a quadtree for two-dimensional space points [24]. The exhibited quadtree is unbalanced due to the non-uniform spatial distribution of the data points.

Quadtrees can be considered as an „adaptive“ grid data structure as the size and number of the grid cells varies with the densities of points in that region. Due to a particle's multiple interactions with the detector material, several hits may be reconstructed in close proximity, resulting in a needlessly fine-grained quadtree. This effect can be mitigated by imposing a minimum quadrant size and collecting several hits within one leaf. Moreover, the unbalanced height of a quadtree seems unfavorable for lockstep processing in the OpenCL execution model. Therefore, quadtrees should *not* be the primary choice for an OpenCL-based triplet finding algorithm.

5.4. R-Tree

R-trees partition two-dimensional space by recursively grouping nearby objects within their Minimum Bounding Rectangle (MBR) [74]. A leaf's MBR contains a single space point. An inner node's MBR envelopes the MBRs of its subtrees and can therefore be considered a more coarse-grained approximation of the data points. Figure 5.3 illustrates a R-tree for two dimensional space. R-trees are balanced search trees and are optimized for external memory, being similar to B-trees [20]. Nodes are stored in pages of up to M nodes, with a minimum guaranteed fill of m. Objects are inserted into the node whose MBR needs to be enlarged the least to encompass the new item. The operation requires $\mathcal{O}(M \log n)$ time if no pages need to be split due to overfilling, since all nodes of a page are inspected to determine the most suitable one. For page splitting however, $\mathcal{O}(M^2)$ operations are performed in order to minimize the total area of the two created pages. Furthermore, the overlap of the two created pages ought to be minimized as well, since range queries can be answered in $\mathcal{O}(\log n)$ in the absence of overlap, yet require $\mathcal{O}(n)$ in the worst case in the presence of overlapping MBRs.

A plethora of variants of R-trees is proposed in literature and practice [121], e.g. R+-trees prohibit overlapping nodes by allowing the storage of one point multiple times in the tree [158]. Kim and Nam [104] present a three-phase R-tree traversal algorithm for multi-dimensional range queries on GPU. Luo et al. [119] describe a parallel R-tree construction algorithm suitable for GPU execution.

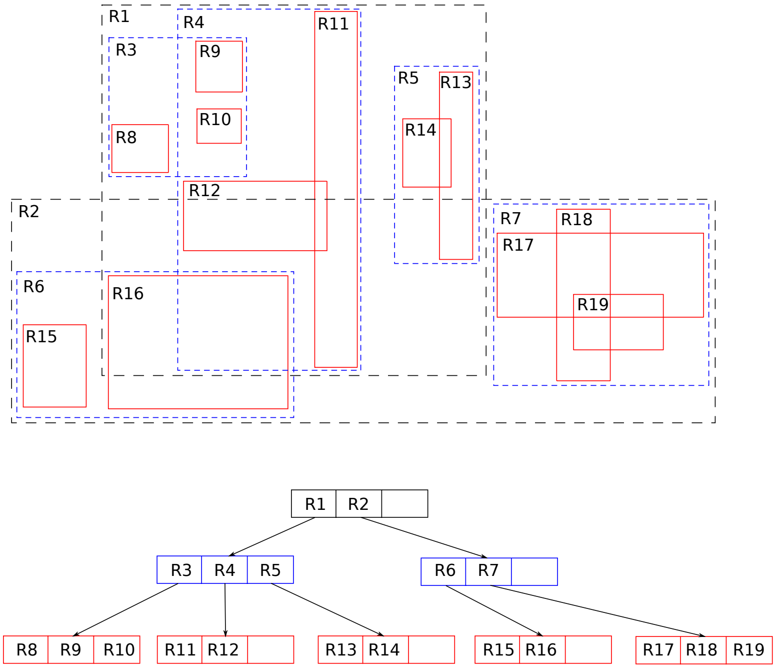


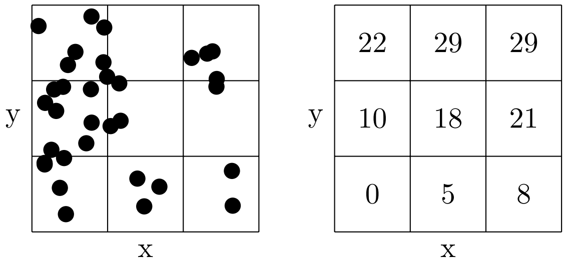
Figure 5.3.: Illustration of a R-tree for two-dimensional MBRs [111]. Due to the overlapping Minimum Bounding Rectangles – e. g. R3 and R4 – range queries have a worst case time complexity of $\mathcal{O}(n)$.

As R-trees are balanced, they are more appropriate for the OpenCL execution model than quad trees. Notwithstanding, they bear the same disadvantage of suffering from many points in close proximity, as common with reconstructed hits. Furthermore, the proposed GPU-based construction and query algorithms are more complex than the ones for other spatial data structures. Thus, preference should be given to a simpler data structure for efficient hit retrieval in triplet finding.

5.5. Uniform Grid

A grid is a regular tessellation of a d-dimensional space into contiguous cells [57]. Figure 5.4 illustrates a grid data structure as a uniform overlay over space – independent of the distribution of the data points. Given a unit d-hypercube, each dimension i is subdivided into G_i cells of size $\frac{1}{G_i}$, $i \in [1, \dots d]$, yielding a total of $c = \prod_{i=1}^{d} G_i$ cells. The data structure requires $\mathcal{O}(n + c)$ space and can be constructed in-situ in $\mathcal{O}(dn \log n)$ time or ex-situ in linear time and extra space [5]. A uniform distribution of points in space is most suitable for grid data structures. In the worst case, point and range queries require $\mathcal{O}(n)$ time, viz. if all points are gathered by one cell. Assuming evenly spread space points, a cell is occupied on average by $m = \frac{n}{c}$ points, resulting in a point location time of $\mathcal{O}(m)$ and range query time of $\mathcal{O}(dm + r)$, with r retrieved points.

Grid data structures haven been applied to manifold applications [5]. Kalojanov and



22	29	29
10	18	21
0	5	8

Figure 5.4.: A grid data structure as uniform overlay over two-dimensional space. The left grid depicts the data points and their distribution over the grid cells. The right grid shows the associated data structure with offsets into the array of data points, sorted by their grid cell affiliation.

Slusallek [98] construct a uniform grid on a GPU, using a multi-pass approach, for ray tracing. Joselli et al. [95] employ a grid for crowd simulation studies on GPU. The ALICE Collaboration [6] constructs a grid data structure ex-situ on CPU, before transferring the data to the device.

Due to its simplicity, a grid data structure appears to be an ideal candidate for OpenCL-based triplet finding. Even though k-d trees achieve lower query times, particularly for non-uniformly distributed points, their construction and traversing is much more cumbersome in the OpenCL programming model [98].

Part II.

Parallel Triplet Finding with OpenCL

Chapter 6

Overview

In CMSSW, hit triplets are seeding the Kalman filter-based track reconstruction, as they provide an initial estimate of the track parameters. For Cellular Automaton-based tracking approaches, cells are often defined as the track segment given by a hit pair. However, defining cells via hit triplets allows for the elimination of implausible hit combinations at an early stage by applying filter criteria not applicable to pairs of hits. Thus, the computational effort invested in triplet finding results in lower combinatoric complexity for the later reconstruction steps. Since triplets need to be found for all detector layers in CA-based track reconstruction, the need for an efficient and performant triplet finding algorithm becomes more evident. Hauth et al. [79] proof the parallelization potential of CMSSW's triplet finding by concurrently processing the hit pairs generated in the first step of Algorithm 2.2 with Intel's Threading Building Blocks (TBB) [105, 180]. Finding triplets in all detector layers simultaneously further increases the level of exploitable parallelism. By reimplementing triplet finding in OpenCL, this parallelism becomes accessible to a wide range of hardware platforms, including multi-core and many-core CPUs as well as GPUs, in a portable manner as required for software designed for execution in the Worldwide LHC Computing Grid (WLCG).

As described in Section 2.2, CMSSW is implemented in C++ and makes extensive use of inheritance and polymorphism. The `Event` and `EventSetup` containers are vital for data transportation along the module execution chain and provide access to the data via smart pointers [93]. The triplet finding algorithm itself relies on abstract classes to interface various pair generators, third hit predictors and multiple scattering parameterizations. Furthermore, the required computations are scattered amongst a vast set of interwoven classes, particularly for multiple scattering effects. OpenCL is *not* object-oriented but data-centered, requiring plain memory layouts, such as arrays-of-structs or structs-of-arrays, and „C-style“ function implementation [140]. Therefore, a complete *reimplementation* of the algorithm is necessary to be compliant with OpenCL's requirements. This gives rise to the opportunity to completely *redesign* triplet finding in order to tailor it towards performant parallel processing, disposing some of the „over-engineered“ calculations in favor of simpler geometric computations and parameterizations [110]. The design of the novel triplet finding pursues the following goals:

- parallelization of the entire pair and triplet finding process on the intra-event level – finding triplets in several layer combinations concurrently – and the inter-event

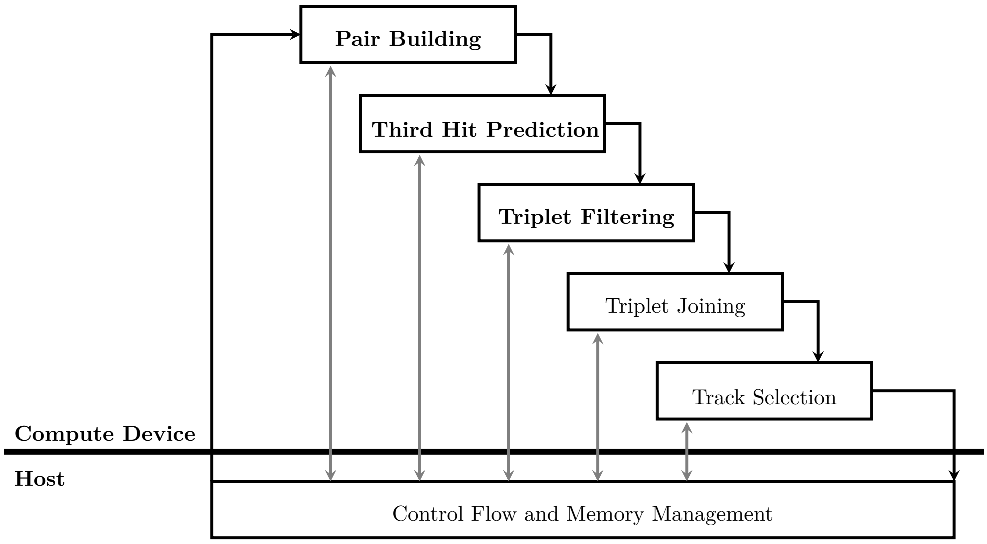


Figure 6.1.: High level processing steps of OpenCL-based CA track finding. Algorithms for the steps required for triplet finding (boldface font) are presented in this thesis. The track formation steps are described by Hauth et al. [80].

level – processing events of the same luminosity section simultaneously;

- use of computationally inexpensive criteria to distinguish *valid* triplets – belonging to the track of a charged particle – from *fake* ones before employing more involved filters;
- storage of required non-event data, such as detector geometry and alignments, and event data, i. e. hits, in data structures pertinent to the OpenCL execution and memory model and
- algorithm implementation in accordance with OpenCL peculiarities, i. e. branch-avoidance, local memory and lack of dynamic memory allocation within kernels.

Figure 6.1 depicts the high level processing steps of the newly designed triplet finding algorithm (boldface font). Albeit not within the scope of this thesis, the figure moreover presents the further processing of the triplets in the CA-based track finding algorithm outlined in Section 4.2. All operations are executed as kernels on the OpenCL compute device in order to minimize data transfers between device and host. Since the number of results a kernel produces is unpredictable, the host needs to compensate for the lack of dynamic memory allocation in a kernel. Allocating ample amounts of memory prior to kernel execution to accommodate *all* possible combinations is infeasible, as the number of combinatorial possible triplets can reach 10^{10}, refer to Section 2.3.4. Hence, every processing step is designed as a two-pass algorithm: a *counting* pass to determine the number of outputs produced and a *storing* pass to actually write the output to memory [34, 45, 152]. Between both passes, the host allocates memory on the device for the identified amount of output. Algorithm 6.1 provides a high-level overview of the interplay

Algorithm 6.1 Data flow of OpenCL triplet finding algorithm.

```
Require: event hits, detector geometry
geom ← Build geometry lookup table
transfer geom to device
transfer event hits to device
kernel build grid data structure for event hits
parfor triplets of detector layers do
    kernel $n$ ← number of valid pairs                         ▷ pair building
    transfer $n$ to host
    allocate memory for $n$ pairs
    kernel store valid pairs
    kernel $n$ ← number of triplet candidates                  ▷ third hit prediction
    transfer $n$ to host
    allocate memory for $n$ triplet candidates
    kernel store triplet candidates
    kernel $n$ ← number of valid triplets                      ▷ triplet filtering
    transfer $n$ to host
    allocate memory for $n$ triplets
    kernel store valid triplets
end parfor                                   ▷ continue with CA-based track finding
```

of host and compute device in the execution of the triplet finding algorithm. Details of the two-pass concept are described in Chapter 9 together with the geometric computations performed by each step. As discussed in Section 3.3, thread divergence due to branching is one of the main concerns when designing algorithms for OpenCL. A major source of branching are differences in the geometric calculations required for the barrel, endcap and transition regions. In order to avoid thread divergence during execution, each region should be addressed by a specific kernel. In this thesis, the validity of the pursued algorithmic approach and the merit of employing OpenCL for triplet finding are assessed with algorithms designed for the *barrel* region. However, the developed concepts can be applied to the endcap and transition regions as well, even though their treatment is beyond the scope of this work.

Algorithm 6.1 not only highlights the algorithms involved in triplet finding but also the employed data structures for event and non-event data. The detector geometry is stored as a compressed lookup table to reduce the memory required for non-event data. As identified by Chapter 8, a uniform grid data structure is used to efficiently query for hits within a detector region of interest. Chapter 8 further details the utilized data structures. Prior to the discussion of algorithmic aspects of the new triplet finding, Chapter 7 introduces the criteria used to distinguish valid triplets from fake ones. In Chapter 10 the physical and algorithmic properties of the developed triplet finding are evaluated.

Chapter 7

Filter Criteria

In a trivial approach, a track finding CA would comprise a cell for every possible combination of three hits in adjacent detector layers. In high pile-up scenarios, this would result in an unfeasibly large automaton. Thus, discriminating *valid* hit triplets, originating from a single particle's trajectory, from those composed of hits from several particles' interactions with the detector material, referred to as *fakes* or background, at an early stage is vital to limit the automaton's size and hence the runtime of the CA-based track finding algorithm. In this chapter, properties of valid hit triplets are examined and suitable criteria are derived to distinguish them from background triplets. In Section 7.1, the properties of a triplet's azimuthal and polar angles are studied. Section 7.2 describes an efficient method to obtain the parameters of the helical path defined by a triplet in order to compute the path's minimum distance to the beam line. This chapter presents the physical concepts of the employed filter criteria; the algorithmic implementations are described in the appropriate sections of Chapter 9.

The derived criteria can not only be used to assess triplets, but are also applicable to determine suitable combinations of triplets in the later processing of the CA.

7.1. Angular Constraints

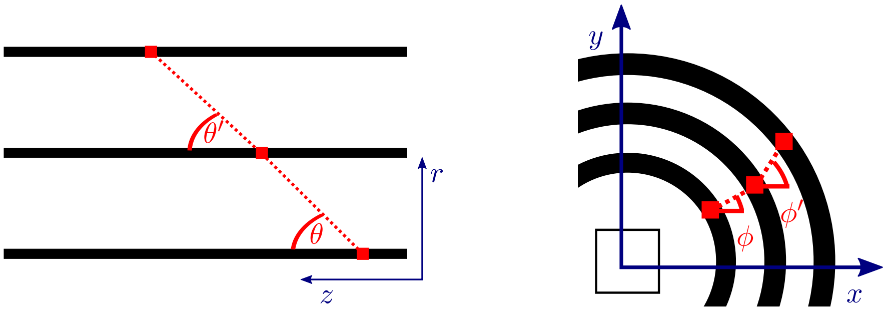
A charged particle follows a helical trajectory in a magnetic field, due to the Lorentz force

$$\mathbf{F}_L = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}),$$

where q and $\mathbf{v}$ denote the charge and velocity of the particle, respectively. The particle is subject to a magnetic field $\mathbf{B}$ and an electric field $\mathbf{E}$. Neglecting the electric field $\mathbf{E}$ and expressing $\mathbf{v}$ in terms of the particle's momentum $\mathbf{p}$ yields

$$\mathbf{F}_L = q\left(\frac{\mathbf{p}}{m} \times \mathbf{B}\right). \tag{7.1}$$

The magnetic field produced by the CMS solenoid is mainly directed along the z-axis and can be idealized to $\mathbf{B} = \begin{pmatrix} 0 & 0 & B_z \end{pmatrix}^{\mathrm{T}}$ [44]. Thus, considering an arbitrary momentum



(a) Constraint on longitudinal bending by restricting $\frac{\theta'}{\theta} \leq d\theta$.

(b) Constraint on transversal bending by restricting $|\phi' - \phi| \leq d\phi$.

Figure 7.1.: Schematic angular criteria to discriminate valid and fake triplets.

vector $\mathbf{p} = \begin{pmatrix} p_x & p_y & p_z \end{pmatrix}^{\mathrm{T}}$,

$$\mathbf{F}_L = \frac{qB_z}{m} \begin{pmatrix} p_y \\ -p_x \\ 0 \end{pmatrix}, \tag{7.2}$$

i.e. a charged particle experiences the Lorentz force merely in the transverse plane, orthogonal to its instantaneous path of motion.

In general, the curvature κ of a curve in three-dimensional space is given by

$$\kappa = \frac{\|\mathbf{v} \times \mathbf{a}\|}{\|\mathbf{v}\|^3}.$$

In conjunction with Equation (7.1)

$$\kappa = q\frac{\|\mathbf{p} \times (\mathbf{p} \times \mathbf{B})\|}{\|\mathbf{p}\|^3} = q\frac{\left\|(\mathbf{p} \cdot \mathbf{B})\mathbf{p} - \|\mathbf{p}\|^2 \mathbf{B}\right\|}{\|\mathbf{p}\|^3}.$$

Considering the idealized magnetic field $\mathbf{B} = \begin{pmatrix} 0 & 0 & B_z \end{pmatrix}^T$, the Lorentz force influences solely the particle's transverse motion and depends only on the its p_T, as shown by Equation (7.2). Therefore, $\mathbf{p}$ can be projected onto its transverse components, $\mathbf{p}_T = \begin{pmatrix} p_x & p_y & 0 \end{pmatrix}^{\mathrm{T}}$, for the derivation of the curvature κ due to the Lorentz force. Since $\mathbf{p}_T \perp \mathbf{B}$, κ is given by

$$\kappa = q\frac{\left\|-\|\mathbf{p}_T\|^2 \begin{pmatrix} 0 & 0 & B_z \end{pmatrix}^{\mathrm{T}}\right\|}{\|\mathbf{p}_T\|^3} = \frac{qB_z}{p_T}. \tag{7.3}$$

Ideally, the particle's trajectory is thus bent merely in the transverse plane and follows a straight line in the longitudinal one.

A *valid* hit triplet, associated with a charged particle, should therefore be compliant with this track model. This leads to the following filter criteria, defined by the two segments constituting the triplet, as depicted by Figure 7.1:

Longitudinal Bending Considering the idealized magnetic field, the track's polar angle θ should be constant along all segments, refer to Figure 7.1a. Even though a particle's trajectory might be bent in the longitudinal plane due to multiple scattering or non-ideal magnetic field conditions, the ratio of the θ angles of a triplet's segments should by bound by

$$\left|\frac{\theta'}{\theta} - 1\right| \leq d\theta. \tag{7.4}$$

The bound $d\theta$ also needs to account for the uncertainty of a hit's position due to the limited detector resolution.

Transversal Bending The curvature of a particle's path is inverse proportional to its p_T. Thus, bounding the difference of the ϕ angles of a triplet's segments by imposing

$$|\phi' - \phi| \leq d\phi \tag{7.5}$$

constrains the admissible minimum p_T of the associated particle. Furthermore, energy loss due to the particle's interaction with the detector material and multiple scattering influence the trajectory and need to be considered in $d\phi$. Similarly to $d\theta$, the detector resolution needs to be accounted for as well. Figure 7.1b depicts the $d\phi$ filter criterion schematically.

As *fake* triplets do not resemble a particle's path, they are not confined by these requirements. Thus, the two criteria are suitable to discriminate between fake and valid triplets. The choice of values for $d\theta$ and $d\phi$ influences the achievable *efficiency* (Equation 2.3) and resulting *fake rate* (Equation 2.4) of the two filter criteria. Section 10.2 presents the derivation of suitable values for $d\theta$ and $d\phi$.

7.2. Transverse Impact Parameter Constraint

Tracks of particles emerging from the initial proton-proton collision originate close to the Interaction Point. The transversal distance of a trajectory's point of closest approach to the beam line is referred to as Transverse Impact Parameter, denoted by d_0 and depicted in Figure 7.2. As detailed in Section 2.3.4, restricting d_0 reduces the combinatorial complexity of CMSSW's triplet seeding significantly. Hence, the TIP is employed as discriminator for valid and fake triplets in the presented triplet finding algorithm as well.

In order to determine the TIP, a circle must be fitted to the three hits of the triplet. The CMSSW triplet seeding utilizes a conformal mapping technique for circle fitting [77]. Strandlie et al. [173] present an alternative approach, transforming the problem of fitting a circle to points in the plane to fitting a plane to points on a Riemann sphere. The method bears the merit of being capable to address the effects of multiple scattering [172] and is found to be computationally more efficient than the conformal mapping approach by Frühwirth et al. [62]. The number of required divisions for the mapping of the points in the plane to the three-dimensional surface can be reduced by employing a circular paraboloid instead of a Riemann sphere [60]. The presented triplet finding algorithm therefore employs this circle fitting technique.

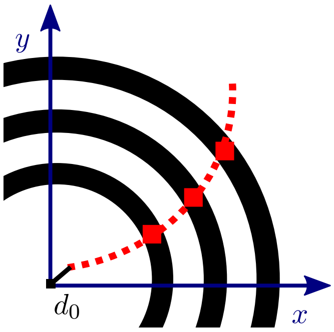


Figure 7.2.: Schematic of the Transverse Impact Parameter d_0, i.e. the distance of a trajectory's point of closest approach to the beam line in the xy-plane.

Given a hit's transverse coordinates (x, y), it is mapped to the circular paraboloid in (u, v, w) space by

$$u = x \qquad v = y \qquad w = x^2 + y^2.$$

Points on the (x, y)-plane lying on a circle with center $\mathbf{c} = (a, b)$ and radius R are subject to

$$(x - a)^2 + (y - b)^2 = R^2.$$

Algebraic manipulation and mapping (x, y) to the paraboloid reveals (ibid.)

$$\begin{pmatrix} -2a \\ -2b \\ 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} = R^2 - a^2 - b^2. \tag{7.6}$$

Equation (7.6) resembles the equation of a plane in space, $\mathbf{nx} = -c$, with unit normal vector $\mathbf{n}$ and distance from the origin c. Therefore, the parameters of the circle in (x, y) can be easily obtained from the plane fitted to the mapped hits in (u, v, w). Frühwirth et al. [60] fit n mapped hits $\mathbf{u}_i$ to a plane by minimizing

$$S = \sum_{i=1}^{n} \mathbf{nu}_i + c$$

with respect to $\mathbf{n}$ and c. For triplet finding with only mapped hits $\mathbf{u}_{1\ldots3}$, fitting is futile, as $\mathbf{n}$ can be obtained directly by

$$\mathbf{n} = \frac{\overrightarrow{\mathbf{u}_1\mathbf{u}_2} \times \overrightarrow{\mathbf{u}_1\mathbf{u}_3}}{\|\overrightarrow{\mathbf{u}_1\mathbf{u}_2} \times \overrightarrow{\mathbf{u}_1\mathbf{u}_3}\|},$$

with $\overrightarrow{\mathbf{xy}}$ denoting the vector from $\mathbf{x}$ to $\mathbf{y}$. The value of c is then given by

$$c = -\mathbf{nu}_i$$

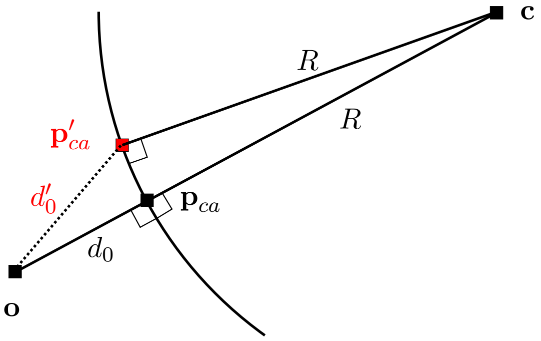


Figure 7.3.: Calculating d_0 based on the point of closest approach $\mathbf{p}_{ca}$ of the particle's trajectory. The intersection between the circle's circumference and the line $\overrightarrow{\mathbf{co}}$ between circle center and origin uniquely defines $\mathbf{p}_{ca}$.

for arbitrary $i \in [1, 2, 3]$. Thus, the circle parameters are (ibid.)

$$\mathbf{c} = \begin{pmatrix} a \\ b \end{pmatrix} = -\frac{1}{2\mathbf{n}_z} \begin{pmatrix} \mathbf{n}_x \\ \mathbf{n}_y \end{pmatrix}$$

$$R = \sqrt{\frac{1 - \mathbf{n}_z^2 - 4c\mathbf{n}_z}{4\mathbf{n}_z^2}}.$$

The TIP d_0 can be obtained from the distance of the point of closest approach, $\mathbf{p}_{ca}$, to the origin $\mathbf{o}$. As Figure 7.3 depicts, $\mathbf{p}_{ca}$ is uniquely defined by the intersection of the circle's circumference with the line $\overrightarrow{\mathbf{co}}$, thus

$$d_0 = \|\mathbf{p}_{ca}\| = \left\| \mathbf{c} + R \cdot \frac{\overrightarrow{\mathbf{co}}}{\|\overrightarrow{\mathbf{co}}\|} \right\|. \tag{7.7}$$

Proof. Suppose $\exists \mathbf{p}'_{ca} \notin \overrightarrow{\mathbf{co}}$, with $\|\mathbf{p}'_{ca}\| < \|\mathbf{p}_{ca}\|$, as shown in Figure 7.3. Every point on the circle's circumference has distance R to the circle's center. If $\mathbf{p}'_{ca}$ existed then $\|\mathbf{p}'_{ca}\| + R < \|\overrightarrow{\mathbf{co}}\| = \|\mathbf{p}_{ca}\| + R$, violating the triangle inequality. Hence, $\mathbf{p}_{ca} \in \overrightarrow{\mathbf{co}}$, yielding the correctness of Equation (7.7). ☐

Furthermore, the z-coordinate of $\mathbf{p}_{ca}$ defines the *longitudinal* impact parameter z_0 of the track. The parameter can be obtained by either using a straight line approximation of the trajectory in the r-z-plane or by employing a circle fit as described above, mapping coordinates (r, z) to the circular paraboloid. As presented in Table 2.1, tracks may originate at z up to ±30 cm, therefore z_0 was not chosen as primary discriminator for valid and fake triplets.

The derivation of suitable d_0 values to reject a large portion of the fake background while still remaining high efficiency is presented in Section 10.2.

Chapter 8

Data Structures

Data structures are a fundamental building block of all data processing applications. The choice of data representation and access techniques is influenced by

- the structure and properties of the data to be stored, e. g. pointers, numbers, one- or multi-dimensional point data, volumes, etc.;
- the calculations to be performed with the data and their requirements on value ranges and precision;
- the access pattern to the data – for instance sequential access or random access via point or range queries – and
- the underlying architecture's memory and programming model, including memory hierarchies, available data types and preferred model of concurrency.

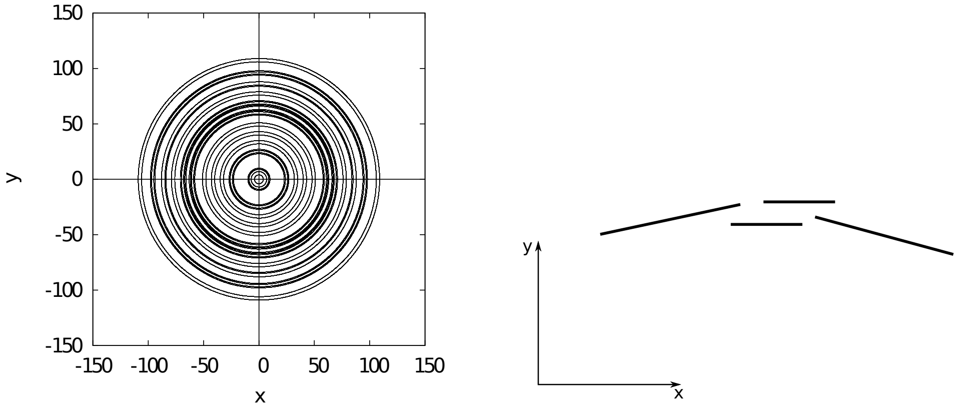
This chapter introduces the employed data structures in the presented OpenCL-based triplet finding algorithm. Two main types of data can be distinguished for this application

- event-*independent* data, such as detector geometry and alignment information or magnetic field data, and
- *event* data, namely *hits* – energy deposits of traversing particles in the detector material.

Since effects due to the magnetic field are captured by the parameterized filter criteria introduced in the previous chapter, solely the geometric description of the detector layout is required for the presented algorithm. Section 8.1 introduces the data structure used to store this information on the compute device. Data structures pertinent to space point storage are discussed in Chapter 5, with the uniform grid being identified as the most suitable for OpenCL-based triplet finding. Section 8.2 describes the implementation of the grid data structure.

8.1. Detector Geometry

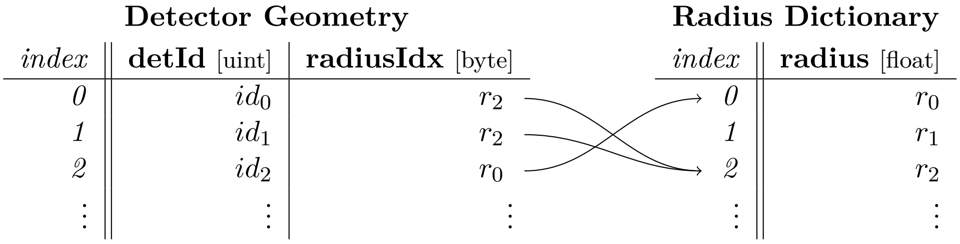
The CMS inner tracking system comprises 1 440 detector *modules* in the pixel- and 15 148 modules in the silicon strip detector [154]. Each module is defined by a three-dimensional



(a) Plot of radius of the 10 548 `DetUnits` of the barrel, exhibiting their clustering around a few discrete values.

(b) Illustration of overlapping detector modules within one detector layer.

Figure 8.1.: Layered construction of the CMS inner tracking system.



Detector Geometry

index	**detId** [uint]	**radiusIdx** [byte]
0	id_0	r_2
1	id_1	r_2
2	id_2	r_0
⋮	⋮	⋮

Radius Dictionary

index	**radius** [float]
0	r_0
1	r_1
2	r_2
⋮	⋮

Figure 8.2.: Schematic drawing illustrating the compressed `DetUnit` radius storage.

position in the global coordinate system, a *bounded surface* describing the extent of the module in module local coordinates and a *rotation* to obtain the global r-, ϕ- and z-span of the module. Storing the position and surface extents requires two **`float3`** vectors in OpenCL, the rotation is given by a 3×3 matrix. In total, the description of *all* detector modules occupies 995 kByte of memory.

In the presented triplet finding algorithm, the geometric description of the barrel region of the detector is merely utilized for prediction purposes. Given a hit on detector layer k, the feasible z-range for a hit in layer $k + 1$ is calculated based on the radial distance traveled by the particle between the two layers, refer to Section 9.4 for details. Therefore, only the radial component of the detector modules' positions need to be stored on the compute device. Thus, 43 kByte suffice to store the radial positions of the 10 548 detector modules in the *barrel* region – for the entire tracker 67 kByte are required.

The detector geometry is read in a random access pattern during third hit prediction since hits, even of small Euclidean distance, can reside on different detector modules, particularly in overlapping regions. As random accesses to global memory are slow, the geometry information should be stored in local memory. Modern GPU architectures, such as NVIDIA's Kepler architecture, provide 48 kByte of local memory per compute unit [138], Intel's Core i7 3960x features 32 kByte local memory (L1 cache) per core [91]. Therefore, reducing the amount of memory required for geometrical information storage is required on the CPU, but also beneficial on the GPU as further data may be cached in local memory.

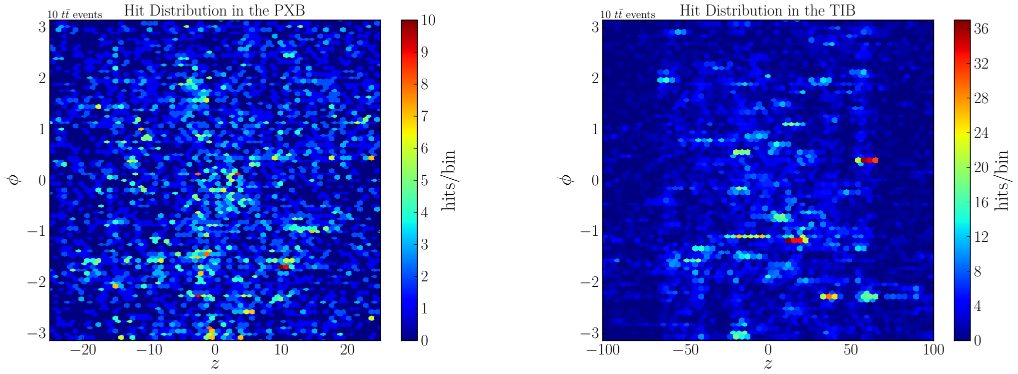
Figure 8.1a illustrates the distribution of r-values for all 10 548 detector modules of the barrel region. As the layered construction of the CMS detector would suggest, the module radii cluster around a few discrete values. However, there are more clusters than the number of layers in the barrel – 13 – due to the overlapping assembly of the detector modules within one layer, depicted in Figure 8.1b.

Since the radial detector module positions are *solely* employed for prediction and *not* for final triplet validation, it suffices to compute the feasible z-range with approximations of the radii. Therefore, the regular structure of the radii can be exploited by a dictionary-based compression [143]. Figure 8.2 depicts the employed compression scheme. A *radius dictionary* stores the approximated radial positions as `float` values. In the *detector geometry* data structure, each detector module is associated to its nearest radius value via an index into the dictionary. Approximating the radii with two digit precision results in 120 entries in the dictionary, thus being addressable by a single byte index; three digit approximation would lead to 290 values, hence exceeding the byte value range. Therefore the less precise approximation is chosen, as the inaccuracy can be mitigated by employing a slightly larger prediction window, refer to Section 9.4. In that scheme, the information about the detector modules' radial position occupies 11 kByte of memory, resulting in a compression ratio of 0.26. Additionally, the minimum and maximum radius of each layer is stored in a supplement to the detector geometry structure, requiring two `float`s per layer, thus 104 Byte.

In later iterations of the hit pair building and triplet prediction algorithms, the lookup of a hit's detector module radius is replaced by calculating the hit's radius based on its reconstructed position. Therefore, only the information about the minimum and maximum radii of the detector layers is required by the algorithms. Nevertheless, the presented concept of compressed detector geometry storage might of interest to other algorithms implemented on GPGPUs/in OpenCL. For instance in the simulation of particle passage through the detector, where hits need to be „placed“ on detector modules along the particle's trajectory.

8.2. Event Hit Data

Reconstructed hits from particles' interactions with the detector material are the central data for triplet finding. Hit data is accessed in both sequential and random access manner via range queries. The three-dimensional measurement points are grouped by detector layer, therefore reducing the dimensionality of their spatial information to two dimensions. As described in Section 2.3.3, a hit in a barrel layer is defined by its ϕ- and z-coordinate, since r is defined by the detector geometry. Due to the overlapping of detector modules in one layer (refer to Figure 8.1b), hits of a layer can differ in their



(a) Pixel barrel layers PXB.

(b) Inner silicon strip layers TIB.

Figure 8.3.: Occupancy of the PXB and TIB during $t\bar{t}$ events over z and ϕ. Apart from a few clusters, the hits are distributed in a nearly uniform manner in the z-ϕ-plane.

actual r value. Nevertheless, triplet finding considers three hits from *different*, adjacent detector layers, therefore validating this simplification.

Chapter 5 introduces various spatial data structures applicable for two-dimensional point data. Out of the presented data structures, the k-d tree and the uniform grid appear to be the most viable candidates. A grid data structure is employed in this work for the following reasons:

- grid data structures are most suitable for uniformly distributed points. Due to particle jets which have a high local track density, the uniformity of the input must be scrutinized. Figure 8.3 shows the occupancy of the PXB and double-sided TIB layers during simulated $t\bar{t}$ events, which can contain six or more jets [42]. Moderate clustering can be identified. Particularly in the PXB, the central z-region $|z| < 2\,\text{cm}$ is more occupied than the outer areas. However, most regions of the detector exhibit a uniform distribution of hits, thus rendering grid data structures a suitable choice.

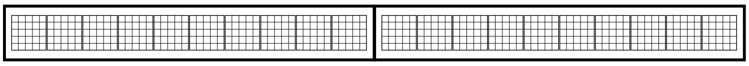
- accessing a particular grid cell takes $\mathcal{O}(1)$ time. By aligning grid size and prediction window size, no search within a grid cell must be performed to determine the beginning or end of a search range. The hits within a cell occupy contiguous memory, allowing efficient prefetching of hits within the predicted range.

- construction of the data structure can be performed efficiently on the compute device, exploiting standard operations – in particular prefix sums – which are employed for other steps of the triplet finding as well, refer to Chapter 9.

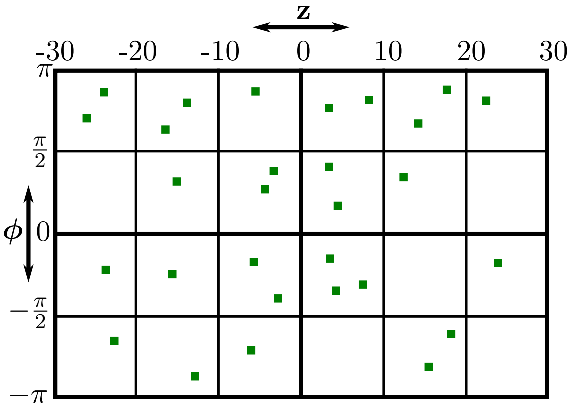
Each hit h is characterized by

- a hit identification number h_{id} (`uint`),

- its global coordinates h_x, h_y and h_z (`float`s),



(a) Contiguous memory layout of multiple events (outer boxes) and detector layers (small boxes). Within each layer, hits are stored in a grid data structure.



(b) Schematic of grid data structure partitioning the detector layer in z and ϕ cells.

Figure 8.4.: Event hit data organization on the compute device

- the detector layer h_l it is associated with,
- the detector module h_{det} the particle interacted with, represented as index into the geometry data structure, and
- the event number h_e (all **`uint`**s).

The data is organized as struct-of-arrays to allow coalesced memory accesses when properly aligned work-items read a specific hit characteristic concurrently. Figure 8.4a illustrates the contiguous memory layout of multiple events and detector layers. Within a detector layer, hits are partitioned by the uniform grid data structure, reaching from z_{min} to z_{max} and $\phi_{\text{min}} = -\pi$ to $\phi_{\text{max}} = \pi$, as depicted in Figure 8.4b. The detector geometry defines the extent in the z-dimension, ranging from -300 cm to 300 cm for the CMS tracker. The z and ϕ dimensions are subdivided into $\#_z$ and $\#_\phi$ segments, respectively, yielding cells of size $\frac{z_{\text{max}} - z_{\text{min}}}{\#_z} \times \frac{2\pi}{\#_\phi}$. The impact of the choice of grid granularity on the algorithmic and physical performance is examined in Chapter 10. For each grid cell, an index into the hit array with the first hit belonging to that cell must be stored. Therefore, considering $\mathcal{E}$ concurrently processed events and $\mathcal{L}$ detector layers with a total of $\mathcal{N}$ hits, the memory consumption for event data storage is given by

$$\mathcal{N} \cdot 28\,\text{Byte} + \mathcal{E} \cdot \mathcal{L} \cdot \#_z \cdot \#_\phi \cdot 4\,\text{Byte}.$$

An ex-situ two-pass algorithm is employed for grid construction, presented in Algorithm 8.1:

Algorithm 8.1 Ex-situ algorithm for grid data structure construction.

```
Input:  $\mathcal{E}$ concurrent events, $\mathcal{L}$ detector layers, $\mathcal{N}_{e,l}$ hits in layer $l$ of event $e$
        $h_{i,l,e}$ event hits
        $[z_{\min}, z_{\max}]$ $[\phi_{\min}, \phi_{\max}]$ grid extent
        $\#_z$, $\#_\phi$ number of grid cells
Output: $h'_{i,l,e}$ restructured event hits
        grid indices
transfer event hits to device
allocate grid size $\mathcal{E} \cdot \mathcal{L} \cdot \#_z \cdot \#_\phi$ on device
kernel determine cell bounds begin
    $(t, l, e) \leftarrow (\texttt{globalID}_1, \texttt{globalID}_2, \texttt{globalID}_3)$    ▷ (thread, layer, event)
    allocate gridLocal size $\#_z \cdot \#_\phi$ on local    ▷ if $4\#_z \cdot \#_\phi$ Byte $<$ local memory
    for $i = t \xrightarrow{\text{+work-group size}} \mathcal{N}_{e,l}$ do
        $h \leftarrow h_{i,l,e}$
        $\text{cell}_z \leftarrow \lfloor \frac{h_z - z_{\min}}{z_{\max} - z_{\min}} \rfloor$
        $\text{cell}_\phi \leftarrow \lfloor \frac{h_\phi - \phi_{\min}}{\phi_{\max} - \phi_{\min}} \rfloor$
        $\text{atomicInc}(\text{grid}_{\text{cell}_z, \text{cell}_\phi})$    ▷ if allocated: gridLocal
    end for
    $\text{grid}^{l,e} \leftarrow \text{gridLocal}$    ▷ if required
end
kernel prefixSum(grid)
allocate $h'$ size $\mathcal{N}$ on device
allocate written size $\mathcal{E} \cdot \mathcal{L} \cdot \#_z \cdot \#_\phi$ on device    ▷ if $4\#_z \cdot \#_\phi$ Byte $>$ local memory
kernel store hits in cells begin
    $(t, l, e) \leftarrow (\texttt{globalID}_1, \texttt{globalID}_2, \texttt{globalID}_3)$    ▷ (thread, layer, event)
    allocate gridLocal size $\#_z \cdot \#_\phi$ on local    ▷ if $8\#_z \cdot \#_\phi$ Byte $<$ local memory
    $\text{gridLocal} \leftarrow \text{grid}^{l,e}$
    allocate written size $\#_z \cdot \#_\phi$ on local    ▷ if $4\#_z \cdot \#_\phi$ Byte $<$ local memory
    for $i = t \xrightarrow{\text{+work-group size}} \mathcal{N}_{e,l}$ do
        $h \leftarrow h_{i,l,e}$
        $\text{cell}_z \leftarrow \lfloor \frac{h_z - z_{\min}}{z_{\max} - z_{\min}} \rfloor$
        $\text{cell}_\phi \leftarrow \lfloor \frac{h_\phi - \phi_{\min}}{\phi_{\max} - \phi_{\min}} \rfloor$
        $\text{pos} \leftarrow \text{grid}_{\text{cell}_z, \text{cell}_\phi} + \text{atomicInc}(\text{written}_{\text{cell}_z, \text{cell}_\phi})$    ▷ if allocated: gridLocal
        $h'_{\text{pos},l,e} \leftarrow h$
    end for
end
```

1. In the first pass, the number of hits within each grid cell is counted. Each work-group processes the hits of one layer of a single event. Figure 8.5 illustrates the three-dimensional index space. The index space is of size $(\mathcal{T}, \mathcal{L}, \mathcal{E})$ with work-groups of size $(\mathcal{T}, 1, 1)$. The work-items increase a counter via an `atomicInc` for the according grid cell of a hit. The local memory of the compute device is used

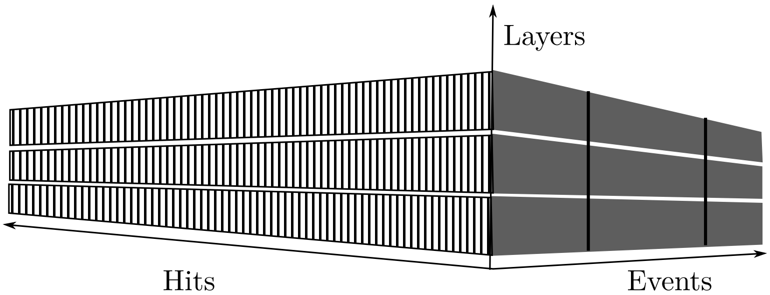


Figure 8.5.: Schematic of three-dimensional index space for $\mathcal{E} = 3$ events with $\mathcal{L} = 3$ detector layers each. The hits of a layer of an event are processed by $\mathcal{T}$ work-items.

to store the grid for the layer if possible, otherwise the data is stored in global memory. For coalesced access to the hit data in global memory, the main work loop is given by

```
for(uint i = globalID; i < nHits; i+= nThreads).
```

Therefore, all work-items of a work-group access contiguous memory segments. Subsequent to processing the assigned hits, the local counter is stored in the global counter if necessary.

2. The prefix sum of the global counter yields the grid cell bounds. Refer to Section 9.2 for the implementation of the prefix sum algorithm.
3. Before invoking the second pass, the host allocates a new hit array in global memory. Additionally, if the grid size exceeds the local memory size, a counter for the hits written to each grid cell is allocated in global memory. Otherwise local memory is preferred to store these values as they must be incremented atomically. Furthermore, if the local memory is large enough, it is used to cache the grid cell bounds for the detector layer of the work-group. For each hit, the appropriate grid cell is determined, the offset of that grid cell is obtained and the counter of written hits for that cell is increased atomically. The `atomicInc(x)` operation returns the value of `x` before the incrementation. Having determined the position of a hit in the new array, it can be written to that location.

The algorithm requires $\mathcal{O}(\mathcal{N} + \#_z \cdot \#_\phi)$ space and $\mathcal{O}(\mathcal{N})$ atomic operations, all of them in local memory if feasible. In the second pass, preference is given to the counter of written hits per grid cell for local memory storage. The data structure needs to be built once per event and henceforth is accessed many times in the subsequent steps of the triplet finding. The runtime performance of the construction algorithm and the impact of local memory utilization is evaluated in Section 10.3.

Chapter 9

Algorithms

Triplets are the foundation of the CA-based track reconstruction for CMS. Each triplet defines a cell in the automaton; neighboring cells, representing compatible triplets, can interact to form longer track candidates. Therefore, triplets must be found throughout all detector layers. This requires an efficient triplet finding algorithm, leveraging the inherent parallelism of the problem. As detailed in Section 3.3, OpenCL constitutes an open framework for high performance, parallel computing in heterogeneous environments. Its programming model closely resembles that of GPU-tailored frameworks, such as NVIDIA's CUDA, albeit being designed for multi-core CPU execution as well. Thus, many results about branch avoidance, memory hierarchy usage or work distribution derived on CUDA can be applied to OpenCL as well [48, 131].

This chapter introduces the algorithms employed for OpenCL-based triplet finding. Before describing the individual steps of the triplet finding process – pair building (Section 9.3), triplet prediction (Section 9.4) and triplet filtering (Section 9.5) – the concept of two-pass algorithms is detailed in Section 9.1. Section 9.2 explains the prefix sum operation, a common building block of all three triplet finding steps.

9.1. Two-pass Algorithms

As outlined in Algorithm 6.1, all three steps of the triplet finding process are designed as *two-pass* algorithms. Two-pass algorithms are a common approach for both CPU- as well as GPU-based algorithms [34, 45, 117, 152]. Facing an unknown, conceivably large, number of outputs, the lack of dynamic memory allocation within a kernel necessitates the use of two passes – a *counting* pass and a *storing* pass – with interwoven memory allocation from the host. In the counting pass, each input is examined for fulfillment of some specific property, with each work-item tracking the number of valid items. The global counter can be realized twofold:

- as a single number n_{valid}. Each work-item adds its locally determined number of valid inputs to n_{valid} in an atomic manner. Thus, assuming a total number of $\mathcal{T}$ work-items, $\mathcal{O}(\mathcal{T})$ atomic operations are required.
- as an array `counter` of length $\mathcal{T} + 1$. The locally determined number of valid inputs of work-item i is stored in $\texttt{counter}_i$. After concluding the counting pass,

Algorithm 9.1 Generic two-pass algorithm with oracle.

```
Input:  $\mathcal{N}$ inputs (on device), processed by $\mathcal{T}$ work-items
Output: a priori unknown number of outputs
allocate counter size $\mathcal{T}+1$ [uint] on device
$s \leftarrow 8 \cdot$ sizeof(uint)                                ▷ bits in uint
allocate oracle size $\lceil \frac{\mathcal{N}^2}{\frac{s}{8}} \rceil$ [uint] on device
kernel count begin
    $n_v \leftarrow 0$
    for $i =$ globalID $\xrightarrow{+\mathcal{T}} \mathcal{N}$ do
        [bool] v $\leftarrow$ check(input$_i$)
        oracle$_{\lfloor \frac{i}{s} \rfloor} \leftarrow$ oracle$_{\lfloor \frac{i}{s} \rfloor} \vee (\mathrm{v} \ll (i \mod s)) =:$ setOracle(i, v)
        $n_v \leftarrow n_v + v$
    end for
    counter$_{\texttt{globalID}} \leftarrow n_v$
end
kernel prefixSum(counter)
transfer counter$_{\mathcal{T}+1}$ to host
allocate output size counter$_{\mathcal{T}+1}$ on device
kernel store begin
    $p \leftarrow$ counter$_t$
    for $i =$ globalID $\xrightarrow{+\mathcal{T}} \mathcal{N}$ do
        [bool] v $\leftarrow$ oracle$_{\lfloor \frac{i}{s} \rfloor} \wedge (1 \ll (i \mod s)) =:$ readOracle(i)
        if $v =$ true then
            output$_p \leftarrow$ input$_i$
            $p \leftarrow p + 1$
        end if
    end for
end
```

the exclusive prefix sum of the count array is determined. Entry `counter`$_{\mathcal{T}+1}$ then contains the total number of valid inputs.

The latter approach avoids the atomic operations at the cost of $\mathcal{O}(\mathcal{T})$ extra space and a prefix sum operation. However, the prefix sum serves a further purpose in the store pass, as `counter`$_i$ determines the position of the first valid input of work-item i in the output array.

If the property under scrutiny is computationally expensive to verify, the result of the verification in the counting pass should be made available to the store pass. This can be achieved by the employment of an „oracle“ bit-string of the length of the maximum conceivable output [152]. For instance, given $\mathcal{N}$ hits in layer one and two each, the oracle is of length $\mathcal{N}^2$ Bit. Algorithm 9.1 gives the details of the use of the oracle bit-string in a two-pass algorithm. The number of processed inputs by each work-item $\frac{\mathcal{N}}{\mathcal{T}}$ should be aligned with the size of the data type used for the oracle to avoid atomic operations.

Assuming 32 bit `uint`s are employed, then

$$\frac{\mathcal{N}}{\mathcal{T}} \mod 32 \equiv 0$$

ensures, that no entry of the oracle array is accessed by two different work-items.

The generic two-pass Algorithm 9.1 is adapted by each step of the triplet finding process and modified to suit its particular requirements. Sections 9.3 through 9.5 discuss the details for each algorithm.

9.2. Prefix Sum Algorithm

One of the most essential building blocks for parallel algorithms is the prefix sum – or scan – operation [27]. A prefix sum operates on a monoid $\mathcal{M}$ with binary associative operation $\oplus : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ and identity element $\epsilon_{\oplus}$. Given an input sequence

$$I = \{i_0, i_1, \ldots, i_{n-1}\} \qquad \forall k \in [0, n-1] : i_k \in \mathcal{M},$$

the *exclusive* prefix sum yields

$$O = \{o_0, o_1, \ldots, o_{n-1}\} = \{\epsilon_{\oplus}, i_0, i_0 \oplus i_1, \ldots, \bigoplus_{k=0}^{n-2} i_k\},$$

whereas the *inclusive* prefix sum is given by

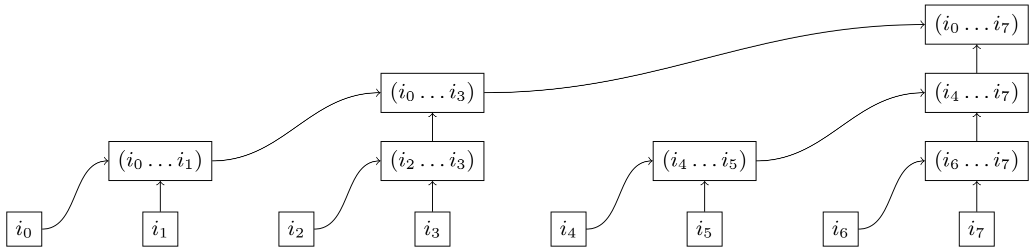
$$O' = \{o'_0, o'_1, \ldots, o'_{n-1}\} = \{i_0, i_0 \oplus i_1, \ldots, \bigoplus_{k=0}^{n-1} i_k\}.$$

The operation appears to be inherently sequential, since o_{n-1} depends on all inputs $i_0, \ldots, i_{n-2}$. A sequential implementations performs $\mathcal{O}(n)$ additions to produce output O. Hillis and Steele [84] present a parallel prefix sum algorithm computing O in $\mathcal{O}(\log n)$ steps with $\mathcal{O}(n \log n)$ additions. Blelloch [27] introduces a work-efficient algorithm requiring $\mathcal{O}(n)$ additions in $\mathcal{O}(\log n)$ steps. The algorithm is adapted to GPGPUs by Sengupta et al. [159] and presented in this section.

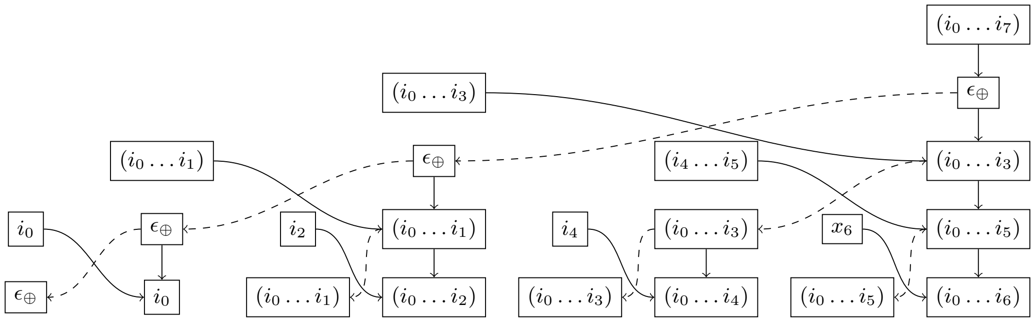
The prefix sum is computed using a balanced tree with n leafs and thus $d = \log n$ levels. The tree is traversed two times to built the prefix sum in place. The first traversal proceeds from the leafs of the tree towards the root, hence referred to as up-sweep, and computes

$$\forall l = d-1 \to 0 : o_x = \begin{cases} \bigoplus_{k=x-2^{d-l}+1}^{x} i_k & \text{if } (x+1) \equiv 0 \mod 2^{d-l} \\ i_x & \text{else} \end{cases}$$

for node $x \in [0, n-1]$. After $\log n$ steps, o_{n-1} equals $\bigoplus_{k=0}^{n-2}$, therefore the up-sweep performs a reduction of the inputs [122]. Figure 9.1a illustrates the up-sweep. Before


$(i_0 \dots i_7)$
$(i_0 \dots i_3)$
$(i_4 \dots i_7)$
$(i_0 \dots i_1)$
$(i_2 \dots i_3)$
$(i_4 \dots i_5)$
$(i_6 \dots i_7)$
i_0
i_1
i_2
i_3
i_4
i_5
i_6
i_7

(a) Up-sweep phase.


$(i_0 \dots i_7)$
$(i_0 \dots i_3)$
$\epsilon_\oplus$
$(i_0 \dots i_1)$
$\epsilon_\oplus$
$(i_4 \dots i_5)$
$(i_0 \dots i_3)$
i_0
$\epsilon_\oplus$
i_2
$(i_0 \dots i_1)$
i_4
$(i_0 \dots i_3)$
x_6
$(i_0 \dots i_5)$
$\epsilon_\oplus$
i_0
$(i_0 \dots i_1)$
$(i_0 \dots i_2)$
$(i_0 \dots i_3)$
$(i_0 \dots i_4)$
$(i_0 \dots i_5)$
$(i_0 \dots i_6)$

(b) Down-sweep phase.

Figure 9.1.: Schematic of the balanced trees employed in the two phase prefix sum algorithm by Blelloch [27]. The node entries $(i_a, \dots, i_b)$ denote $\bigoplus_{k=a}^{b} i_k$.

commencing the second traversal, the down-sweep, the last element o_{n-1} is set to identity $\epsilon_\oplus$ in order to calculate an exclusive prefix sum. The down-sweep computes

$$\forall l = 1 \rightarrow d : o_x = \begin{cases} o_x \oplus o_{x-2^{d-l}} & \text{if } (x+1) \equiv 0 \mod 2^{d-l+1} \\ o_{x+2^{d-l}} & \text{if } (x+1) \equiv 0 \mod 2^{d-l} \wedge (x+1) \not\equiv 0 \mod 2^{d-l+1} \\ o_x & \text{else} \end{cases}$$

for node $x \in [0, n-1]$, refer to Figure 9.1b.

Algorithm 9.2 presents pseudocode for the prefix sum kernel. Each work-item processes two inputs, thus the maximum length for the input array is bound by $2\mathcal{T}^*$ – $\mathcal{T}^*$ denoting the maximum work-group size – if only one work-group is employed. Processing larger arrays requires the use of multiple work-groups. However, work-items across multiple work-groups are not synchronizable, therefore the input is partitioned into chunks. Chunks are twice as large as the work-group size and can be processed independently. An array p of length $\lceil \frac{n}{\mathcal{T}} \rceil$ is used to store the partial sum of each work-group after the reduction phase, refer to Line 17. The prefix sum of the partial sum array p is computed in a recursive manner. Subsequently, each work-group $i \in [0, \lceil \frac{n}{\mathcal{T}} \rceil]$ uniformly adds p_i to its inputs. Algorithm 9.3 describes the details of the recursive prefix sum computation.

The shared memory of a GPU's Streaming Multiprocessor is organized in several memory banks. If multiple work-items access the same memory bank a *bank conflict* occurs [159], unless all work-items access the same word. Bank conflicts result in a serialization of the processing, as the memory accesses are served sequentially. In tree

Algorithm 9.2 Prefix sum kernel.

```
Input:  $I = \{i_0, \dots, i_{n-1}\}$
        array $p$ length $\lceil \frac{n}{\mathcal{T}} \rceil$ if $n >$ work-group size $\mathcal{T}$
Output: $\forall g \in [0, \lfloor \frac{n}{\mathcal{T}} \rfloor] :$
        $I = \{\dots, i_{2g\mathcal{T}} = \epsilon_{\oplus}, i_{2g\mathcal{T}+1} = i_{2g\mathcal{T}}, \dots, i_{4g\mathcal{T}-1} = \bigoplus_{k=0}^{n-2} i_{2g\mathcal{T}+k}, \dots\}$
        $p_g = \bigoplus_{k=0}^{n-2} i_{2g\mathcal{T}+k}$ if $n > \mathcal{T}$
$l \leftarrow \texttt{localID}_1$
$g \leftarrow \texttt{workGroupID}_1$
allocate local size $2\mathcal{T}$ + padding on local
offset $\leftarrow 2 \cdot g \cdot \mathcal{T}$
$\text{local}_{\text{localIdx}(l)} \leftarrow i_{\text{offset+l}}$
$\text{local}_{\text{localIdx}(l+\mathcal{T})} \leftarrow i_{\text{offset+l+}\mathcal{T}}$
$k \leftarrow 1$
for $d = \mathcal{T} \xrightarrow{/2} 0$ do
    if $l < d$ then
        $a \leftarrow k \cdot (2l+1) - 1$
        $b \leftarrow k \cdot (2l+2) - 1$
        $\text{local}_{\text{localIdx}(b)} = \text{local}_{\text{localIdx}(a)} + \text{local}_{\text{localIdx}(b)}$
    end if
    $k \leftarrow 2k$
end for
if $l = 0$ then
    $p_g \leftarrow \text{local}_{\text{localIdx}(2s-1)}$          ▷ only for multiple work-groups
    $\text{local}_{\text{localIdx}(2s-1)} \leftarrow 0$
end if
for $d = 1 \xrightarrow{\cdot 2} \mathcal{T}$ do
    $k \leftarrow \frac{k}{2}$
    if $l < d$ then
        $a \leftarrow k \cdot (2l+1) - 1$
        $b \leftarrow k \cdot (2l+2) - 1$
        $t \leftarrow \text{local}_{\text{localIdx}(a)}$
        $\text{local}_{\text{localIdx}(a)} = \text{local}_{\text{localIdx}(b)}$
        $\text{local}_{\text{localIdx}(b)} = t + \text{local}_{\text{localIdx}(b)}$
    end if
end for
$i_{\text{offset+l}} \leftarrow \text{local}_{\text{localIdx}(l)}$
$i_{\text{offset+l+}\mathcal{T}} \leftarrow \text{local}_{\text{localIdx}(l+\mathcal{T})}$
```

traversal algorithms, the memory access stride doubles at each level of the tree, resulting in twice as many bank conflicts. Bank conflicts can be mitigated by introducing a memory bank offset to shared memory accesses. Therefore, each shared memory address

Algorithm 9.3 Recursive prefix sum computation.

```
Input:  $I = \{i_0, \dots, i_{n-1}\}$
        $\mathcal{T}$ work-group size
        $\#_M$ number of memory banks per SMX
Output: $I = \{\epsilon_\oplus, i_0, i_0 \oplus i_1, \dots, \bigoplus_{k=0}^{n-2} i_k\}$
$\hat{n} \leftarrow \max\left(1, \lceil \frac{n}{2 \cdot \mathcal{T}} \rceil\right)$
padding $\leftarrow \frac{2 \cdot \mathcal{T}}{\#_M}$
if $\hat{n} = 1$ then
    $l \leftarrow \max\left(1, \frac{n}{2}\right)$
    $l \leftarrow 2^{\lceil \log l \rceil}$
    kernel prefixSum($I$)                    ▷ 1 work-group with $l$ work-items
else
    allocate partial size $\hat{n}$ on device
    kernel prefixSum($I$, partial)           ▷ $\hat{n}$ work-groups with $\mathcal{T}$ work-items each
    recursive prefix sum on partial
    kernel uniformAdd begin                  ▷ $\hat{n}$ work-groups with $\mathcal{T}$ work-items each
        $l \leftarrow \texttt{localID}_1$
        $g \leftarrow \texttt{workGroupID}_1$
        $x \leftarrow \text{partial}_g$
        offset $\leftarrow 2 \cdot g \cdot \mathcal{T}$
        $i_{\text{offset+l}} \leftarrow i_{\text{offset+l}} + x$
        $i_{\text{offset+l}+\mathcal{T}} \leftarrow i_{\text{offset+l}+\mathcal{T}} + x$
    end
end if
```

in Algorithm 9.2 is mapped to

$$\text{localIdx}(x) = x + \frac{x}{\#_M}$$

with $\#_M$ shared memory banks (ibid.). To accommodate the offset memory addresses, the amount of reserved local memory for each work-group needs to be increased by $\frac{2l}{\#_M}$, with l work-items per work-group.

Sengupta et al. [159] use NVIDIA's CUDA SDK to implement the introduced prefix sum algorithm. An OpenCL version is provided by Apple Inc. [13]. However, the implementation is overly complicated and suffers from a difficult-to-use interface. Hence, the algorithm is reimplemented, achieving a modest speedup over Apple's implementation, refer to Chapter 10.

9.3. Pair Building

Pairs of hits in adjacent detector layers are the basis for later triplet finding. However, two hits reveal limited information about a particle's track. Therefore, only very general filter criteria can be applied to limit the combinatorics and discard *background* hit pairs.

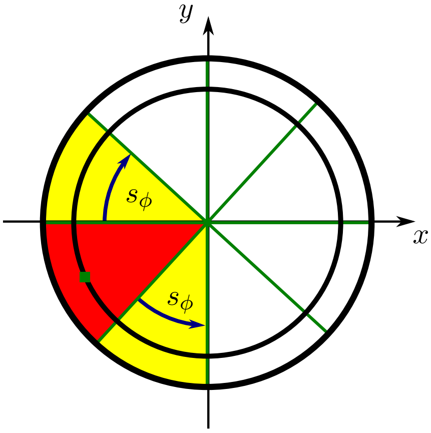


Figure 9.2.: Schematic of ϕ-slicing of the detector induced by the the grid data structure. A hit's slice and its s_ϕ neighboring ones are considered for pair building.

The grid-based pair building described in Section 9.3.1 imposes no origin constraint on the particle track and strives for the highest efficiency attainable. Later evaluation showed that an origin constraint is inevitable in order to contain the number of generated hit pairs within a feasible range. Therefore, a second approach employs more advanced predictive calculation to restrain the combinatorics, detailed in Section 9.3.2.

9.3.1. Grid-based Pair Building

Given a hit in the first layer, the search window in the second layer is defined by coarse proximity constraints. Proximity is determined by an asymmetric Moore neighborhood of radii (s_z, s_ϕ) grid cells. Figure 9.2 illustrates the neighborhood in s_ϕ. The parameters are obtained by the general geometric acceptance of the CMS tracker and evaluation of simulated $t\bar{t}$ events.

The inner tracker is designed for the tracking of charged particles with $|\eta| < 2.5$, thus θ is approximately in $\pm[0.05\pi, 0.95\pi]$ rad. The maximum longitudinal distance Δ_z traveled by a particle between layers of radius r_1 and r_2 can be limited to

$$|\Delta_z| \leq (r_2 - r_1) \cot(0.05\pi) \,.$$

In the transverse plane, triplet filtering and pair building use the same bounds on the change of ϕ of a valid track. Section 10.2 presents the conducted studies to attain Δ_ϕ.

Given the values for Δ_z and Δ_ϕ, the radii of the Moore neighborhood are chosen such that

$$\begin{aligned} |\Delta_z| &\leq s_z \frac{z_\text{max} - z_\text{min}}{\#_z} \\ |\Delta_\phi| &\leq s_\phi \frac{\phi_\text{max} - \phi_\text{min}}{\#_\phi}, \end{aligned} \tag{9.1}$$

for a grid partitioning (z, ϕ)-space ranging from $[z_\text{min}, z_\text{max}]$ and $[\phi_\text{min}, \phi_\text{max}]$ into $(\#_z, \#_\phi)$ cells.

Algorithm 9.4 Grid-based pair building *count* kernel.

```
Input:  $\mathcal{E}$ concurrent events, $\mathcal{L}$ detector layers, $\mathcal{N}_{e,l}$ hits in layer $l$ of event $e$
        $h_{i,l,e}$ event hits
        $[z_{\text{min}}, z_{\text{max}}]$ $[\phi_{\text{min}}, \phi_{\text{max}}]$ grid extent
        $\#_z$, $\#_\phi$ number of grid cells
        $(s_z, s_\phi)$ neighborhood size
Output: counter of found hit pairs
        oracle for *store* kernel
$(t, l, e) \leftarrow (\texttt{globalID}_1, \texttt{globalID}_2, \texttt{globalID}_3)$      ▷ (thread, layer, event)
$n_v \leftarrow 0$
for $i = t \xrightarrow{+\text{work-group size}} \mathcal{N}_{e,l}$ do
    $h^1 \leftarrow h_{i,l,e}$                                                  ▷ predict $z$-$\phi$-range
    $\text{minCell}_z \leftarrow \lfloor \frac{h^1_z - z_{\text{min}}}{z_{\text{max}} - z_{\text{min}}} \rfloor$
    $\text{maxCell}_z \leftarrow \min(\text{minCell}_z + s_z + 1, \#_z)$
    $\text{minCell}_z \leftarrow \max(0, \text{minCell}_z - s_z)$
    $\text{minCell}_\phi \leftarrow \lfloor \frac{h^1_\phi - \phi_{\text{min}}}{\phi_{\text{max}} - \phi_{\text{min}}} \rfloor - s_\phi$
    $\text{maxCell}_\phi \leftarrow \text{minCell}_\phi + 2s_\phi + 1$
    $[\texttt{bool}]\ \text{b} \leftarrow (\text{minCell}_\phi < 0) \vee (\text{maxCell}_\phi > \#_\phi)$      ▷ wraparound at $\pm\pi$
    $\text{minCell}_\phi \leftarrow \text{minCell}_\phi + (\text{minCell}_\phi < 0) \cdot \#_\phi$
    $\text{maxCell}_\phi \leftarrow \text{maxCell}_\phi - (\text{maxCell}_\phi > \#_\phi) \cdot \#_\phi$
    for $z \in [\text{minCell}_z, \text{maxCell}_z]$ do                          ▷ $z$-slice loop
        $j \leftarrow \text{grid}^{l+1,e}_{z,\text{minCell}_\phi}$
        $w \leftarrow \text{grid}^{l+1,e}_{z,\#_\phi+1} - \text{grid}_{z,0}$
        $e \leftarrow \text{grid}^{l+1,e}_{z,\text{maxCell}_\phi} + b \cdot w$
        for $j < e$ do                                                          ▷ second hit loop
            $h^2 \leftarrow h_{[j-(j \geq \text{grid}^{l+1,e}_{z,\#_\phi+1}) \cdot w],l+1,e}$
            $\text{setOracle}(i \cdot n_{l+1} + j - (j \geq \text{grid}^{l+1,e}_{z,\#_\phi+1}) \cdot w, 1)$
            increment $n_v$
        end for
    end for
end for
$\text{counter}_i \leftarrow n_v$
```

Algorithm 9.4 presents the details of the pair building *count* kernel. The algorithm determines the grid cell of a hit h in the first layer by

$$\text{cell}_z = \lfloor \frac{h_z - z_{\text{min}}}{z_{\text{max}} - z_{\text{min}}} \rfloor \qquad \text{cell}_\phi = \lfloor \frac{h_\phi - \phi_{\text{min}}}{\phi_{\text{max}} - \phi_{\text{min}}} \rfloor$$

and inspects the grid of the second layer in the cells within the neighborhood of cell $(\text{cell}_z, \text{cell}_\phi)$. Each hit within the search window in the second layer is counted and the corresponding bit in the oracle bit-string is set. The further processing follows the general two-pass algorithm scheme described by Algorithm 9.1. The *store* kernel iterates for each hit in the first layer over all hits in the second one and uses the information from the oracle to decide whether a given hit combination is accepted as hit pair. The set of all hit pairs (h^1, h^2) is stored in an array henceforth denoted by $\mathcal{H}^2$. Implementation details of Algorithm 9.4 are discussed in Section 9.3.3.

9.3.2. Prediction-based Pair Building

The computationally inexpensive grid-based pair building suffers from an tremendous amount of hit pairs when facing events with many tracks – refer to Section 10.2 for details. Thus, a more fine-grained prediction is needed to limit the number of generated hit pairs while still retaining high efficiency. Similar to CMSSW, the predictive pair building imposes constraints on the transverse and longitudinal impact parameter of a track – z_0 and d_0, respectively – and the maximum transverse curvature, corresponding to a minimum p_T.

In the transverse plane, the minimum p_T as well as the admissible transverse impact parameter d_0 constrain the Δ_ϕ between the first and the second hit of a pair. The minimum p_T defines the maximum transverse curvature of the particle's track in the magnetic field according to Equation (7.3)

$$\frac{1}{r_{\text{min}}} = \kappa_{\text{max}} = \frac{qB_z}{p_{T,\text{min}}}.$$

Therefore, Δ_ϕ can be limited to

$$|\Delta_\phi| \leq \left| \overbrace{\arccos\left(\frac{r_2}{2r_{\text{min}}}\right)}^{\alpha} - \overbrace{\arccos\left(\frac{r_1}{2r_{\text{min}}}\right)}^{\beta} \right| + \overbrace{\arctan\left(\frac{d_0(r_2 - r_1)}{r_1 r_2}\right)}^{\gamma}.$$

The angles α, β and γ refer to the pictorial representation of the formula in Figure 9.3.

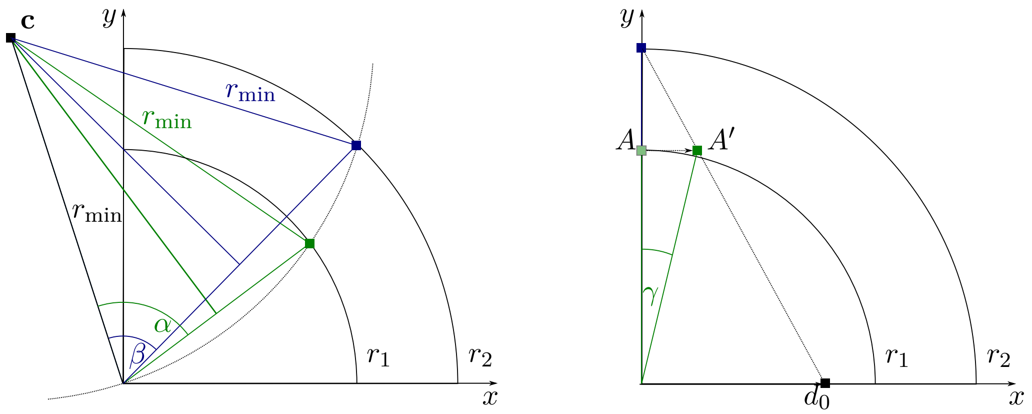
The Δ_ϕ constraint can be used for both inward and outward prediction. In order to limit the z-range for the second hit of a pair, a backward prediction is geometrically simpler. Given a hit in the *outer* layer,

$$\theta' = \arctan\frac{z_2 + z_0}{r_2} \qquad \theta'' = \arctan\frac{z_2 - z_0}{r_2}. \tag{9.2}$$

Figure 9.4 illustrates the angles θ' and θ''. For simplicity, d_0 is not accounted for in Equation (9.2). A compatible hit in the first layer must therefore be contained in the z-range

$$\left[r_1 \cot\theta' - z_0, r_1 \cot\theta'' + z_0\right].$$

Algorithm 9.5 displays the required computations to backward predict both z- and ϕ-range. The algorithm replaces Line 4 through Line 12 marked in green in Algorithm 9.4. Furthermore, layer l is accessed in the z-slice and second hit loop instead of layer $l+1$ – marked in blue – as hits in the first layer are predicted based upon a hit in the second.



(a) $\Delta_\phi \geq |\alpha - \beta|$ due to $r_{\min} \propto p_{T,\min}$.

(b) $\Delta_\phi \geq \gamma$ due to TIP d_0, assuming a high p_T track and small γ. Therefore, A' is approximated by (x, r_1) and the intercept theorem can be applied.

Figure 9.3.: Schematics for calculating the maximum $\Delta_\phi = |\alpha - \beta| + \gamma$ for pair finding.

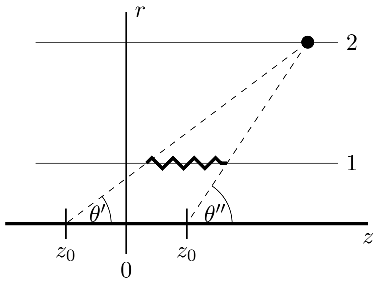


Figure 9.4.: Calculating the feasible z-range for the inner hit of a hit pair. The maximum longitudinal impact parameter z_0 limits the width of the search window.

Due to the more accurate prediction of the z-ϕ-range, not all hits within the grid cell span need to be accepted as hit pairs. In a *tight* prediction mode, only hits actually within the predicted range are selected. Therefore the update of the oracle and incrementation of n_v (Lines 18 and 19) are conditional on

$$h^1_\phi \in [\mathrm{pred}_{\phi,\min}, \mathrm{pred}_{\phi,\max}] \wedge h^1_z \in [\mathrm{pred}_{z,\min}, \mathrm{pred}_{z,\max}].$$

The *store* kernel for the prediction-based pair building follows the design of the general *store* kernel described in Algorithm 9.1 and used in the grid-based pair building as well, adapted to the interchanged roles of outer and inner layer. The kernel produces the hit pair collection $\mathcal{H}^2$.

Algorithm 9.5 Prediction of z-ϕ-range in pair building *count* kernel.

```
Input:  $r_{l,\text{min}}/r_{l,\text{max}}$ minimum/maximum radius of inner layer
        $(z_0, d_0)$ longitudinal and transverse impact parameter
        $r_{\text{min}}$ minimum radius of the trajectory's curvature
Output: $(\text{minCell}_z, \text{maxCell}_z)$, $(\text{minCell}_\phi, \text{maxCell}_\phi)$ prediction window
 1: $h^1 \leftarrow h_{i,l+1,e}$
 2: $\cot\theta' \leftarrow \frac{h_z^2 + z_0}{h_r^2}$
 3: $\cot\theta'' \leftarrow \frac{h_z^2 - z_0}{h_r^2}$
 4: $\text{pred}_{z,\text{min}} \leftarrow \min(r_{l,\text{min}} \cdot \cot\theta' - z_0, r_{l,\text{max}} \cdot \cot\theta' - z_0)$
 5: $\text{pred}_{z,\text{max}} \leftarrow \max(r_{l,\text{min}} \cdot \cot\theta'' + z_0, r_{l,\text{max}} \cdot \cot\theta'' + z_0)$
 6: $\text{minCell}_z \leftarrow \max\left(0, \lfloor \frac{\text{pred}_{z,\text{min}} - z_{\text{min}}}{z_{\text{max}} - z_{\text{min}}} \rfloor\right)$
 7: $\text{maxCell}_z \leftarrow \min\left(\lfloor \frac{\text{pred}_{z,\text{max}} - z_{\text{min}}}{z_{\text{max}} - z_{\text{min}}} \rfloor, \#_z\right)$
 8: $d\phi \leftarrow \max\left(\left|\arccos \frac{h_r^2}{2r_{\text{min}}} - \arccos \frac{r_{l,\text{min}}}{2r_{\text{min}}}\right|, \left|\arccos \frac{h_r^2}{2r_{\text{min}}} - \arccos \frac{r_{l,\text{max}}}{2r_{\text{min}}}\right|\right)$
 9: $d\phi \leftarrow d\phi + \max\left(\arctan d_0 \frac{h_r^2 - r_{l,\text{min}}}{h_r^2 \cdot r_{l,\text{min}}}, \arctan d_0 \frac{h_r^2 - r_{l,\text{max}}}{h_r^2 \cdot r_{l,\text{max}}}\right)$
10: $\text{pred}_{\phi,\text{min}} \leftarrow h_\phi^2 - d\phi$
11: $\text{pred}_{\phi,\text{max}} \leftarrow h_\phi^2 + d\phi$
12: [bool] $b \leftarrow \text{pred}_{\phi,\text{min}} \notin [-\pi, \pi] \vee \text{pred}_{\phi,\text{max}} \notin [-\pi, \pi]$        ▷ wraparound at ±π
13: $\text{pred}_{\phi,\text{min}} \leftarrow \text{pred}_{\phi,\text{min}} + 2\pi \cdot (\text{pred}_{\phi,\text{min}} < -\pi) - 2\pi \cdot (\text{pred}_{\phi,\text{min}} > \pi)$
14: $\text{pred}_{\phi,\text{max}} \leftarrow \text{pred}_{\phi,\text{max}} + 2\pi \cdot (\text{pred}_{\phi,\text{max}} < -\pi) - 2\pi \cdot (\text{pred}_{\phi,\text{max}} > \pi)$
15: $\text{minCell}_\phi \leftarrow \lfloor \frac{\text{pred}_{\phi,\text{min}} - \phi_{\text{min}}}{\phi_{\text{max}} - \phi_{\text{min}}} \rfloor$
16: $\text{maxCell}_\phi \leftarrow \lfloor \frac{\text{pred}_{\phi,\text{max}} - \phi_{\text{min}}}{\phi_{\text{max}} - \phi_{\text{min}}} \rfloor$
```

9.3.3. Implementation Details

The hit pairs identified by the *count* kernel are stored in an oracle bit-string of length

$$4 \sum_{\substack{e<\mathcal{E} \\ l<\mathcal{L}-2}} \lceil \frac{\mathcal{N}_{e,l} \cdot \mathcal{N}_{e,l+1}}{32} \rceil \,\text{Byte}.$$

Furthermore, an offset into the oracle bit-string is provided for each work-group, requiring another $\mathcal{E} \cdot \frac{\mathcal{L}-2}{3}$ Byte. For each event and triplet of layers, $\mathcal{T}$ partial sums are stored by the work-items. Thus, the array for the prefix sum operation comprises $4\frac{\mathcal{L}-2}{3}\mathcal{T} + 1\,\text{Byte}$.

Algorithm 9.4 exhibits several subtleties concerning the determination of the appropriate grid cell span and the indexing of hits in the hit array. The boundary conditions for the z-span are enforced via min and max operations, refer to Line 5 onward. To reduce the register usage of the kernel, the minCell_z register serves as both temporary variable and final minimum grid cell in z. Starting Line 8, the ϕ-span is determined. The procedure needs to address the wraparound at $\pm\pi$ as exemplified in Figure 9.2. If a wraparound occurs, it is ensured that minCell_ϕ points to the appropriate grid cells

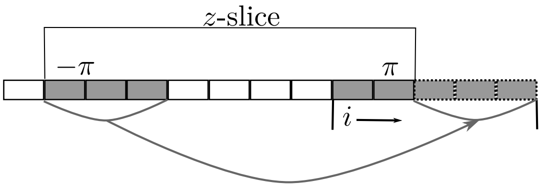


Figure 9.5.: Illustration of ϕ-wraparound. The selected grid cells (shaded in gray) from the third quadrant are mapped behind the grid cells of the second quadrant.

Operation	**Grid-based**	**Prediction-based**
division/modulo	$2 + 2n_{\text{recall}}$	$12 + 2n_{\text{recall}}$
square root	0	1
trigonometric	1	$5 + n_{\text{recall}}{}^{\ddagger}$
global memory	$3 + 4n^{\dagger}_{\text{cells},z}$	$9 + 4n^{\dagger}_{\text{cells},z} + 4n_{\text{recall}}{}^{\ddagger}$
atomic	n_{recall}	n_{recall}

Table 9.1.: Overview of expensive operations performed by the pair building *count* kernels *per* first hit. The number of grid cells within the z-range is denoted by $n_{\text{cells},z}$; n_{recall} marks the number of hits within the predicted range.
† : only if grid does *not* fit in local memory. ‡ : only if tight prediction is used.

in second quadrant of the unit circle, $[\frac{\pi}{2}, \pi]$, and maxCell$_{\phi}$ to the ones in the third quadrant, $[-\pi, -\frac{\pi}{2}]$. Predicated multiplications are used to avoid branching. As depicted in Figure 9.5, if a wraparound in ϕ occurs, the hits of the third quadrant are mapped behind the ones of the second quadrant by index manipulation, refer to Algorithm 9.4 Line 14 onward.

In order to accelerate the access to the hit positions (12 Byte per position) in the inner loop, a `prefetch` instruction for each z-slice is issued in Line 14. Prefetching the entire z-slice avoids complications due to possible ϕ-wraparounds. Considering the 1 536 kByte L2 cache of NVIDIA's Kepler architecture [138] and the 256 kByte L2 cache of Intel's Core i7 [91], ample amounts of cache are available to justify this simplification.

If the local memory is sufficiently large, the grid data structure for the second layer (grid-based pair building) or for the first layer (prediction-based pair building) is cached before entering the main loop. This reduces the required global memory accesses significantly. Refer to Table 9.1 for an overview of expensive operations performed by both variants of pair building. It becomes apparent that the reduced number of generated hit pairs by the prediction-based approach comes at the cost of many more compute intense operations. In Section 10.2, the reduction of hit pairs is quantified and the merit of the more expensive calculations examined.

The kernel is executed in the same three-dimensional index space as employed for grid building, refer to Figure 8.5. Each work-group processes the hits of one detector layer – for the first hit of the pair – of a single event.

Algorithm 9.6 Angle-based prediction of the z-range.

```
Input:  $(h^1, h^2)$ hit pair, $l$ third layer
        $p_\theta$ prediction window size
        $r_{l,\min}$ / $r_{l,\max}$ minimum/maximum radius of outer layer
Output: $[\mathrm{pred}_{z,\min}, \mathrm{pred}_{z,\max}]$ admissible $z$-range
$s \leftarrow \operatorname{sgn}\left(h_y^2\right)$
$\theta \leftarrow \arctan\left(s\frac{\sqrt{(h_x^2-h_x^1)^2+(h_y^2-h_y^1)^2}}{h_z^2-h_z^1}\right) =: \boldsymbol{\theta}(\mathbf{h^1}, \mathbf{h^2})$
$\theta_{\min} \leftarrow (1 - p_\theta) \cdot \theta$
[char] $w \leftarrow 1 - 2 \cdot (|\theta_{\min}| > \pi)$                ▷ $-1$ indicates wraparound
$\theta_{\min} \leftarrow \theta_{\min} - 2\pi \cdot (\theta_{\min} > \pi)$
$\theta_{\min} \leftarrow \theta_{\min} + 2\pi \cdot (\theta_{\min} < -\pi)$
$\theta_{\min} \leftarrow \theta_{\min} \cdot w$                            ▷ $\theta$-wraparound changes sign
$\theta_{\max} \leftarrow (1 + p_\theta) \cdot \theta$
...                                                     ▷ same overflow treatment
$r \leftarrow s \cdot \sqrt{(h_x^2 - h_x^1)^2 + (h_y^2 - h_y^1)^2}$
$\Delta_{r,\min} \leftarrow s \cdot r_{l,\min} - r$
$\Delta_{r,\max} \leftarrow s \cdot r_{l,\max} - r$
$\mathrm{pred}_{z,\min} \leftarrow \min(h_z^2 + \Delta_{r,\min} \cot\theta_{\min}, h_z^2 + \Delta_{r,\min} \cot\theta_{\max})$     ▷ see Algorithm 9.7
$\mathrm{pred}_{z,\max} \leftarrow \max(h_z^2 + \Delta_{r,\min} \cot\theta_{\min}, h_z^2 + \Delta_{r,\min} \cot\theta_{\max})$
$\mathrm{pred}_{z,\min} \leftarrow \min\left(\mathrm{pred}_{z,\min}, h_z^2 + \Delta_{r,\max} \cot\theta_{\min}, h_z^2 + \Delta_{r,\max} \cot\theta_{\max}\right)$
$\mathrm{pred}_{z,\max} \leftarrow \max\left(\mathrm{pred}_{z,\max}, h_z^2 + \Delta_{r,\max} \cot\theta_{\min}, h_z^2 + \Delta_{r,\max} \cot\theta_{\max}\right)$
```

9.4. Triplet Prediction

Restricting the number of generated triplet candidates, given a set of hit pairs, is of paramount importance to cope with the combinatoric challenge posed by high pile-up events. By extrapolating the track „defined“ by a hit pair to the third layer, the feasible z- and ϕ range can be limited. In one approach, the prediction is based on the filter criteria derived in Section 7.1. The prediction is based on the two hits of a pair in the lower layers and two *speculative* hits in the third layer – one representing the minimum acceptable value and one the maximum acceptable value with respect to the filter criteria. Alternatively, the prediction can be based on the extrapolation of the track subject to a minimum p_T and transverse as well as longitudinal impact parameter.

Sections 9.4.1 and 9.4.2 elaborate on the extrapolation methods. Their use within the triplet prediction kernels is detailed in Section 9.4.3.

9.4.1. Angular-based Prediction

In this approach, the prediction of the admissible ϕ and z ranges is based upon the maximum $d\phi$ and $d\theta$ given by the filter criteria introduced in Section 7.1.

Algorithm 9.7 Branch-less min/max determination.

```
Input:  $h^2$ hit in second layer
        $\Delta_{r,\text{min}}/\Delta_{r,\text{max}}$ minimum/maximum traversed radial distance
        $\cot\theta_{\text{min}}/\cot\theta_{\text{max}}$ cotangents of minimum/maximum polar angle
Output: $(\text{pred}_{z,\text{min}}, \text{pred}_{z,\text{max}})$ feasible $z$-range
$\text{tmp} \leftarrow h_z^2 + \Delta_{r,\text{max}} \cot\theta_{\text{min}}$    ▷ treatment of $\Delta_{r,\text{max}}$
$\text{pred}_{z,\text{max}} \leftarrow h_z^2 + \Delta_{r,\text{max}} \cot\theta_{\text{max}}$
$\text{pred}_{z,\text{min}} \leftarrow (\text{tmp} < \text{pred}_{z,\text{max}}) \cdot \text{tmp} + (\text{tmp} > \text{pred}_{z,\text{max}}) \cdot \text{pred}_{z,\text{max}}$
$\text{pred}_{z,\text{max}} \leftarrow (\text{tmp} < \text{pred}_{z,\text{max}}) \cdot \text{pred}_{z,\text{max}} + (\text{tmp} > \text{pred}_{z,\text{max}}) \cdot \text{tmp}$
$\text{tmp} \leftarrow h_z^2 + \Delta_{r,\text{min}} \cot\theta_{\text{min}}$    ▷ treatment of $\Delta_{r,\text{min}}$
$\text{pred}_{z,\text{min}} \leftarrow (\text{tmp} < \text{pred}_{z,\text{min}}) \cdot \text{tmp} + (\text{tmp} > \text{pred}_{z,\text{min}}) \cdot \text{pred}_{z,\text{min}}$
$\text{pred}_{z,\text{max}} \leftarrow (\text{tmp} > \text{pred}_{z,\text{max}}) \cdot \text{tmp} + (\text{tmp} < \text{pred}_{z,\text{max}}) \cdot \text{pred}_{z,\text{max}}$
$\text{tmp} \leftarrow h_z^2 + \Delta_{r,\text{min}} \cot\theta_{\text{max}}$
$\text{pred}_{z,\text{min}} \leftarrow (\text{tmp} < \text{pred}_{z,\text{min}}) \cdot \text{tmp} + (\text{tmp} > \text{pred}_{z,\text{min}}) \cdot \text{pred}_{z,\text{min}}$
$\text{pred}_{z,\text{max}} \leftarrow (\text{tmp} > \text{pred}_{z,\text{max}}) \cdot \text{tmp} + (\text{tmp} < \text{pred}_{z,\text{max}}) \cdot \text{pred}_{z,\text{max}}$
```

Algorithm 9.8 Angle-based prediction of the ϕ-range.

```
Input:  $(h^1, h^2)$ hit pair
        $p_\phi$ prediction window size
Output: $[\text{pred}_{\phi,\text{min}}, \text{pred}_{\phi,\text{max}}]$ admissible $\phi$-range
        $b$ indicates $\phi$-wraparound
$\phi \leftarrow \arctan\left(\frac{h_y^2 - h_y^1}{h_x^2 - h_x^1}\right) =: \boldsymbol{\phi}(\mathbf{h^1}, \mathbf{h^2})$
$\text{pred}_{\phi,\text{min}} \leftarrow \min((1 - p_\phi) \cdot \phi, (1 + p_\phi) \cdot \phi)$
$\text{pred}_{\phi,\text{max}} \leftarrow \max((1 - p_\phi) \cdot \phi, (1 + p_\phi) \cdot \phi)$
[bool] $b \leftarrow \text{pred}_{\phi,\text{min}} \notin [-\pi, \pi] \vee \text{pred}_{\phi,\text{max}} \notin [-\pi, \pi]$
$\text{pred}_{\phi,\text{min}} \leftarrow \text{pred}_{\phi,\text{min}} + 2\pi \cdot (\text{pred}_{\phi,\text{min}} < -\pi) - 2\pi \cdot (\text{pred}_{\phi,\text{min}} > \pi)$
$\text{pred}_{\phi,\text{max}} \leftarrow \text{pred}_{\phi,\text{max}} + 2\pi \cdot (\text{pred}_{\phi,\text{max}} < -\pi) - 2\pi \cdot (\text{pred}_{\phi,\text{max}} > \pi)$
```

The prediction of the feasible z-range in the third layer for a hit pair (h^1, h^2) is presented in Algorithm 9.6. The hit pair defines the polar angle θ of the line segment between both points,

$$\theta = \arctan\left(\frac{\text{sgn}\left(h_y^2\right) \cdot \sqrt{(h_x^2 - h_x^1)^2 + (h_y^2 - h_y^1)^2}}{h_z^2 - h_z^1}\right).$$

A signed radius is used to indicate the detector half above (positive) and below (negative) the beam pipe. Assuming a straight line trajectory over the signed radial distance Δ_r between the second and third detector layer, the particle's z-coordinate in the third layer is given by

$$h_z^3 = h_z^2 + \Delta_r \cot\theta.$$

Algorithm 9.9 Extrapolation-based prediction the z-range.

```
Input:   $(h^1, h^2)$ hit pair, $l$ third layer
         tolerance for $z$-prediction $\sigma_z$
         $r_{l,\min}/r_{l,\max}$ minimum/maximum radius of outer layer
Output:  $[\text{pred}_{z,\min}, \text{pred}_{z,\max}]$ admissible $z$-range
$d_0^2 \leftarrow \frac{\left(h_y^1 \cdot (h_x^2 - h_x^1) - h_x^1 \cdot (h_y^2 - h_y^1)\right)^2}{(h_x^2 - h_x^1)^2 \cdot (h_y^2 - h_y^1)^2}$
$r_o \leftarrow \sqrt{(h_x^1)^2 + (h_y^1)^2 - d_0^2}$
$\cot\theta \leftarrow \frac{h_z^2 - h_z^1}{\sqrt{(h_x^2)^2 + (h_y^2)^2 - d_0^2}}$
$\text{pred}_{z,\min} \leftarrow \min\left(h_z^1 + \left(r_{l,\min}^2 - d_0^2\right) \cdot \cot\theta, h_z^1 + \left(r_{l,\max}^2 - d_0^2\right) \cdot \cot\theta\right) - \sigma_z$
$\text{pred}_{z,\max} \leftarrow \max\left(h_z^1 + \left(r_{l,\min}^2 - d_0^2\right) \cdot \cot\theta, h_z^1 + \left(r_{l,\max}^2 - d_0^2\right) \cdot \cot\theta\right) + \sigma_z$
```

As the magnetic field of the CMS solenoid is not ideal, particle tracks might be bent in the longitudinal plane. Furthermore, the resolution in z varies between pixel and silicon strip detector, with the latter featuring a significantly lower resolution than the former. Thus, the θ of the line segment between h^2 and the predicted h^3 is allowed to span the range $[\theta_{\min}, \theta_{\max}]$,

$$\theta_{\min} = (1 - p_\theta) \cdot \theta \qquad \theta_{\max} = (1 + p_\theta) \cdot \theta,$$

with p_θ specifying the size of the prediction window, $p_\theta > d\theta$ as defined by Equation (7.4). If either $|\theta_{\min}|$ or $|\theta_{\max}|$ is greater than π, a wraparound at $\pm\pi$ occurred. The angles are normalized to the interval $[-\pi, \pi]$ by predicated additions to avoid branching. Furthermore, the wraparound entails a transition from the upper to the lower half of the detector, thus requiring changing the sign of the radius accordingly. Since $|\theta_{\min}|$ and $|\theta_{\max}|$ could point to different halves, the radius's sign may differ between the two. Separate variables for the respective sign can be omitted by employing the identity $\cot(-x) = -x$ and encoding the radius's sign into the angle variable, refer to Line 7.

Detector modules of one layer are mounted at differing radii, therefore the minimum and maximum radius of the third layer are both used for z-range prediction. The minimum/maximum finding – lines marked in green in Algorithm 9.6, starting Line 13 – is implemented in a branch-less manner as detailed in Algorithm 9.7. In addition to branch avoidance, the algorithm also highlights the reuse of registers to minimize private memory consumption.

Algorithm 9.8 describes the prediction of the feasible ϕ-range. The calculations closely follow the ones for the θ-range, with the same branch-less wraparound treatment. The ϕ-prediction window size p_ϕ is chosen such that $p_\phi > d\phi$.

9.4.2. Extrapolation-based Prediction

In order to more accurately predict the path of a charged particle through the detector, this approach employs calculations similar to the ones used in CMSSW.

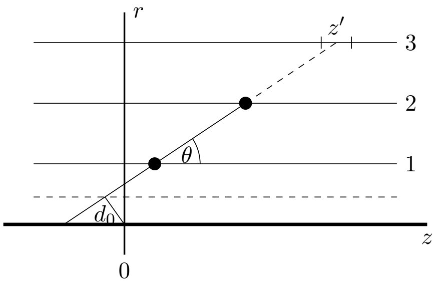


Figure 9.6.: Prediction of the feasible z-range of the third hit based upon a straight line extrapolation. The transverse impact parameter d_0 is determined by the distance of the line to the origin.

The z-range of the hit in the third layer is extrapolated based upon a straight line approximation of the particle's track in the longitudinal plane. As Figure 9.6 illustrates, the extrapolation takes into account the effects of the transverse impact parameter d_0, defined by the distance of the track to the origin. The distance of a function $f(x)$ to the origin can be determined by minimizing

$$d(x) = \sqrt{x^2 + f(x)^2}.$$

For a straight line

$$f(x) = mx + n,$$

the quadratic distance is given by

$$d_0^2 = \frac{n^2}{m^2 + 1}. \tag{9.3}$$

Given the hit pair (h^1, h^2), Equation (9.3) yields

$$d_0^2 \leftarrow \frac{\left(h_y^1 \cdot (h_x^2 - h_x^1) - h_x^1 \cdot (h_y^2 - h_y^1)\right)^2}{(h_x^2 - h_x^1)^2 \cdot (h_y^2 - h_y^1)^2}.$$

Algorithm 9.9 presents the details of the computation. The size of the prediction window is governed by parameter σ_z, which accounts for measurement uncertainties, multiple scattering etc. The minimum and maximum (Line 4) is determined similarly to Algorithm 9.7 in a branch-less manner.

The ϕ-range is predicted in a similar manner as in prediction-based pair building. The minimum p_T defines the minimum radius of the curvature r_{min} – refer to Equation (7.3) – that bounds the maximum change in ϕ between two hits of the same track

$$|\Delta_\phi| \leq \left| \arccos\left(\frac{d}{2r_{\text{min}}}\right) - \arccos\left(\frac{r_3}{2r_{\text{min}}}\right) \right|,$$

with r_3 denoting the radius of the third layer and d the radial distance between h^1 and h^2. The transverse impact parameter does not increase the Δ_ϕ, since the effect is already accounted for by radial distance d. As Algorithm 9.10 shows, a wraparound of ϕ is treated in the same manner as in Algorithm 9.8.

Algorithm 9.10 Extrapolation-based prediction of the ϕ-range.

```
Input:  $(h^1, h^2)$ hit pair, $l$ third layer
        $r_{l,\text{min}}/r_{l,\text{max}}$ minimum/maximum radius of outer layer
        $r_{\text{min}}$ minimum radius of the trajectory's curvature
Output: $[\text{pred}_{\phi,\text{min}}, \text{pred}_{\phi,\text{max}}]$ admissible $\phi$-range
        $b$ indicates $\phi$-wraparounds
$d\phi \leftarrow \left| h^2_\phi - h^1_\phi \right|$
$d \leftarrow \sqrt{(h^2_x - h^1_x)^2 + \left(h^2_y - h^1_y\right)^2}$
$d\phi' \leftarrow \max\left( \left| \arccos \frac{d}{2r_{\text{min}}} - \arccos \frac{r_{l,\text{min}}}{2r_{\text{min}}} \right|, \left| \arccos \frac{d}{2r_{\text{min}}} - \arccos \frac{r_{l,\text{max}}}{2r_{\text{min}}} \right| \right)$
$d\phi \leftarrow \max(d\phi, d\phi')$
$\text{pred}_{\phi,\text{min}} \leftarrow h^2_\phi - d\phi$
$\text{pred}_{\phi,\text{max}} \leftarrow h^2_\phi + d\phi$
[bool] $b \leftarrow \text{pred}_{\phi,\text{min}} \notin [-\pi, \pi] \vee \text{pred}_{\phi,\text{max}} \notin [-\pi, \pi]$        ▷ wraparound at $\pm\pi$
$\text{pred}_{\phi,\text{min}} \leftarrow \text{pred}_{\phi,\text{min}} + 2\pi \cdot (\text{pred}_{\phi,\text{min}} < -\pi) - 2\pi \cdot (\text{pred}_{\phi,\text{min}} > \pi)$
$\text{pred}_{\phi,\text{max}} \leftarrow \text{pred}_{\phi,\text{max}} + 2\pi \cdot (\text{pred}_{\phi,\text{max}} < -\pi) - 2\pi \cdot (\text{pred}_{\phi,\text{max}} > \pi)$
```

Operation	**Angular-based**	**Extrapolation-based**
division/modulo	4	12
square root	2	10
trigonometric	2	$4 + n_{\text{recall}}$†
global memory	$20 + 4n_{\text{cells},z}$	$16 + 4n_{\text{cells},z} + 4n_{\text{recall}}$†
atomic	0	0

Table 9.2.: Overview of expensive operations performed by the triplet prediction kernels *per* hit pair. The number of grid cells within the z-range is denoted by $n_{\text{cells},z}$; n_{recall} marks the number of hits within the predicted range.
† : only if tight prediction is used.

9.4.3. Implementation Details

The *count* and *store* kernel employed for triplet prediction is given by Algorithm 9.11, which is executed in a one-dimensional index space. Each work-item processes one hit pair, resulting in an index space of size $\|\mathcal{H}^2\|$. Consequently, the count array is of size $(4\,\|H^2\| + 1)$ Byte). No oracle bit-string is used to tie the *count* and *store* kernel together. The size of the oracle would quickly reach tremendous magnitudes of several hundred MByte per event. Moreover, looping over all hits in the third layer for every hit pair in the *store* kernel is prohibitive for large events, thus, the predictive calculations need to be repeated inevitably. Therefore, both kernels differ only slightly as highlighted in Algorithm 9.11. In addition to incrementing the counter of found triplet candidates n_v, the *store* kernel writes the triplet candidate to its appropriate position in the triplet candidate collection $\mathcal{H}^C$.

Algorithm 9.11 Triplet prediction *count*/*store* kernel.

```
Input:   $h_{i,l,e}$ event hits
         $\mathcal{H}^2$ hit pair collection
         $[z_{\text{min}}, z_{\text{max}}]$ $[\phi_{\text{min}}, \phi_{\text{max}}]$ grid extent
         $\#_z$, $\#_\phi$ number of grid cells
         prefixSum of found triplet candidates                         ▷ store kernel
Output:  counter of found triplet candidates                           ▷ count kernel
         $\mathcal{H}^C$ triplet candidate collection                  ▷ store kernel
 1: $i \leftarrow$ globalID$_1$                                        ▷ thread
 2: $(h^1, h^2) \leftarrow \mathcal{H}^2_i$
 3: $(l, e) \leftarrow (\text{layer}, \text{event})$ of $h^1$
 4: $n_v \leftarrow 0$      $n_v \leftarrow \text{prefixSum}_i$            ▷ count / store respectively
 5: $(\text{pred}_{z,\text{min}}, \text{pred}_{z,\text{max}}) \leftarrow \text{predict}_z(h^1, h^2, l+2)$
 6: $\text{minCell}_z \leftarrow \max(\lfloor \frac{\text{pred}_{z,\text{min}} - z_{\text{min}}}{z_{\text{max}} - z_{\text{min}}} \rfloor, 0)$
 7: $\text{maxCell}_z \leftarrow \min(\lfloor \frac{\text{pred}_{z,\text{max}} - z_{\text{min}}}{z_{\text{max}} - z_{\text{min}}} \rfloor, \#_z)$
 8: $(\text{pred}_{\phi,\text{min}}, \text{pred}_{\phi,\text{max}}, b) \leftarrow \text{predict}_\phi(h^1, h^2)$   ▷ [bool] $b$ indicates wraparound
 9: $\text{minCell}_\phi \leftarrow \lfloor \frac{\text{pred}_{\phi,\text{min}} - \phi_{\text{min}}}{\phi_{\text{max}} - \phi_{\text{min}}} \rfloor$
10: $\text{maxCell}_\phi \leftarrow \lfloor \frac{\text{pred}_{\phi,\text{max}} - \phi_{\text{min}}}{\phi_{\text{max}} - \phi_{\text{min}}} \rfloor$
11: for $z \in [\text{minCell}_z, \text{maxCell}_z]$ do
12:     $j \leftarrow \text{grid}^{l+2,e}_{z,\text{minCell}_\phi}$
13:     $w \leftarrow \text{grid}^{l+2,e}_{z,\#_\phi+1} - \text{grid}_{z,0}$
14:     $k \leftarrow \text{grid}^{l+2,e}_{z,\text{maxCell}_\phi} + b \cdot w$
15:     for $j < k$ do
16:         $h^3 \leftarrow h_{[j-(j \geq \text{grid}^{l+2,e}_{z,\#_\phi+1}) \cdot w],l+2,e}$
17:         if $h^3_\phi \in [\text{pred}_{\phi,\text{min}}, \text{pred}_{\phi,\text{max}}] \wedge h^3_z \in [\text{pred}_{z,\text{min}}, \text{pred}_{z,\text{max}}]$ then ▷ tight mode
18:             $\mathcal{H}^C_{n_v} \leftarrow ((h^1, h^2), h^3)$              ▷ store kernel
19:             increment $n_v$
20:         end if
21:     end for
22: end for
23: $\text{counter}_i \leftarrow n_v$
```

The inner loop of the kernels resembles the one of the pair building kernel (Algorithm 9.4), employing the same index manipulation to avoid branching due to ϕ wraparounds – refer to Figure 9.5.

The more precise prediction of the extrapolation-based triplet prediction allows for a *tight* third hit candidate selection, accepting only hits within in the predicted range as opposed to all hits of the selected grid cells, refer to Line 17. This limits the number of produced triplet candidates, however requires an arctan operation per hit in the selected grid cells. Table 9.2 gives an overview of the performed compute-intense operations by the triplet prediction kernels. Similar to pair building, the more precise prediction requires more expensive operations,

A `prefetch` instruction is issued in Line 14 for the entire z-slice, following the same rational as in Section 9.3.3.

9.5. Triplet Filtering

Given the hit triplet candidates produced by the (coarse) extrapolation of the previous step, a more fine-grained analysis is required to separate valid from fake triplets. The criteria employed to distinguish triplets belonging to a particle's track from background ones are discussed in Chapter 7. Algorithm 9.12 shows the details of the performed computations. First, the change in θ and ϕ is calculated and verified against the maximum admissible values $d\theta$ and $d\phi$, respectively. From Line 7 onward, the transverse impact parameter is calculated following the deliberations of Section 7.2. Standard OpenCL operations are used for all vector operations to exploit native implementations of the underlying hardware.

The index space is one-dimensional and of size $\left\|\mathcal{H}^C\right\|$, i. e. each work-item processes a single triplet candidate. In order to count the number of valid triplets, a counter of length $4\left(\left\|\mathcal{H}^C\right\| + 1\right)$ Byte would be needed. To limit the amount of memory used by

Algorithm 9.12 Triplet filtering *count* kernel.

```
Input:  $\mathcal{H}^C$ triplet candidate collection
        $(d\theta, d\phi, d_0)$ filter criteria
Output: oracle as counter and for store kernel
 1: $i \leftarrow$ globalID$_1$                                              ▷ thread
 2: $(h^1, h^2, h^3) \leftarrow \mathcal{H}^C_i$
 3: bool $v \leftarrow 1 - d\theta \leq \frac{\theta(h^2,h^3)}{\theta(h^1,h^2)} \leq 1 + d\theta$
 4: $\Delta_\phi \leftarrow \phi(h^2, h^3) - \phi(h^1, h^2)$
 5: $\Delta_\phi \leftarrow \Delta_\phi - (\Delta_\phi > \pi) \cdot 2\pi + (\Delta_\phi < -\pi) \cdot 2\pi$
 6: $v \leftarrow v \cdot (|\Delta_\phi| \leq d\phi)$
 7: $\forall i \in \{1, 2, 3\} : \hat{h}^i \leftarrow \left(h^i_x, h^i_y, (h^i_x)^2 + (h^i_y)^2\right)$
 8: float3 $\mathbf{n} \leftarrow \frac{\overrightarrow{\hat{h}^1\hat{h}^2} \times \overrightarrow{\hat{h}^1\hat{h}^3}}{\left|\overrightarrow{\hat{h}^1\hat{h}^2} \times \overrightarrow{\hat{h}^1\hat{h}^3}\right|}$
 9: float2 $\mathbf{c} \leftarrow \left(-\frac{\mathbf{n}_x}{2\mathbf{n}_z}, -\frac{\mathbf{n}_y}{2\mathbf{n}_z}\right)$                 ▷ center of circle
10: float $r \leftarrow \sqrt{\frac{1-\mathbf{n}_z^2+4\mathbf{n}_z(\mathbf{n}\cdot\hat{h}_1)}{4\mathbf{n}_z^2}}$                 ▷ radius of circle
11: float2 $\mathbf{d} \leftarrow \mathbf{c} - r \cdot \frac{\mathbf{c}}{|\mathbf{c}|}$         ▷ point of closest approach to beam line
12: $v \leftarrow v \cdot (|\mathbf{d}| \leq d_0)$
13: setOracle$(i, v)$
```

Algorithm 9.13 Hamming weight calculation [109].

Input: oracle bit-string
Output: counter of valid triplets

```
$i \leftarrow \texttt{globalID}_1$                                        ▷ thread
$v \leftarrow \text{oracle}_i$
$v \leftarrow v - ((i \gg 1) \wedge \text{0x55555555})$
$v \leftarrow (v \wedge \text{0x33333333}) + ((v \gg 2) \wedge \text{0x33333333})$
$\text{counter}_i \leftarrow (((v + (v \gg 4)) \wedge \text{0x0F0F0F0F}) \cdot \text{0x01010101}) \gg 24$
```

Operation	Triplet Filtering
division/modulo	7
square root	5
trigonometric	4
global memory	13
atomic	1

Table 9.3.: Overview of expensive operations performed by the triplet filtering kernel *per* triplet candidate.

the kernel, the oracle bit-string of size $4\left\lceil \frac{\|\mathcal{H}^C\|}{32} \right\rceil$ Byte serves two purposes: it stores the outcome of the validation of a triplet candidate to be accessible by the *store* kernel and it is used as basis for the prefix sum. For the latter, for each **`uint`** of the oracle bit-string, the *Hamming weight* is calculated [75]. The Hamming weight – also referred to as population count or *popcount* [109] – denotes the number of set bits (1 bits) within a bit-string. It is computed by a separate kernel subsequently to the *count* kernel. OpenCL 1.2 specifies popcount as standard operation [102]. Since NVIDIA only supports OpenCL 1.1 in its current drivers, the Hamming weight is computed following Algorithm 9.13. Prior to kernel execution, a counter array of size $4\left(\left\lceil \frac{\|\mathcal{H}^C\|}{32} \right\rceil + 1\right)$ Byte is allocated. The one-dimensional index space of the *popcount* kernel is of size $\left\lceil \frac{\|\mathcal{H}^C\|}{32} \right\rceil$.

The *store* kernel follows the principal scheme of Algorithm 9.1. Each work-item processes one **`uint`** of the oracle bit-string, therefore requiring only a single global memory access to obtain the oracle information for 32 triplet candidates..

Table 9.3 summarizes the expensive operations performed by the triplet filtering kernel.

Chapter 10

Evaluation

The evaluation of heuristic algorithms needs to address two aspects [145]: the quality of the algorithm's results and the time required to produce it. In the case of particle track reconstruction, the quality of the produced results is described by the *efficiency* (Equation 2.3), *fake rate* (Equation 2.4) and *clone rate* (Equation 2.5). The runtime behavior is characterized by the *kernel* time – the time spent by the compute device executing the kernel – and the *wall* time – including the kernel time, data transfers, scheduling overhead etc.

Before scrutinizing the physical and algorithmic properties of the presented OpenCL-based triplet finding in Sections 10.2 and 10.3, respectively, the setup for the evaluation is presented in Section 10.1.

10.1. Evaluation Setup

This section presents the simulated event types employed to assess the quality of the reconstructed triplets and the hardware and software configurations used for measuring the runtime behavior of the algorithm.

10.1.1. Simulated Events

The physical performance of the algorithm is studied with Quantum Chromodynamics (QCD) and $t\bar{t}$ events. Both are generated with Pythia 6 [168], version 4.26, with a center-of-mass energy of $\sqrt{s} = 14\,\mathrm{TeV}$. QCD processes are the predominant interactions observed in the CMS detector and therefore constitute a suitable benchmark to measure the performance of the triplet finding in average events. The transverse momentum of the interacting quarks is set to be in the range of $80\,\mathrm{GeV}\,c^{-1}$ to $120\,\mathrm{GeV}\,c^{-1}$, resulting in on average 110 of tracks per event. Collisions producing a top and antitop quark feature a complex topology, including many jets, and thus allow the evaluation of the performance of the algorithm in more intricate events. The produced sample contains on average 144 tracks per event. For both types of processes, 2 000 events are generated. As particles of high energy and transverse momentum are the primary target of physical analyses, particles below $1\,\mathrm{GeV}\,c^{-1}$ p_T are excluded from the set of to be found tracks. Furthermore, only tracks comprising a hit in all considered barrel layers are deemed

	OpenCL term	**CPU**	**GPU**
vendor		Intel	NVIDIA
model		Core i7-3930K	GTX 660
clock speed		3.20 GHz	1.03 GHz
cores	compute units	$6 \cdot 2$	5
L1 cache	local memory	32 kByte	48 kByte
L2 cache		256 kByte	1 536 kByte
L3 cache		12 MByte	–
main memory	global memory	16 GByte	3 GByte
memory bandwidth		51.2 GByte s^{-1}	144.2 GByte s^{-1}
throughput		2.83 GByte s^{-1}	32.9 GByte s^{-1}

Table 10.1.: CPU and GPU used for runtime evaluations. The Core i7 features Intel's „Hyper-Threading“ technology, offering two logical cores to the operating system per physical one. The throughput is determined experimentally by evaluating a simple addition kernel.

findable, discarding about 5 % of the generated tracks.

In order to study the runtime behavior of the algorithm, fine-grained control of the number of tracks per event is necessary. Thus, a „random particle gun“ – simulating the path of a individual particle through the detector – is used to generate a specified number of particle tracks. The produced samples features 1 to 4 096 muon tracks, originating at the transverse origin $(0, 0)$, with a p_T uniformly distributed in $\mathcal{U}[1, 10]$ GeV c^{-1} and η in $\mathcal{U}[-1, 1]$.

10.1.2. Hardware and Software Configuration

The algorithm is evaluated on an Intel Core i7 3930K processor and a NVIDIA GTX 660 graphics card. The details of the hardware specifications are listed in Table 10.1. The graphics card is paired with an AMD Athlon II X2-220 with 4 GByte of main memory in a machine running Ubuntu Linux 12.04 with version 319.23 of NVIDIA's Linux driver, supporting OpenCL 1.1. The NVIDIA driver also provides the OpenCL compiler for the kernels; for the host application, GCC version 4.7.3 is used. The Core i7 CPU is installed in a machine operating under Scientific Linux CERN (SLC) 6.4 with Intel's OpenCL SDK 2012 for OpenCL 1.1, providing OpenCL compiler version 1.0.2. GCC version 4.7.2 is used to compile the host application. All runtime measurements are performed on an otherwise unoccupied system and repeated ten times.

In order to gain an understanding of the relative performance of both devices, the runtime for calculating the prefix sum of 2^{20} `uint`s is determined for varying work-group sizes in $[1, 1024]$. Figure 10.1 shows the obtained speedup measurements. The relative speedup for a device is given with respect to the runtime of the prefix sum algorithm on the same device with work-group size 1,

$$S_{\text{rel}}(n) = \frac{T_x(1)}{T_x(n)} \qquad x \in \{\text{CPU}, \text{GPU}.\}$$

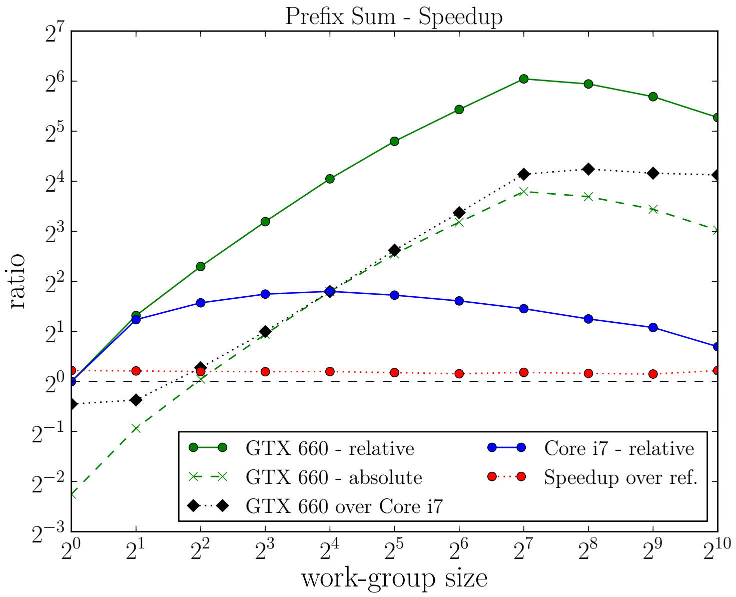


Figure 10.1.: Study of the relative performance of CPU and GPU used in the runtime evaluation. Relative speedup is with respect to the runtime with a work-group of size 1 on the same device type. The absolute speedup for the GPU is with respect to the minimum runtime of the CPU. The ratio refers to the speedup of the GPU over the CPU for the same work-group size. The reference implementation is provided by Apple Inc. [13]

The absolute speedup for the GPU is calculated with respect to the minimum runtime of the algorithm on the CPU,

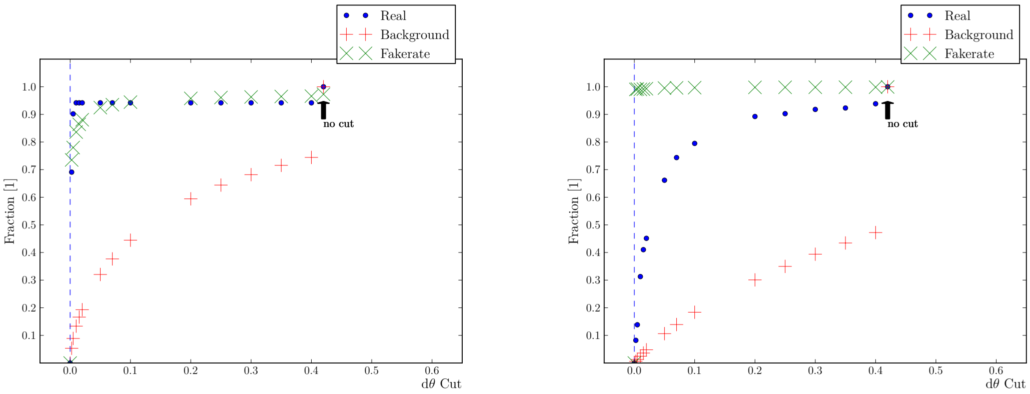
$$S_{\text{rel}}(n) = \frac{\min_{x\in[1,1024]} T_{\text{CPU}}(x)}{T_{\text{GPU}}(n)}.$$

The ratio relates the runtime on CPU over GPU for the same work-group size.

As Figure 10.1 reveals, the GPU is very sensitive to the choice of work-group size. Setting the work-group size to 128 work-items yields a speedup of factor 64 over work-group size 1. The CPU is more robust to the parameter, showing a maximum gain of factor 3 for work-groups of size 16. The maximum speedup of the GPU over the CPU is a factor 16.

Comparing the achieved peak throughputs for the kernel – $0.66\,\text{GByte}\,\text{s}^{-1}$ on the CPU and $9.14\,\text{GByte}\,\text{s}^{-1}$ on the GPU – with the experimentally determined maximum throughputs (refer to Table 10.1) indicates a compute-bound prefix sum implementation. The maximum throughputs are determined by a simple kernel – reading one value from global memory, adding a constant and writing back the result to global memory – in 1 000 iterations.

The physical and algorithmic performance of CMSSW is studied for version 6.0.0, incorporating the k-d tree data structure by Reid [146].



(a) Triplets in the pixel layers 1-2-3.

(b) Triplets in the pixel and silicon strip layers 3-4-5.

Figure 10.2.: Study of the longitudinal bending of particle tracks for the $\left|\frac{\theta'}{\theta} - 1\right| \leq d\theta$ filter criterion. The percentage of *real* and *background* triplets passing the filter criterion for varying cutoff values as well as the resulting fake rate is given for the pixel layers and a combination of pixel and silicon strip layers.

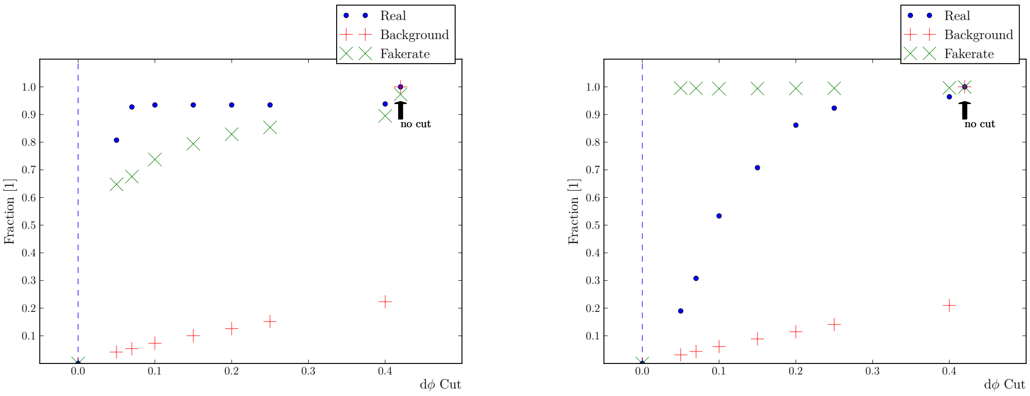
10.2. Physics Performance

The attained quality of the produced results depends on the cutoff values used for the filter criteria introduced in Chapter 7. Looser cutoffs result in a higher efficiency but also entail more fakes passing the filter. Furthermore, a larger number of produced outputs incurs longer processing times for subsequent steps of triplet finding and track reconstruction. Therefore, the chosen cutoff values must carefully balance the need for efficiency with the desire for low fake rates and fast processing times. The following Section 10.2.1 elaborates on the determination of the cutoff values for the $d\theta$, $d\phi$ and d_0 filter. Subsequently, the resulting physical performance for QCD and $t\bar{t}$ events as well as for the muon samples is presented in Sections 10.2.3 to 10.2.4.

10.2.1. Determination of Cutoff Values for Filter Criteria

In order to determine suitable cutoff values for the $d\phi$, $d\theta$ and d_0 filter criteria presented in Chapter 7, $t\bar{t}$ events are studied due to their complex topology. For each filter criterion, the fraction of *real* – i. e. valid – and *background* – i. e. fake – triplets passing the criterion for various cutoff values is scrutinized. The criteria are examined for triplets of hits in layers

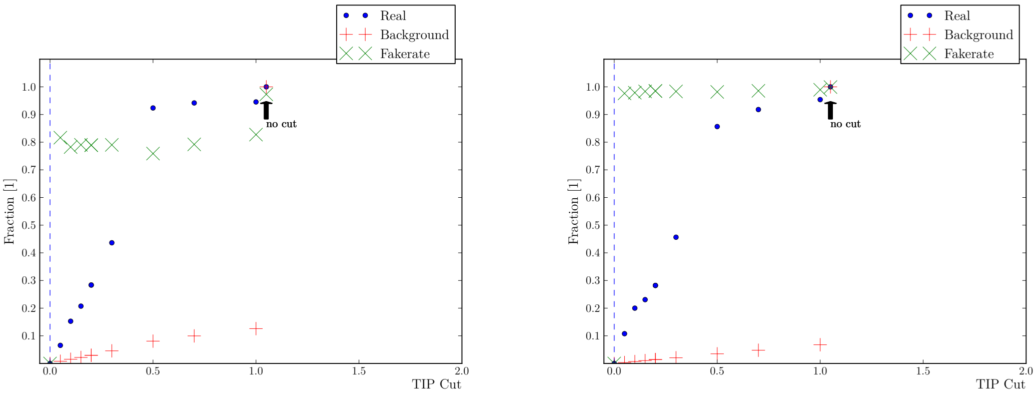
- 1-2-3, therefore using only pixel layers in the PXB;
- 2-3-4, using the first stereo layer of the silicon strip TIB; and
- 3-4-5, employing both stereo layers of the TIB.



(a) Triplets in the pixel layers 1-2-3.

(b) Triplets in the pixel and silicon strip layers 3-4-5.

Figure 10.3.: Study of the transverse bending of particle tracks for the $|\phi' - \phi| \leq d\phi$ filter criterion. The percentage of *real* and *background* triplets passing the filter criterion for varying cutoff values as well as the resulting fake rate is given for the pixel layers and a combination of pixel and silicon strip layers.

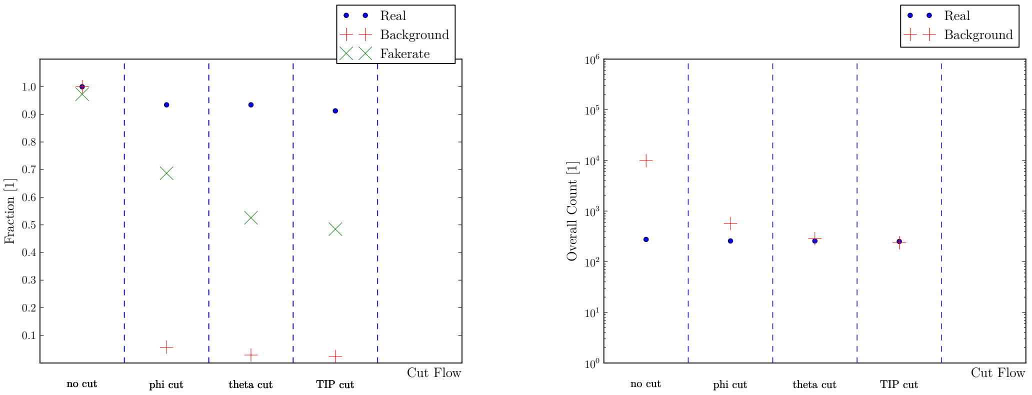


(a) Triplets in the pixel layers 1-2-3.

(b) Triplets in the pixel and silicon strip layers 3-4-5.

Figure 10.4.: Study of the transverse impact parameters of particle tracks for the d_0 filter criterion. The percentage of *real* and *background* triplets passing the filter criterion for varying cutoff values as well as the resulting fake rate is given for the pixel layers and a combination of pixel and silicon strip layers.

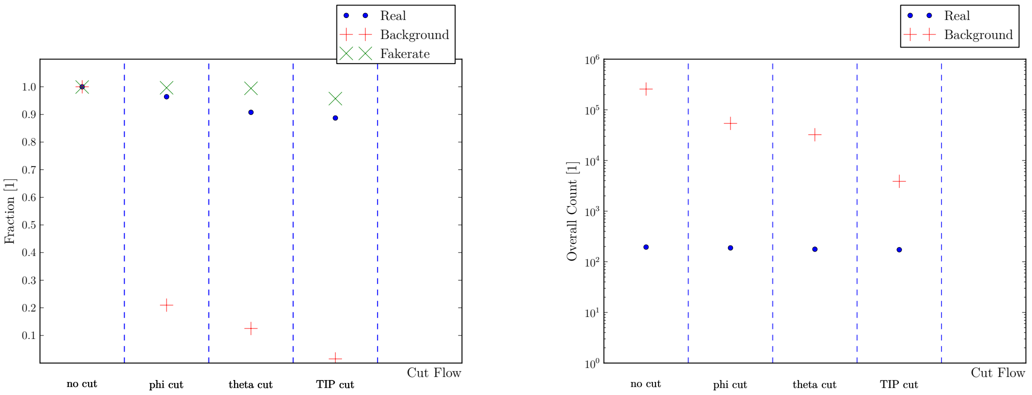
Figure 10.2 shows the results for the $d\theta$ filter. As discussed in Section 7.1, a straight line approximates the path of a charged particle in the longitudinal plane. Given the high precision measurements of the pixel detector, a small tolerance on $\frac{\theta'}{\theta}$ suffices to accept most valid triplets, refer to Figure 10.2a. Background triplets can be successfully rejected using small enough $d\theta$ values, while still obtaining high efficiency. On the other hand, hits in the stereo silicon strip layers suffer from a more coarse-grained resolution



(a) Percentage of *real* and *background* triplets passing the consecutive application of the $d\theta$, $d\phi$ and d_0 filter. Furthermore, the resulting fake rate is given.

(b) Number of *real* and *background* triplets passing the consecutive filter step.

Figure 10.5.: Consecutive application of the proposed filter criteria for the chosen cutoff value for each filter. The plots show the data for the pixel layers 1-2-3.



(a) Percentage of *real* and *background* triplets passing the consecutive application of the $d\theta$, $d\phi$ and d_0 filter. Furthermore, the resulting fake rate is given.

(b) Number of *real* and *background* triplets passing the consecutive filter step.

Figure 10.6.: Consecutive application of the proposed filter criteria for the chosen cutoff value for each filter. The plots show the data for combination of pixel and silicon strip layers 3-4-5.

in z of 230 µm, as opposed to 20 µm in the PXB. Hence, the tolerance on $d\theta$ needs to be an order of magnitude larger to achieve similar efficiency as in the pixel detector as shown by Figure 10.2b. The background rejection for a given $d\theta$ value in the 3-4-5 layer combination is even stronger than in the PXB; still, the immense number of fake combinations – refer to Figure 10.6b – results in a very high fake rate.

Layer Combination	**dϕ**	**dθ**	**d$_0$**
1-2-3	0.075	0.01	0.5
2-3-4	0.2	0.2	0.5
3-4-5	0.4	0.4	1.0
4-5-8	0.4	0.5	3.5

Table 10.2.: Cutoff values for the $d\phi$, $d\theta$ and d_0 filter criteria.

The results of the $d\phi$ filter studies are presented in Figure 10.3. A trajectory's transverse curvature depends on the particle's transverse momentum p_T, refer to Equation (7.3). Thus, a certain change in ϕ is expected between hits of the track in adjacent detector layers. The CMS detector features a high resolution in ϕ in both the PXB and the TIB, with 10 µm and 23 µm to 35 µm, respectively. Nevertheless, Figures 10.3a and 10.3b show that a much larger tolerance on $d\phi$ is required in the 3-4-5 layer combination than in the pixel layers to attain similar efficiency. This is due to the greater radial distance traveled by the particle between the layers of the TIB than between the layers of the PXB. The pixel layers are mounted at radii ≈ 3 cm apart, whereas the silicon strip layers are separated by ≈ 15 cm. Between the third PXB layer and the first TIB layer, the particle traverses ≈ 15 cm of radial distance. For a detailed description of the detector layers' positions refer to Table 1.1. The background rejection is of similar order for both layer configurations; for the fake rate, the same observations as for the $d\theta$ criterion apply.

The transverse impact parameter d_0 is determined by the Riemann fit method described in Section 7.2. As illustrated by Figure 10.4, the filter achieves similar efficiency in both layer combinations for the same cutoff value, as the varying radial distances between the layers are accounted for by the fitted circle parameters. Particularly in the 3-4-5 layer configuration, the d_0 criterion is very efficient in rejecting background triplets.

The cutoff values chosen for the further analysis of the OpenCL-based triplet finding algorithm are listed in Table 10.2. They are chosen to attain an efficiency of about 90 % in order to limit the background triplets. Tracks not within the scope of this parameter configuration can be found by a second iteration of the algorithm. As hits already covered by a reconstructed track are masked off in later iterations, less restrictive cutoff values can be applied. The cutoff values for the outermost layer combination – 4-5-8, involving the first layer of the TOB detector – are derived from the values of the 3-4-5 configuration as it is introduced in later stages of the studies.

Whereas the previous plots addressed the filter criteria in isolation from one another, Figures 10.5 and 10.6 show the effects of the consecutive application of the $d\phi$, $d\theta$ and $d\phi$ filters with the chosen cutoff values. While accepting approximately the same *percentage* of valid and fake triplets – $\approx 90\,\%$ and $< 0.1\,\%$, respectively – in both layer configurations, the *number* of passing fakes is an order of magnitude higher in the TIB than in the PXB, explaining the high fake rate.

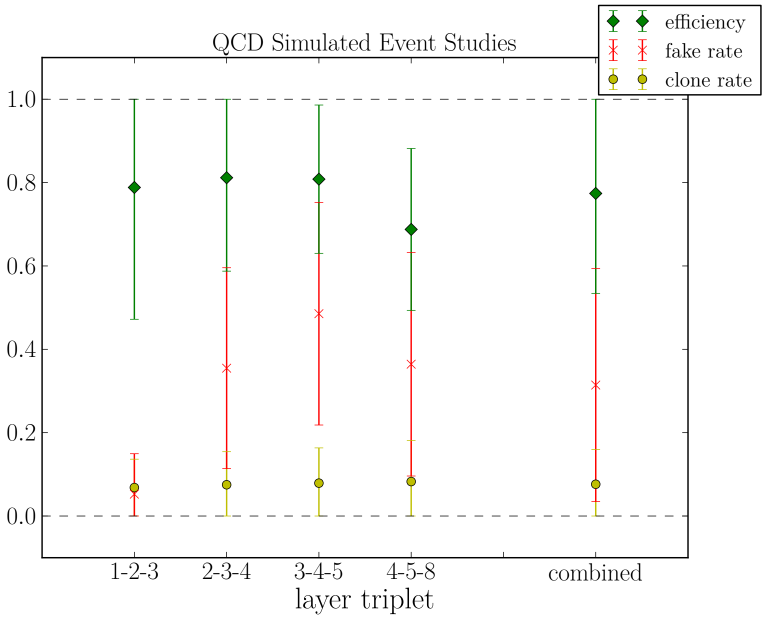
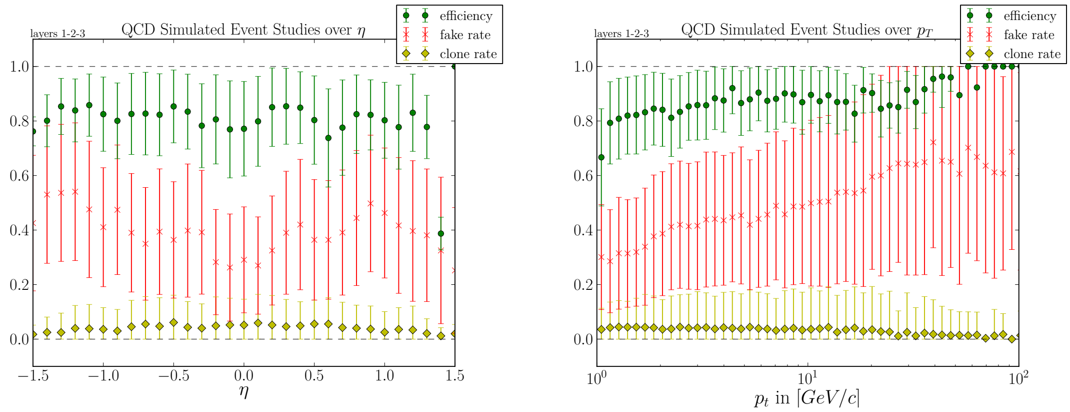


Figure 10.7.: Overview of physics performance for all considered layer combinations for QCD events.



(a) Physics performance measures over η.

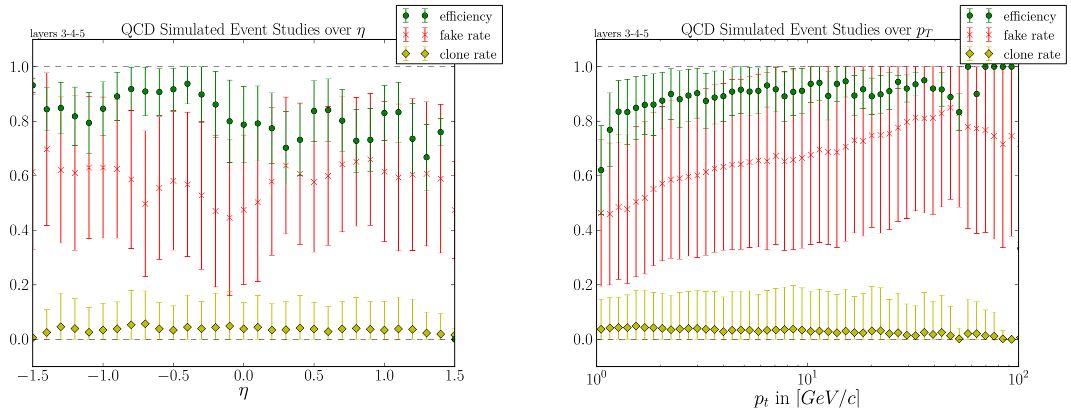
(b) Physics performance measures over p_T.

Figure 10.8.: Efficiency, fake rate and clone rate for QCD events in the pixel layers 1-2-3.

10.2.2. Physics Performance for QCD Events

QCD processes are the most observed interactions in proton-proton collisions. Hence, they provide good insight into the performance of the triplet finding for the average case.

Figure 10.7 summarizes the quality of the results produced by the OpenCL-based triplet finding algorithm. The performance is shown for prediction-based pair building and extrapolation-based triple prediction, both with tight selection. The rationale of this choice is discussed in Section 10.2.4. For all PXB and TIB layers, the efficiency is about



(a) Physics performance measures over η. (b) Physics performance measures over p_T.

Figure 10.9.: Efficiency, fake rate and clone rate for QCD events in the combined pixel and silicon strip layers 3-4-5.

80 %. For the last layer configuration, including layer eight in the outer silicon strip detector (TOB), the efficiency drops to 70 %, due to the large radial distance traversed by the particle between layer five and eight ($\approx$ 20 cm) and the lower resolution in ϕ (40 µm) as well as z (530 µm). The fake rate increases from $\approx$ 10 % in the pixel layers to 40 % to 50 % in the silicon strip layers because of the looser cutoff values required to maintain a reasonable efficiency. Clones are mainly due to overlapping detector modules within one layer, refer to Figure 8.1b. If both modules record a particle's traversal, two triplets will be generated from the reconstructed hits; one using the inner module's hit and another including the outer hit. The clone rate fluctuates around 7 % throughout all studied layer combinations.

The efficiency remains constant throughout the η range of the PXB, as illustrated by Figure 10.8a. Since the implementation of the presented algorithm is restricted to the barrel region, the drop in efficiency for the outer $\eta = 1.5$ bin is expected as the transition of particles from the barrel into the endcap region needs to be considered to find tracks in the higher η-range. The fake rate increases with growing η due to larger predicted z-range in pair building, refer to Figure 9.4. Studying the performance measures over p_T in Figure 10.8b reveals a constant efficiency, with a slight drop for low transverse momentum tracks. The greater change in ϕ between layers for low p_T particles due to the CMS solenoid exceeds the thresholds imposed by the $d\phi$ filter. The same behavior can be observed in Figure 10.9 for layer configurations involving TIB layers, with increased fake rate because of the less restrictive filter cutoffs.

Even though the filter cutoff values are derived from the study of $t\bar{t}$ events, they show good performance for QCD processes as well. Thus, the approach of using the more complex event topology to determine the cutoff values is validated.

Detailed plots for the physics performance of the presented algorithm in the other layer configurations are presented in Appendix A.

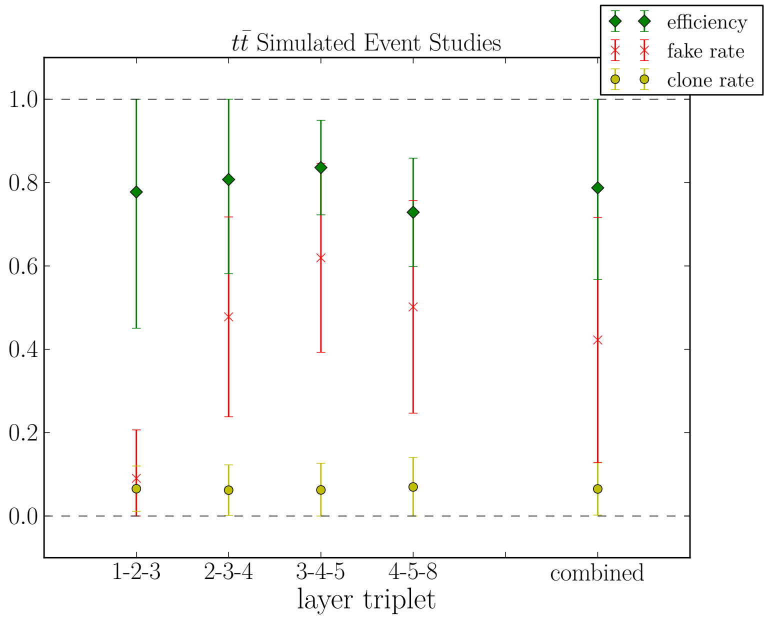


Figure 10.10.: Overview of physics performance for all considered layer combinations for $t\bar{t}$ events.

10.2.3. Physics Performance for $t\bar{t}$ Events

The complex topology of the top-antitop production provides a good benchmark for the performance of the triplet finding for more challenging events.

The efficiency in $t\bar{t}$ events – as shown by Figure 10.10 – is comparable to the one in QCD events, with $\approx 80\,\%$ for the PXB and TIB layer configurations and a moderate decrease when including the TOB. The fake rate increases by about $10\,\%$ in the silicon strip detectors compared to QCD events. The increase can be attributed to the parton jets present in $t\bar{t}$ events. The jets produce many hits within a narrow cone, therefore allowing the creation of many invalid triplets that are still passing the filter criteria due to their close proximity.

The performance measures exhibit a similar behavior over η and p_T as described in the previous section and confirm the gained insights. Detailed plots are displayed in Appendix A.

10.2.4. Physics Performance for Muon Events

The muon samples are specifically produced for runtime evaluation purposes. Their topology – with up to 4 096 muon tracks originating at the nominal transverse beamspot – does *not* resemble any physical processes of proton-proton interactions. Nevertheless, the artificial sample is well suited for runtime studies, providing a similar workload as anticipated for the data-taking period starting 2015, with 1 000 to 2 000 tracks per event. Therefore, the algorithm's physical performance for the sample needs to be verified in order for meaningful runtime measurements.

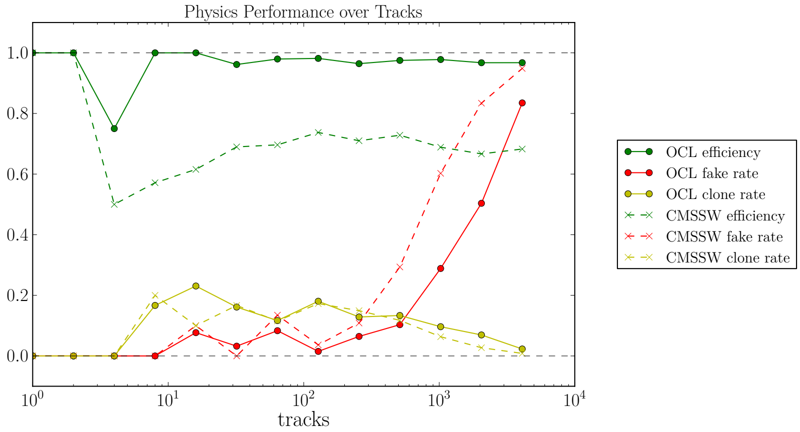


Figure 10.11.: Physics performance for muon events over number of simulated tracks. Solid lines represent the performance of the OpenCL-based triplet finding. Dashed lines indicate the performance of the initial seeding step of CMSSW.

	CMSSW	OpenCL		
Pair Generator		grid-based	prediction-based	tight prediction-based
Triplet Predictor		angular-based	angular-based	tight extrapolation-based
Pairs	256 327	7 827 618	616 535	235 596
	×0.90	×7.30	×6.34	×1.97
Candidates	231 427	57 296 731	3 902 299	463 367
	×0.25	×0.000 56	×0.009 0	×0.046 2
Triplets	58 632	33 788	35 482	21 409
Efficiency	68 %	90 %	99 %†	97 %
Fake rate	95 %	90 %	90 %	83 %
Clone rate	0.8 %	1.3 %	1.4 %	2.3 %

Table 10.3.: Number of combinations processed by CMSSW and different configurations of OpenCL-based triplet finding for an event with 4 096 muon tracks. The first factor gives the number of produced triplet candidates per hit pair; the second factor indicates the fraction of valid triplets per candidate.

† the increase in efficiency is due to an enhanced ϕ-prediction in the triplet prediction step.

Comparing the presented algorithm's physical performance with the one of CMSSW suffers from many caveats. As several iterative steps are employed by CMSSW, each with highly tuned parameters, evaluating the efficiency and fake rate for a single step can only provide limited insights into the performance of CMSSW's triplet finding algorithm. Evaluating the combined performance of all steps would constitute an unfair comparison for the presented OpenCL-based algorithm, since many special cases could be treated by the individual steps that would be prohibitive in an algorithm geared towards fast processing.

The parameters of the muon sample are deliberately chosen to produce tracks findable by the *initial* step of CMSSW's seeding, with no transverse impact parameter and a minimum p_T of $1\,\mathrm{GeV}\,c^{-1}$ for the simulated particles. Still, the comparison of the performance measures presented in Figure 10.11 allows only qualitative insights. Both algorithms feature fake rates of the same order. For large samples with more than a thousand tracks originating at the same point, the occupancy of the pixel layers is very high, explaining the tremendous fake rate.

The efficiencies of the algorithms differ by 30 % to 40 %. While the presented algorithm is able to find triplets for almost all of the simulated tracks, CMSSW's initial step only reconstructs triplets for 60 % of them. Tracks not found by the initial step of CMSSW are reconstructed in later iterations, employing less restrictive seeding algorithms. Using these algorithms in the initial step, however, would not be representative for the runtime behavior of CMSSW.

Nevertheless, the qualitative insight that the presented algorithm achieves a physical performance of the same order as CMSSW remains and gives credence to the runtime measurements presented in the following section.

The merit of more precise calculations in pair building and triplet prediction are studied with the largest produced event, containing 4 096 muon tracks. The grid-based pair generator produces over 7.8×10^6 pairs, a factor 32 more than CMSSW, as listed in Table 10.3. By predicting the z- and ϕ-range in pair building, the number of pairs can be reduced by factor 10. The more precise prediction also enables the use of *tight* selection as introduced in Sections 9.3 and 9.4, further decreasing the number of pairs by factor three to about the same order as CMSSW. Extrapolation-based triplet prediction is also essential to contain the amount of processed combinations. For every hit pair, it produces ≈ 2 triplet candidates, opposed to over 6 candidates per pair with angular-based triplet prediction. Combining both methods reduces the the number of found triplet candidates by two orders of magnitude, thus justifying the more complex calculations involved.

10.3. Algorithmic properties

The primary goal of this thesis is the design of a fast and efficient triplet finding algorithm, capable to cope with the increased event complexity of the upcoming run period in 2015. Having established the physical efficiency of the algorithm in the previous section, alongside the derivation of the appropriate physical configuration parameters, the runtime behavior of the implementation is examined in the following.

A central algorithmic tuning parameter of OpenCL-based algorithms is the work-group size, which determines the number of work-items executed by a single compute-unit [171]. Factors influencing the choice of work-group size are

- the underlying hardware's maximum work-group size, which is 1 024 for both Intel's and NVIDIA's OpenCL SDK;

- the *private* memory demand of the executed kernel, limiting the work-group size for kernels with many local variables;

			cell size		
Configuration	$\#_{\mathbf{z}}$	$\#_{\phi}$	**z** [cm]	ϕ [rad]	**local memory**
coarse	300	8	2	$\frac{\pi}{4}$	local grid *count* and *store* local pair gen
medium	300	16	2	$\frac{\pi}{8}$	local grid *count* grid *store* only local `written` local pair gen
fine	600	32	1	$\frac{\pi}{16}$	no local memory

Table 10.4.: Configuration of the grid data structure and resulting usage of *local* memory in grid building and pair generation kernels.

- the available *local* memory of a compute unit if the local memory consumption of the kernel depends on the work-group size; and
- the desired sequential work-load for every work-item, for kernels processing a fixed number of data points with a variable number of work-items.

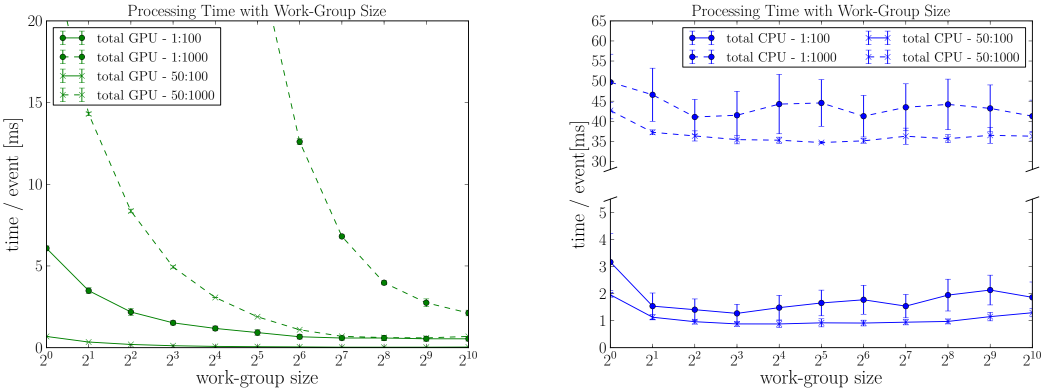
The latter point is of particular significance to grid building and pair generation, as the three-dimensional index space – refer to Figure 8.5 – employed for both kernels foresees one work-group to process all hits of its assigned detector layer. Hence, an increasing work-group size decreased the sequential load. In the one-dimensional index space used in the triplet prediction and filter kernels, the sequential work-load is not sensitive to the work-group size.

Local memory is only used in grid building and pair generation. The amount of required local memory does *not* depend on the work-group size, but rather on the granularity of the grid-data structure. Table 10.4 lists the grid configurations used in the further runtime studies and their resulting usages of local memory. In the *medium* configuration, preference is given to the counter of written elements per grid cell for local memory storage in the grid building *store* kernel, since it needs to be accessed via atomic operations – refer to Section 8.2.

The following sections present studies on the impact of the work group size (Section 10.3.1), event parallelism (Section 10.3.2), the grid granularity (Section 10.3.3) and the track multiplicity (Section 10.3.5) on the runtime of OpenCL-based triplet finding. All studies are conducted with the muon track sample.

10.3.1. Work-Group Size

Figure 10.12 illustrates the total *kernel* time of all triplet finding steps for varying work-group sizes. The experiments are conducted for small (100 tracks) and large (1 000 tracks) events, with one and fifty concurrently processed events. The CPU is *not* sensitive to the choice of work-group size and exhibits similar performance for all tested sizes. Processing more events concurrently results in a modest decrease of processing time per event. The GPU displays the contrary behavior: for one, the runtime depends strongly



(a) GPU.

(b) CPU.

Figure 10.12.: Kernel time of triplet finding for varying work-group sizes. The label $e : t$ indicates the number of concurrently processed events e and the number of tracks per event t.

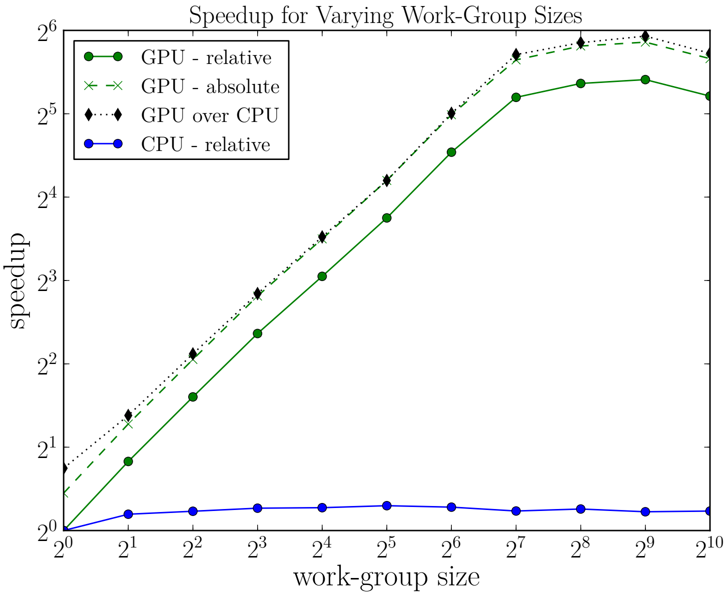


Figure 10.13.: Speedup over work-group size for 50 concurrently processed events with 1 000 tracks each. Relative speedup is with respect to the runtime with a work-group of size 1 on the same device type. The absolute speedup for the GPU is with respect to the minimum runtime of the CPU. The ratio refers to the speedup of the GPU over the CPU for the same work-group size.

on the work-group size, showing gains of more than an order of magnitude for the right parameter choice. Moreover, the kernel time per event is reduced by a factor of two when processing 50 events concurrently.

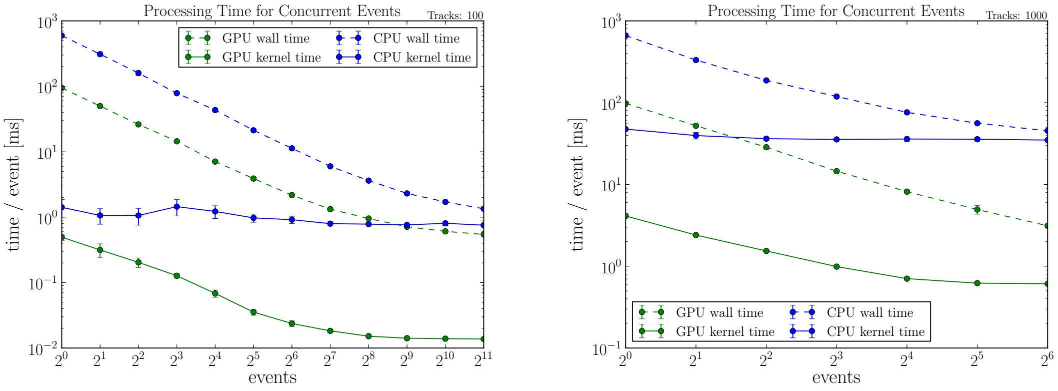
Studying the speedup of the algorithm with respect to work-group size reveals further insights. Figures 10.13 and 10.1 show, that the triplet finding algorithm behaves comparably to the prefix sum implementation with respect to work-group size sensitivity. The prefix sum algorithm reaches its peak relative speedup of 64 for work-group size 128, whereas the triplet finding peaks at 32 for 256 work-items per work-group. The lower maximum speedup can be attributed partly to the higher number of global memory accesses for triplet finding, as well as to the thread divergence induced by the different number of hits processed within the search window in the pair building and triplet prediction steps. In both algorithms, the CPU achieves its maximum speedup for work-group size 32. The achieved maximum relative speedup of ≈ 1.2 for triplet finding – compared to 3 for the prefix sum algorithm – indicates that the algorithm scales worse on the CPU than on the GPU. The CPU's performance is hampered by the immaturity of the auto-vectorization features in Intel's OpenCL compiler, thus the hardware's SIMD capabilities remain unexploited. Contact with Intel's OpenCL compiler team has been established in order to investigate the matter [132].

Plots of the behavior of individual processing steps with varying work-group sizes are presented in Appendix B. All further experiments are conducted with the devices' optimal work-group sizes of 32 for the CPU and 256 for the GPU.

10.3.2. Concurrent Events

The study of various work-group sizes already evinced the importance of high workloads to fully leverage the hardware's – particularly the GPU's – potential. To further substantiate the insight, the *kernel* time and *wall* time for concurrently processed events is studied, refer to Figure 10.14. The wall time is measured for the entire triplet finding algorithm but does *not* comprise the transfer of hits to the compute device. Including the transfer time of the hit data to the measured wall time would add a constant offset per event that could hide more subtle differences of kernel and wall time due to

- allocation of memory for counters and oracle bit-strings,



(a) 100 tracks per event.

(b) 1 000 tracks per event.

Figure 10.14.: Runtime per event for concurrent triplet finding in many events of different size.

- transfer of total number of found outputs after prefix sum kernel execution,
- allocation of the output array by the host between *count* and *store* kernel and
- overhead due to the OpenCL implementation of the hardware's vendor.

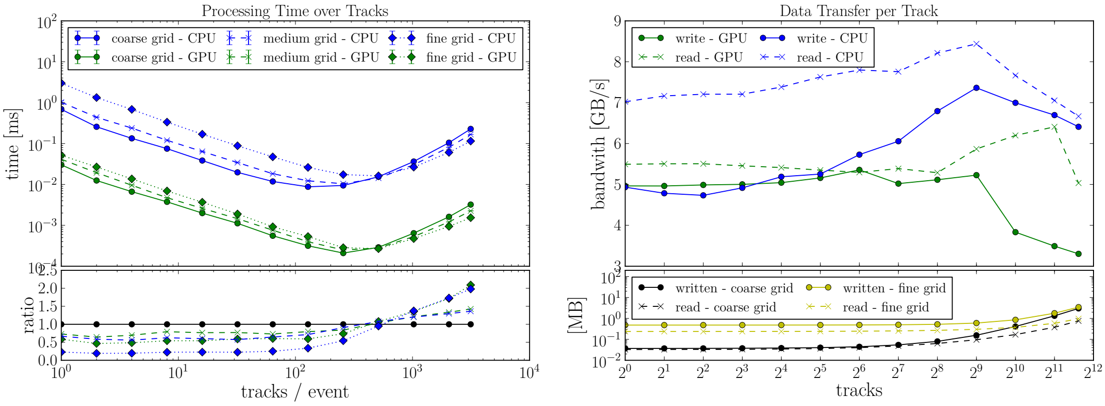
The larger gap between kernel time and wall time on the GPU is partly due to the higher memory allocation and data transfer times and partially due to the improvable quality of NVIDIA's OpenCL implementation [85]. Hence, more and larger events are required to amortize the overhead of the GPU. For small events (100 tracks, Figure 10.14a), the gap is still quite pronounced for 2 048 concurrently processed events, indicating that GPU employment is only beneficial for events with many tracks. As shown in Figure 10.14b, the overhead amortizes more quickly for large events with 1 000 tracks. The 3 GByte of main memory on the available GPU restricts the number of concurrently processed large events to merely 64. Increasing the device's main memory could therefore further aid the amortization of the incurred overhead.

10.3.3. Grid Granularity

The grid data structure's influence on the algorithm's performance is manifold and depicted in Figure 10.15. For small events, fine-grained grids incur a performance penalty due to the loss of local memory caching in the grid building and pair generation kernels, resulting in an increase in high-latency global memory accesses. Furthermore, since the size of the data structure grows with the number of grid cells – $\mathcal{E} \cdot \mathcal{L} \cdot \#_z \cdot \#_\phi \cdot 4\,\text{Byte}$[1] – more data needs to be transfered between host and device. Considering large events, these initially higher costs amortize as fewer hit combinations are processed, reflected in both runtime as well as transfered data volume.

---

[1] $\mathcal{E}$ concurrently processed events, $\mathcal{L}$ detector layers and $\#_z \cdot \#_\phi$ grid cells.



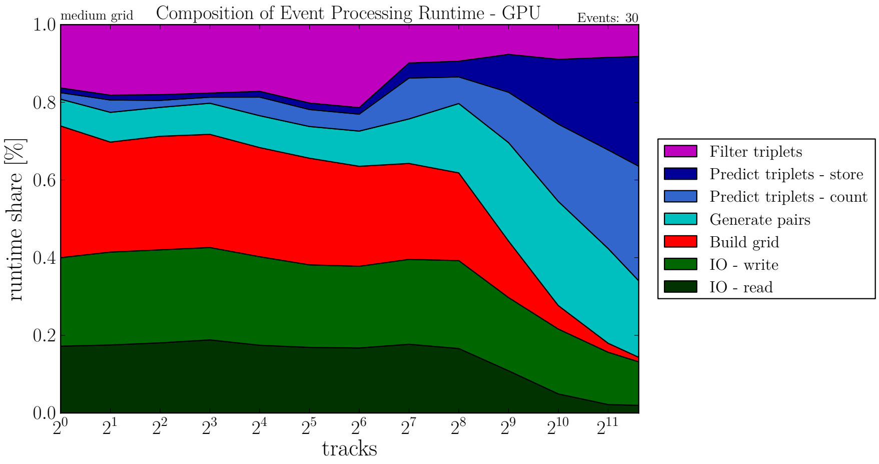
(a) Runtime and ratio of varying grid granularities.

(b) Bandwidth and transfered data volume.

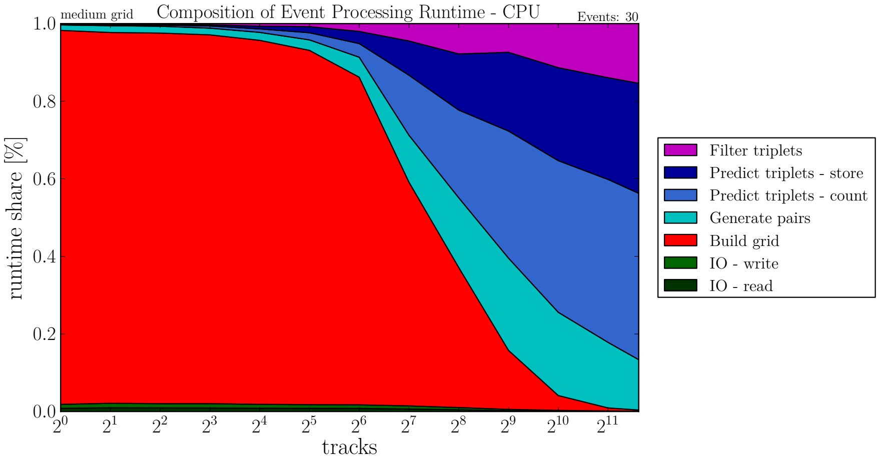
Figure 10.15.: Influence of grid granularity on runtime and data volume for one processed event. The configuration of grid granularities follow Table 10.4.

10.3.4. Runtime Composition

Studying the share of the individual processing steps of the entire runtime reveals further understanding of the algorithm's behavior. Figure 10.16a depicts the runtime composition for GPU execution of the algorithm. The I/O time *includes* the hit data transfer in order to present its relation to the execution time of triplet finding. Building the grid data structure requires a sizable portion of the runtime for events with fewer than ≈ 250 tracks, a trait that is even more pronounced in the CPU's runtime profile (Figure 10.16b). For larger events, triplet building becomes the predominant contributor to the algorithm's runtime. Many accesses to global memory for hit positions and thread divergence in



(a) GPU.



(b) CPU.

Figure 10.16.: Share of individual processing steps of total runtime for the medium granularity of the grid data structure.

the inner loop of the kernel impair the runtime. Data transfer times on CPU are negligible since only pointers to the data need to be provided to the OpenCL runtime. Appendix B presents further plots of runtime compositions for other grid granularities. The main observed features remain identical, with the I/O contribution becoming more predominant as processing times lessen because of fewer processed hit combinations with a more fine-grained grid.

10.3.5. Tracks per Event

The runtime behavior for growing input sizes is one of the most basic characteristics of an algorithm. Figure 10.17 displays the runtime of OpenCL-based triplet finding and CMSSW for varying number of tracks. OpenCL suffers from significant constant overhead which dominates the runtime for events with less than 200 tracks. The overhead can be mitigated by processing several events concurrently. The GPU's main memory limits the event parallelism to merely 30 due to the largest sample with 4 096 tracks. however, the overhead amortization for large events is apparent. The OpenCL-based triplet finding outperforms CMSSW for events with more than 1 000 tracks by a factor of 64 if executed on a GPU. If a CPU serves as compute device, it achieves a performance similar to CMSSW. As discussed in Section 10.3.1, improvement's to Intel's OpenCL compiler could aid the algorithm in surpassing CMSSW when performed on a CPU. Furthermore, OpenCL-based triplet finding processes about twice as many triplet candidates as CMSSW (Table 10.3) due to the simpler geometric calculations, even in the extrapolation-based

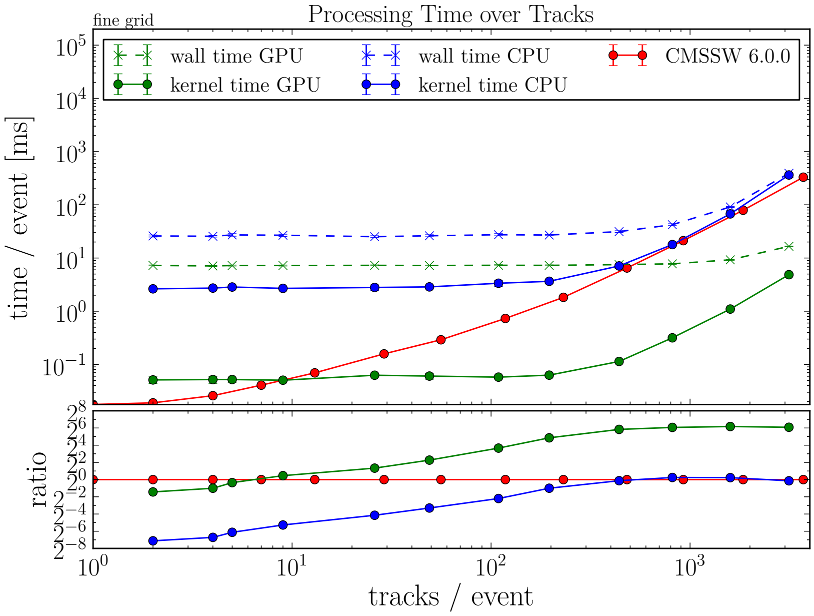


Figure 10.17.: Runtime of OpenCL-based triplet finding in comparison to the initial seeding step of CMSSW. The OpenCL implementation processes 30 events concurrently to study the amortization of wall time overhead.

triplet prediction. Further limiting the number of candidates, e. g. by employing a circular trajectory extrapolation to reduce the predicted ϕ-range, could improve the algorithm's performance.

In Figure 10.17, the fine grid configuration is used, yielding a runtime sweet spot for events with ≈ 800 to 2 000 tracks. Employing a more coarse-grained grid results in a sweet spot for fewer tracks per event, refer to Appendix B. Hence, the grid data structure's parameters need to be adapted to the anticipated number of tracks per event.

Chapter 11

Conclusion

The CMS experiment at the LHC particle accelerator has already led to many profound insights into the fundamental interactions of the universe's particles, most noticeably the ground-breaking discovery of a Higgs-like boson in 2012. These insights were enabled by the study of precisely reconstructed particle trajectories, produced by proton-proton collisions with an unprecedented center-of-mass energy of 7 TeV. The reconstruction of tracks from the interaction of traversing particles with the detector material poses a tremendous computational challenge. When the LHC resumes operation in 2015 with an increased luminosity and center-of-mass energy of 14 TeV, this challenge becomes even greater. As the number of simultaneous proton-proton collisions increases with the leap in energy, more particles will be traversing the detector. Hence, the reconstruction of particle tracks – a combinatorial pattern recognition problem – needs to cope with a significantly higher combinatorial complexity for the involved algorithms. The combination of increased combinatorics and the stagnation of clock frequencies of individual CPU cores for the last years gives rise to the need to examine alternatives more suitable for parallelization and modern computing technology than the currently employed iterative Kalman filter-based track finding. One of the pursued approaches is Cellular Automaton-based track finding, using simple, local and parallelizable computations to form particle tracks by joining compatible triplets of energy deposits – hits – in adjacent layers of the tracking system.

This thesis presents the *design* of a parallel algorithm to identify *valid* triplets of hits, originating from the same particle's path through the detector, while discarding *fake* ones – triplets comprising hits of different particles' trajectories. Even though facing the complex detector geometry of CMS, simple filter criteria are employed to separate valid from fake triplets. In order to limit combinatorics in the search for triplets, the feasible range for an additional hit is predicted based on the information from a given individual hit or hit pair. A uniform grid data structure is used to efficiently answer queries for hits within the predicted range.

Parallelism is exploited on three levels, processing multiple events and layer configurations concurrently as well as treating individual hits, pairs of hits and triplet candidates simultaneously. The diverse computing landscape of the Worldwide LHC Computing Grid requires to address this parallelism in a portable manner, exploiting the capabilities of various processor- and graphic card types – the latter in a first attempt to bring GPU computing to CMS. OpenCL as an open framework for heterogeneous, massively-parallel computing provides the ideal platform to implement the presented algorithm, however requires careful consideration in the algorithm design.

The presented algorithm is *validated* by studying simulated $t\bar{t}$ and QCD collision events. It achieves high quality of the physical output – identifying on average 80 % of the simulated particle tracks. The number of fake triplets in the produced results varies with the detector region, being as low as 15 % in the inner, most precise, pixel detector and as high as 60 % in the less precise silicon-strip detector. Several prediction methods for pair generation and triplet finding are studied, attesting the more complex ones a dramatic decrease in the inspected hit combinations. Since all subsequent processing steps benefit from the reduced combinatorics, the more compute-intense calculations are justified.

In *benchmarks*, the algorithm proves to be very suitable to address *many* concurrently processed events with *high* activity. In comparison to the initial triplet seeding algorithm of the CMS experiment software, the presented algorithm achieves a speedup of 64 for events with more than 1 000 tracks when executed on a GPU. Performed on the CPU, it exhibits a similar runtime as the experiment software's algorithm. To fully exploit the capabilities of a GPU and amortize associated OpenCL overheads, a high workload is essential. Therefore, the processing of multiple concurrent events is eminent. For small events with about 100 tracks, using the GPU yields little merit even for a large number of concurrent events, thus they should be processed solely by the CPU. The recently announced next generation of GPUs [135] promises even greater gains in the face of multiple, large events. OpenCL proves to be capable to exploit both hardware platforms with no effort required to switch between the two. The grid data structure provides efficient access to the measured hits in the predicted range. The granularity of the data structure needs to be carefully weighed to balance the time and space required for data structure construction and storage with the merit of fewer hits per grid cell. A fine-grained grid is most beneficial for large events.

The triplet finding algorithm is implemented outside the experiment's software framework. Therefore, its integration remains for *future work*. Processing multiple events concurrently requires restructuring the current framework's data flow; a matter addressed by [82]. As the algorithms performance depends highly on properly tuned parameters – grid granularity and number of concurrently processed events – for the event size, heuristics based on the expected number of proton-proton interactions for the current particle beam parameters should be examined. As the goal of the presented work is to assess the validity of a simplified algorithmic approach to triplet finding and the merit of exploiting modern CPUs and GPUs, the geometric calculations are restricted to the barrel region of the CMS detector. For productive use, the endcap and transition regions need to be addressed as well.

In *conclusion*, an efficient, both in the physical as well as algorithmic sense, triplet finding algorithm is presented in this thesis, capable of coping with the processing challenges ahead.

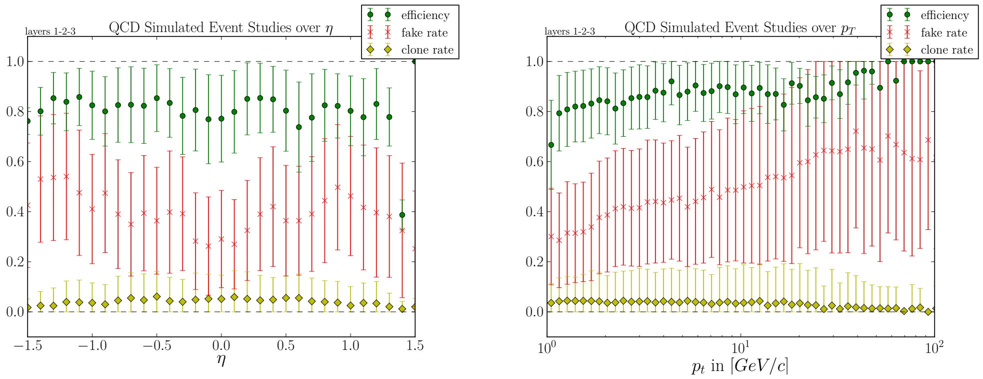
Part III.

Appendices

Appendix A

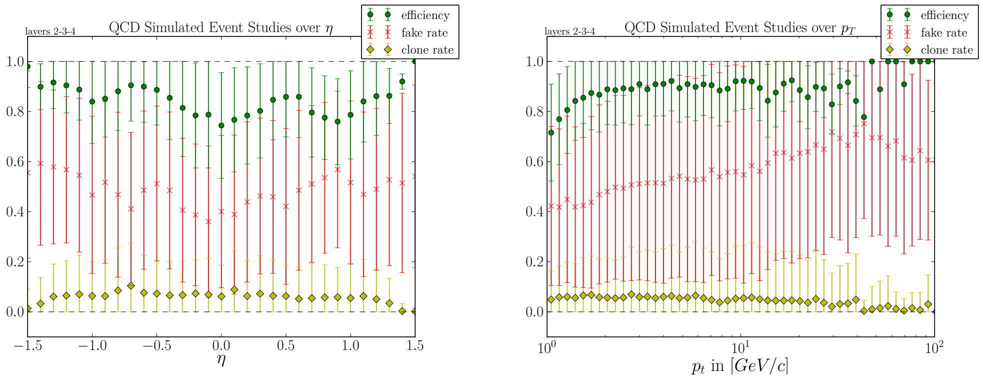
Supplement to Physics Performance

Performance in QCD Events



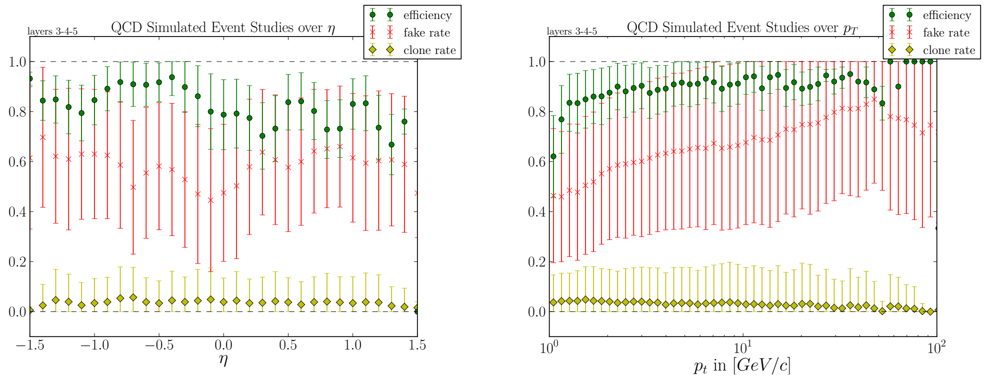
(a) Physics performance measures over η. (b) Physics performance measures over p_T.

Figure A.1.: Efficiency, fake rate and clone rate for QCD events layers *1-2-3*.



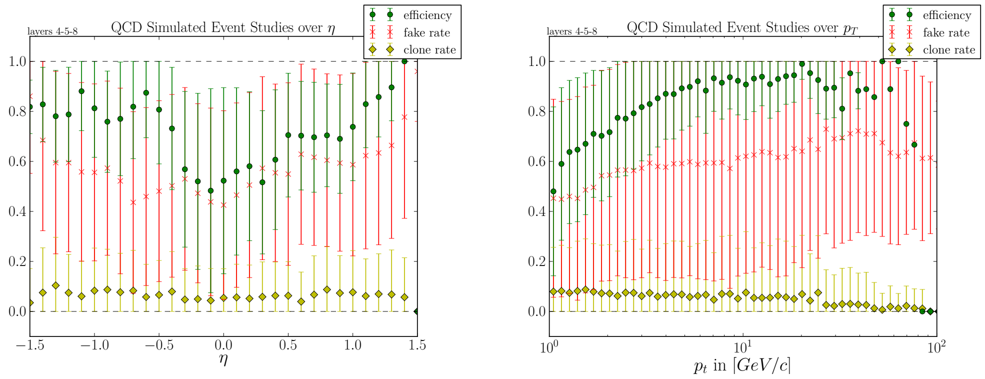
(a) Physics performance measures over η. (b) Physics performance measures over p_T.

Figure A.2.: Efficiency, fake rate and clone rate for QCD events in layers *2-3-4*.



(a) Physics performance measures over η. (b) Physics performance measures over p_T.

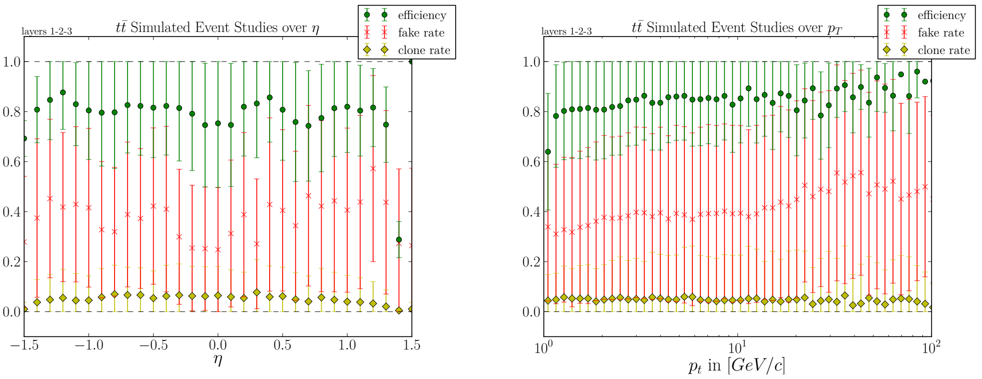
Figure A.3.: Efficiency, fake rate and clone rate for QCD events in layers *3-4-5*.



(a) Physics performance measures over η. (b) Physics performance measures over p_T.

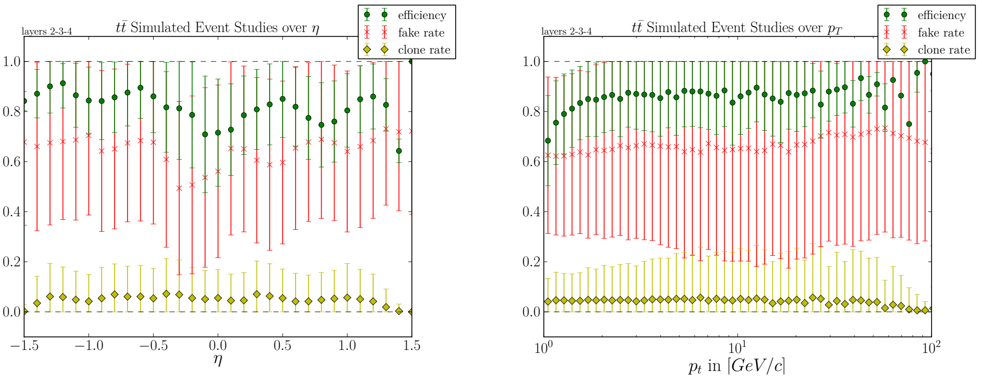
Figure A.4.: Efficiency, fake rate and clone rate for QCD events in layers *4-5-8*.

Performance in $t\bar{t}$ Events



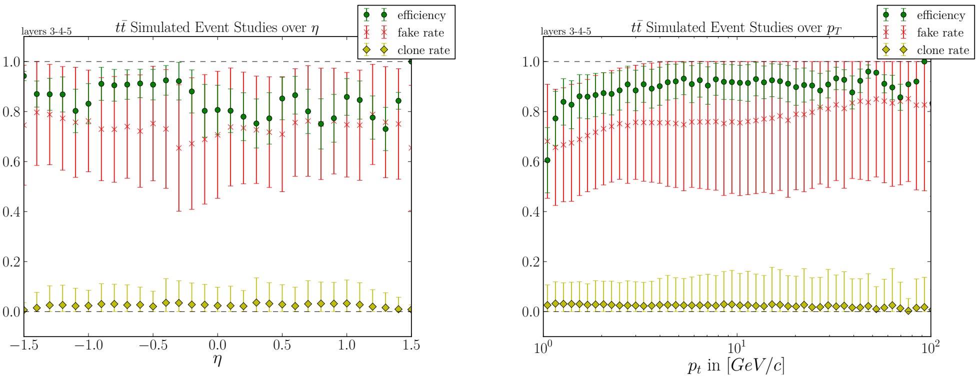
(a) Physics performance measures over η. (b) Physics performance measures over p_T.

Figure A.5.: Efficiency, fake rate and clone rate for $t\bar{t}$ events layers *1-2-3*.



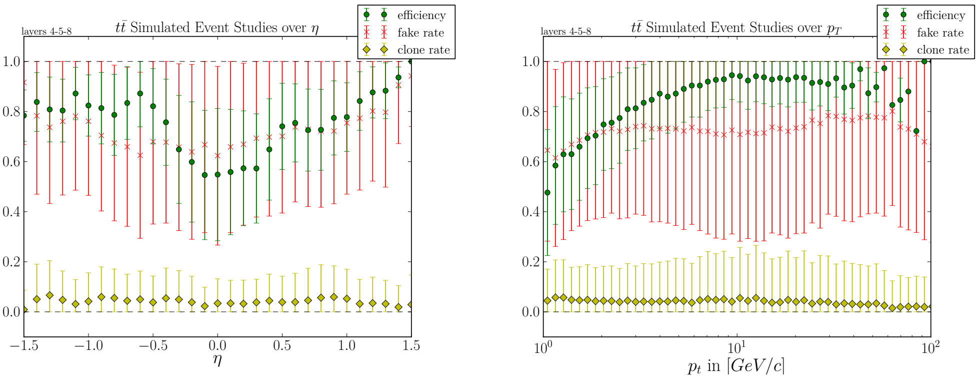
(a) Physics performance measures over η. (b) Physics performance measures over p_T.

Figure A.6.: Efficiency, fake rate and clone rate for $t\bar{t}$ events in layers *2-3-4*.



(a) Physics performance measures over η. (b) Physics performance measures over p_T.

Figure A.7.: Efficiency, fake rate and clone rate for $t\bar{t}$ events in layers *3-4-5*.



(a) Physics performance measures over η. (b) Physics performance measures over p_T.

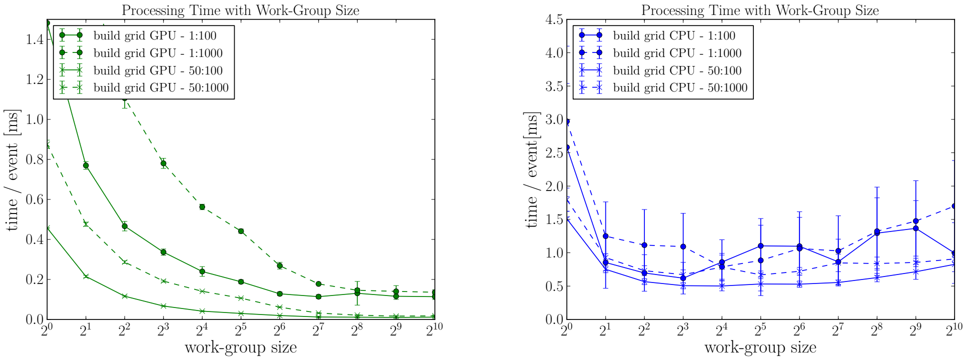
Figure A.8.: Efficiency, fake rate and clone rate for $t\bar{t}$ events in layers *4-5-8*.

Appendix B

Supplement to Algorithmic Performance

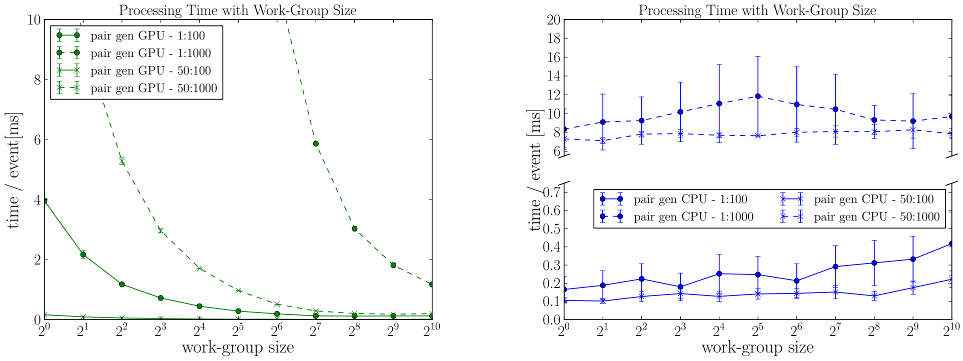
Work-Group Size Studies

In the following plots, the label $e : t$ indicates the number of concurrently processed events e and the number of tracks per event t.



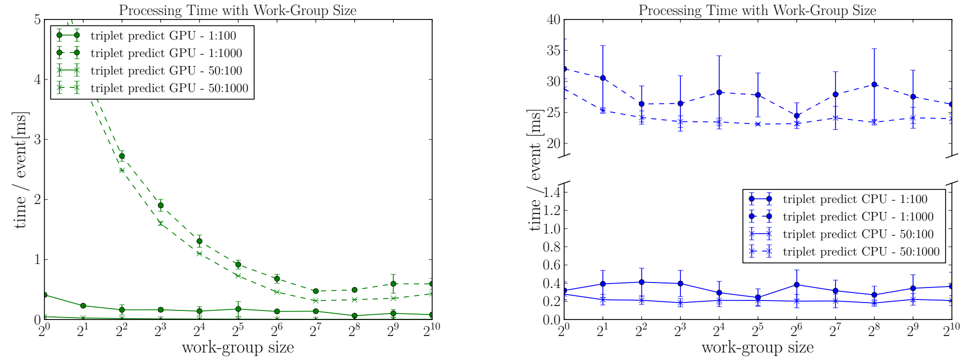
(a) GPU. (b) CPU.

Figure B.1.: Kernel time of *grid building* for varying work-group sizes.



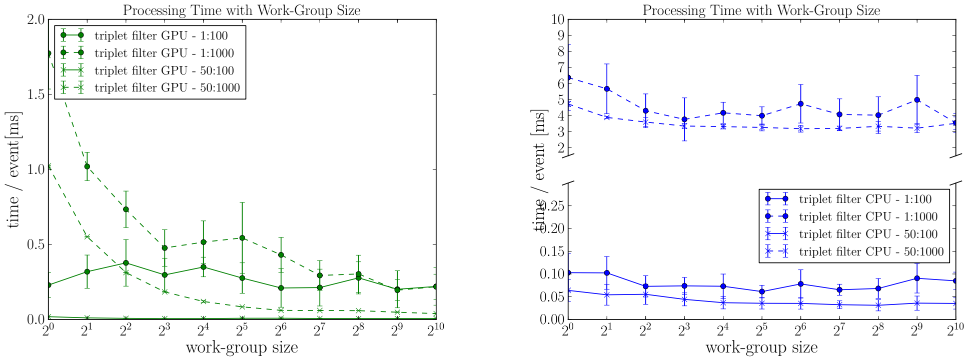
(a) GPU. (b) CPU.

Figure B.2.: Kernel time of *pair generation* for varying work-group sizes.



(a) GPU. (b) CPU.

Figure B.3.: Kernel time of *triplet prediction* for varying work-group sizes.

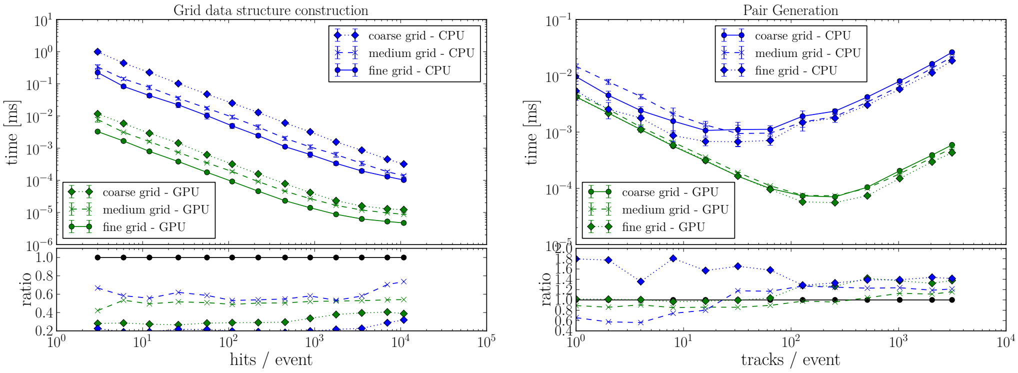


(a) GPU. (b) CPU.

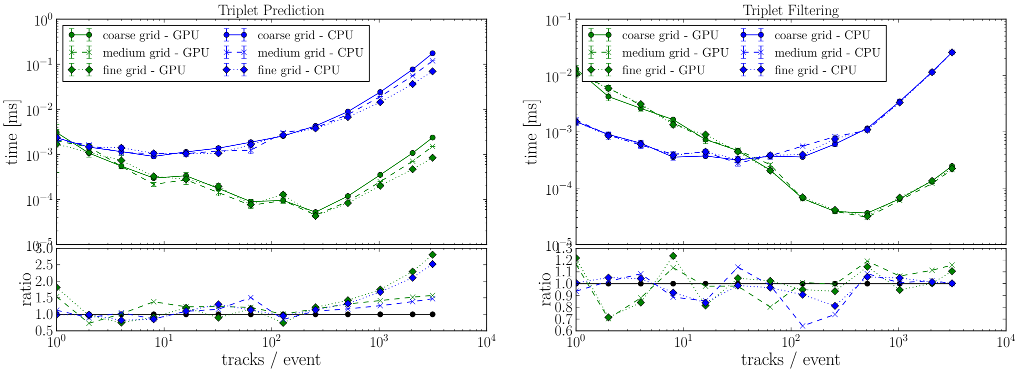
Figure B.4.: Kernel time of *triplet filtering* for varying work-group sizes.

Grid Granularity Studies

The following plots demonstrate the behavior of individual processing steps with different grid data structure granularities. The granularities are defined according to Table 10.4.

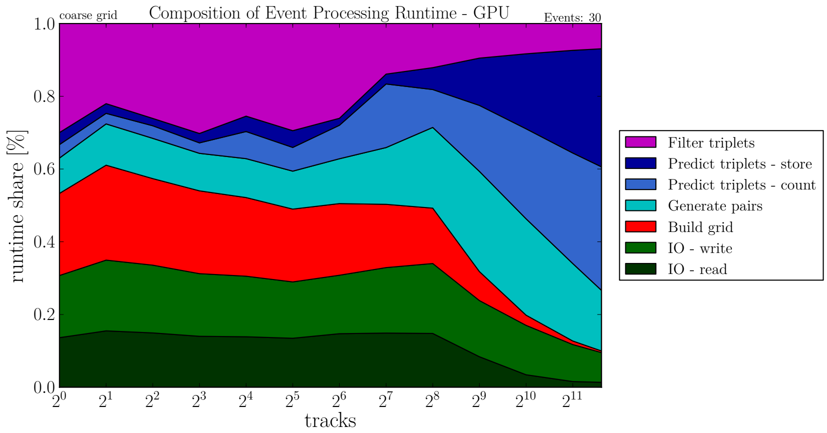


(a) Grid building.
(b) Pair generation.

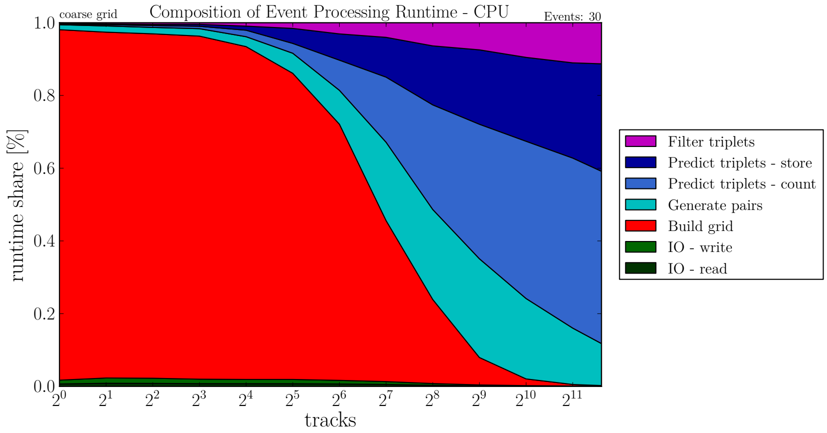


(c) Triplet prediction.
(d) Triplet filtering.

Figure B.5.: Kernel time of individual processing steps for varying grid granularities.

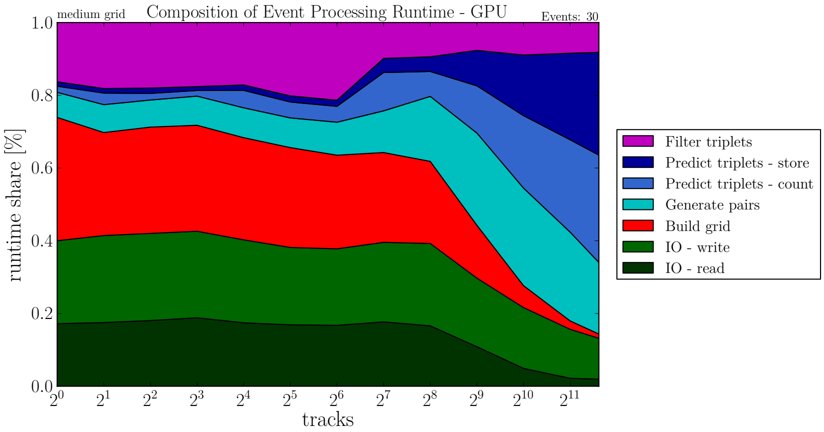


(a) GPU.

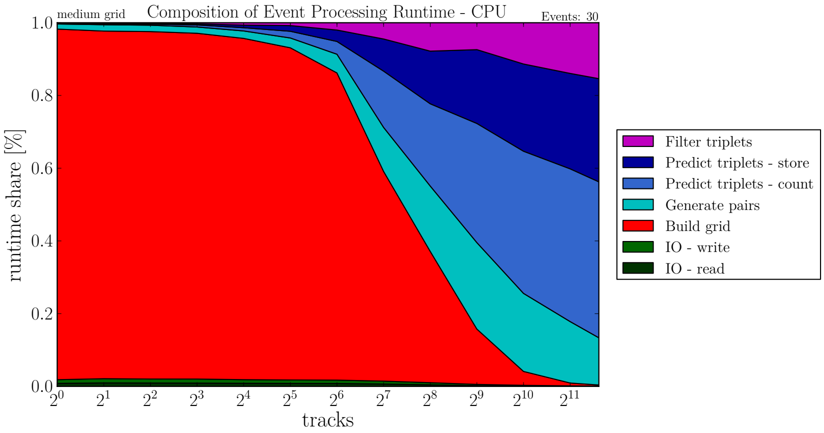


(b) CPU.

Figure B.6.: Contribution of individual processing steps for *coarse* grid configuration.

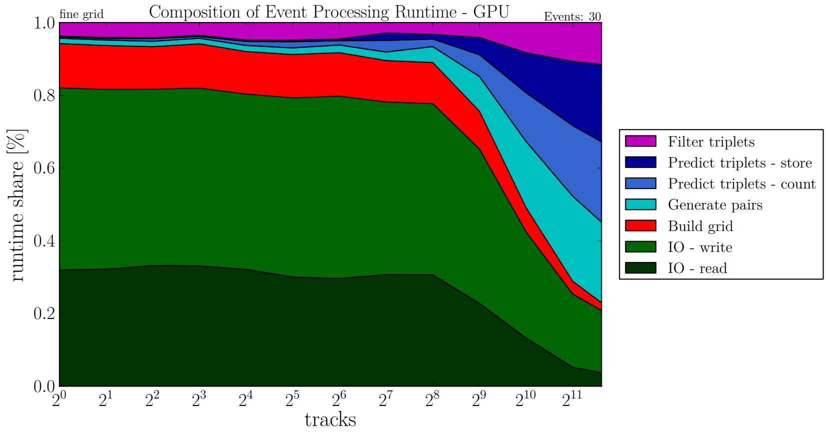


(a) GPU.

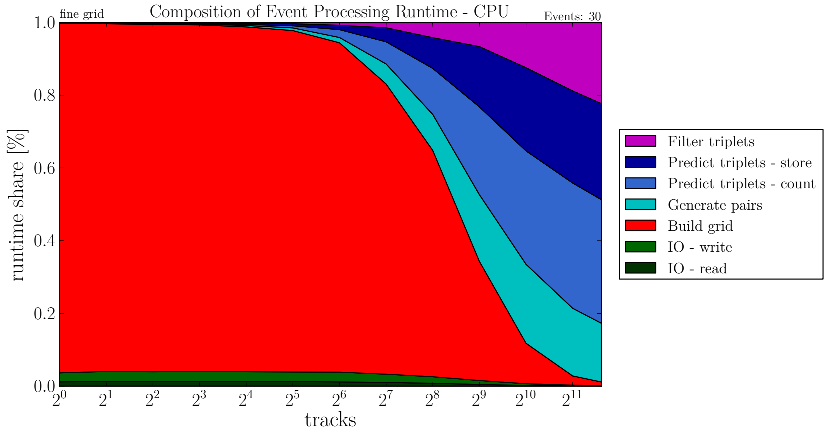


(b) CPU.

Figure B.7.: Contribution of individual processing steps for *medium* grid configuration.



(a) GPU.



(b) CPU.

Figure B.8.: Contribution of individual processing steps for *fine* grid configuration.

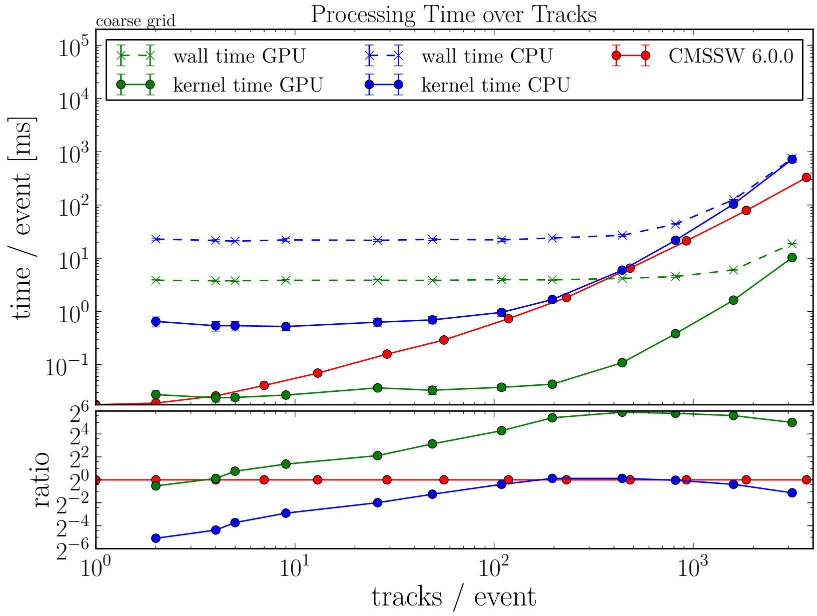


Figure B.9.: Runtime for muon track sample for *coarse* grid configuration.

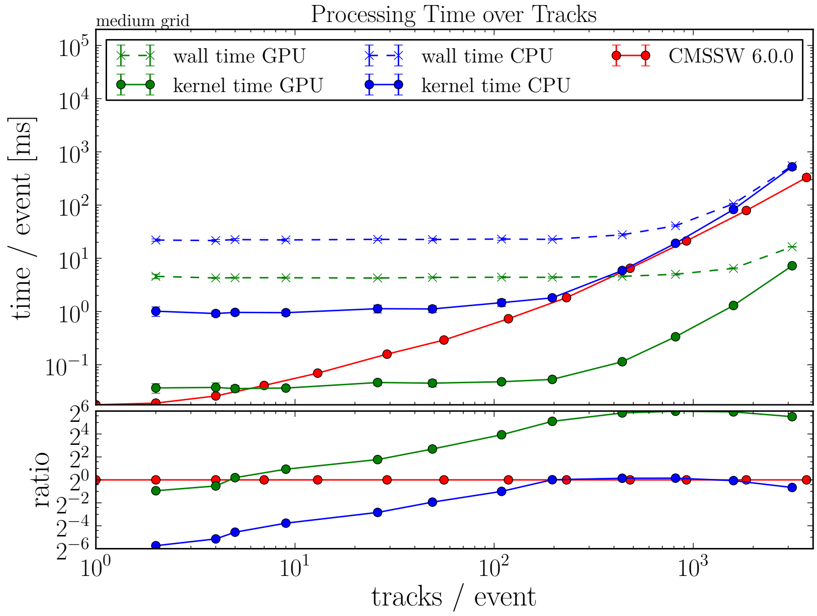


Figure B.10.: Runtime for muon track sample for *medium* grid configuration.

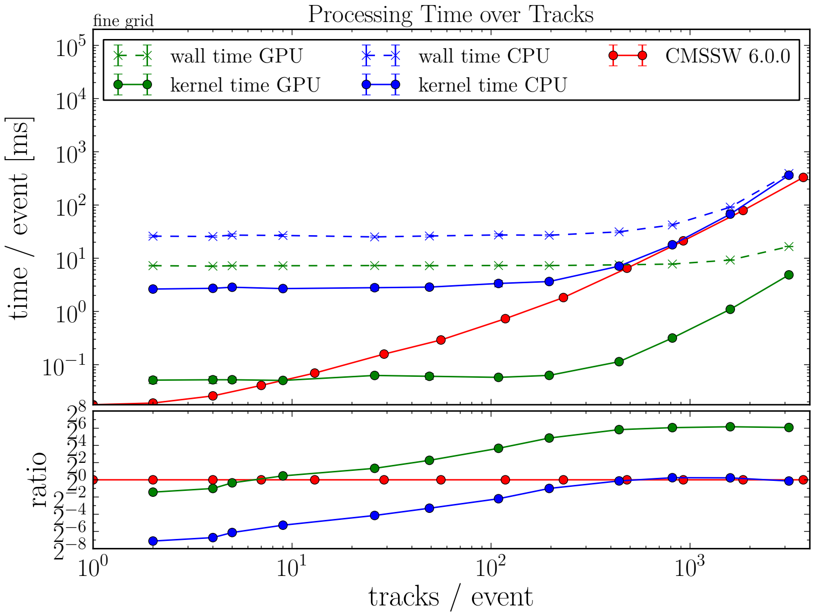


Figure B.11.: Runtime for muon track sample for *fine* grid configuration.

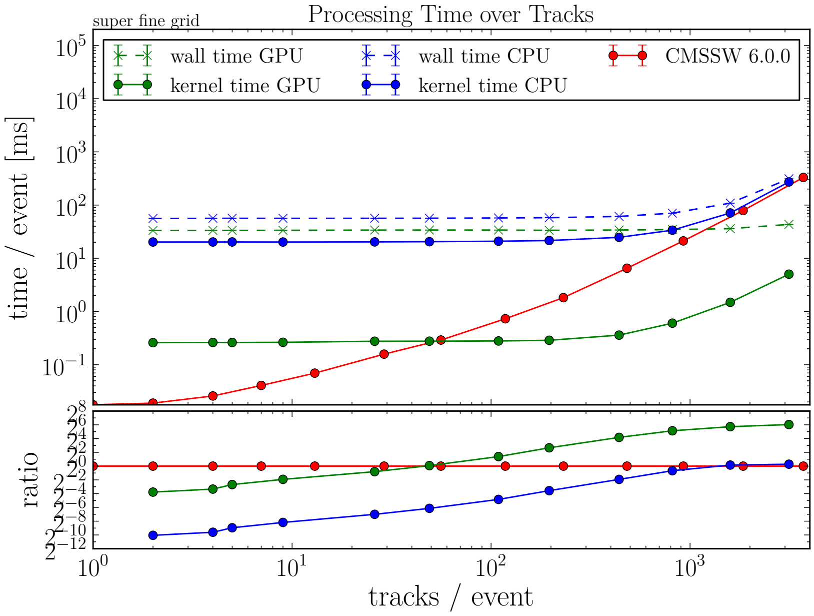
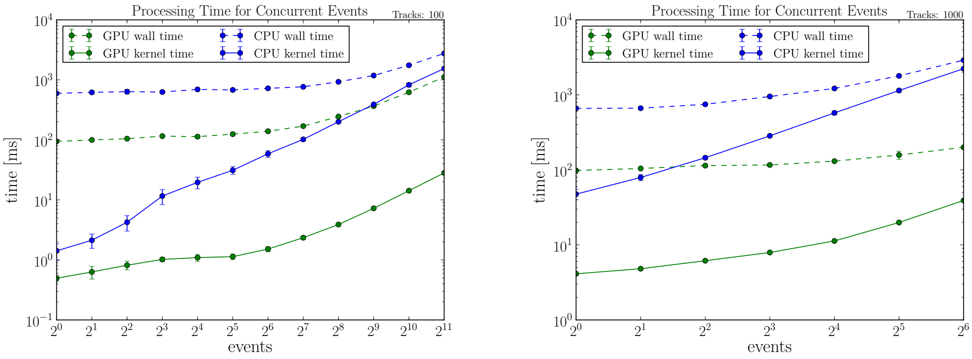


Figure B.12.: Runtime for muon track sample for *super fine* grid configuration.

Concurrent Events Studies

The following figures illustrate the behavior of the algorithm for processing multiple *small* (100 tracks) and *large* (1 000 tracks) events.



(a) Small events.

(b) Large events.

Figure B.13.: Kernel time for processing multiple events concurrently.

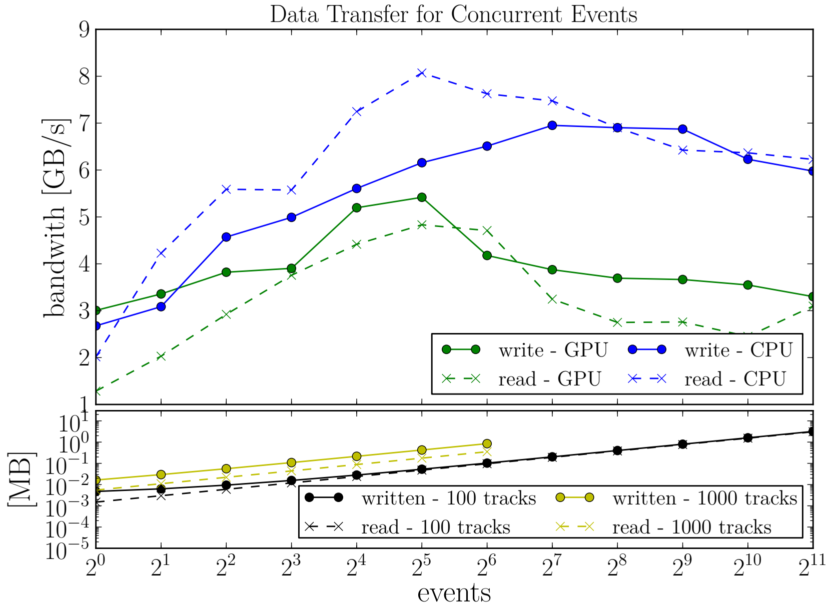
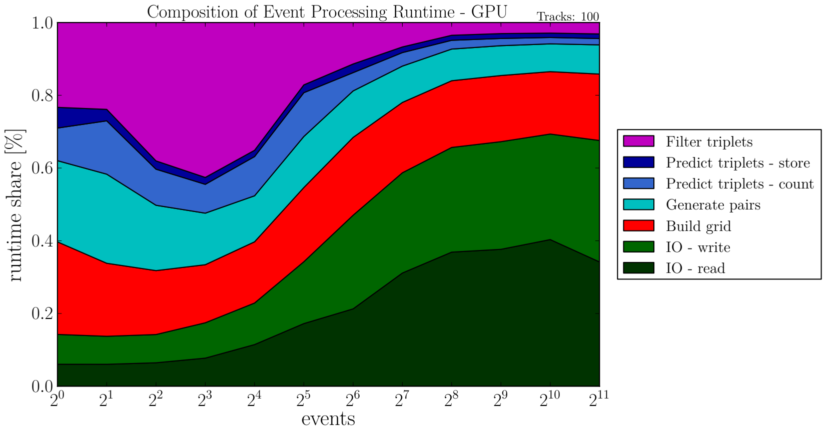
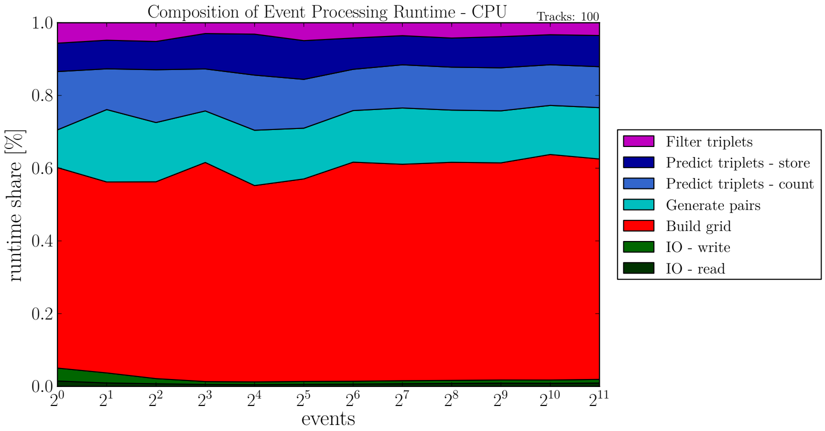


Figure B.14.: Transferred data volume and bandwidth for processing multiple events.

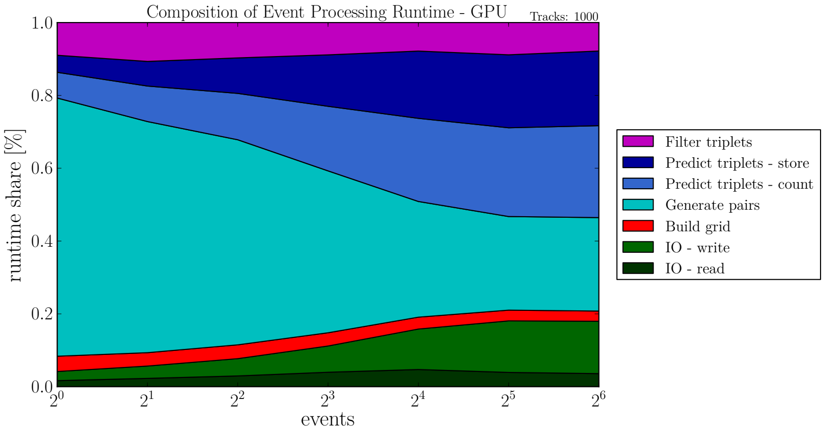


(a) GPU.

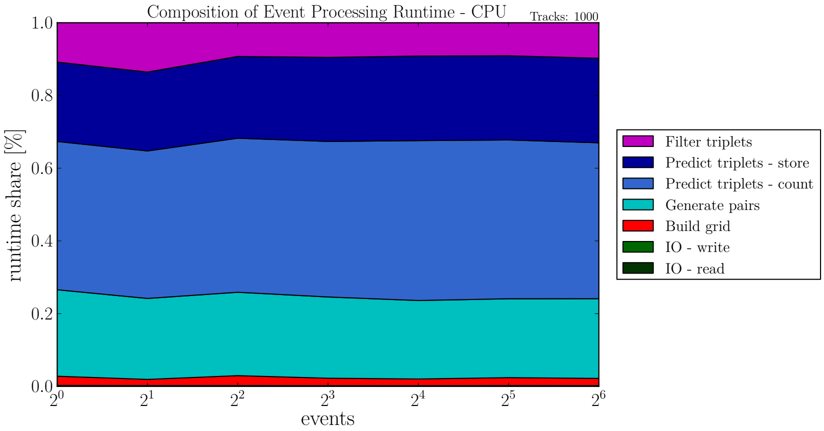


(b) CPU.

Figure B.15.: Contribution of individual processing steps for *small* events.



(a) GPU.



(b) CPU.

Figure B.16.: Contribution of individual processing steps for *large* events.

Appendix C

Configuration Parameters

Grid Data Structure Configuration

Parameter	**Technical**	**Remarks**
z_{min}	`minZ`	grid extend – $z_{\text{min}} = -300$ for tracker
z_{max}	`maxZ`	grid extend – $z_{\text{min}} = 300$ for tracker
ϕ_{min}	`minPhi`	grid extend – $\phi_{\text{min}} = -\pi$
ϕ_{max}	`maxPhi`	grid extend – $\phi_{\text{max}} = \pi$
$\#_z$	`nSectorsZ`	number of grid cells in z
$\#_\phi$	`nSectorsPhi`	number of grid cells in ϕ
	`sectorSizeZ`	derived – $\frac{z_{\text{max}} - z_{\text{min}}}{\#_z}$
	`sectorSizePhi`	derived – $\frac{\phi_{\text{max}} - \phi_{\text{min}}}{\#_\phi}$

Table C.1.: Parameters for the grid data structure.

Event Data Loading Configuration

Parameter	**Technical**	**Remarks**
	`eventDataSrc`	data file with event data
	`maxEvents`	number of events to process
$\mathcal{L}$	`maxLayer`	outermost layer to load – derived – $\max l_3$
*	`maxTracks`	number of tracks to load per event
* $p_{T,\text{min}}$	`minPt`	minimum p_T of tracks to load
*	`onlyTracks`	merely consider tracks with a hit in each layer

Table C.2.: Parameters for event data loading. Parameters marked with * are only valid for simulated data.

Layer Configuration and Triplet Parameters

Parameter	Technical	Remarks
l_1	`layer1`	inner layer of triplet
l_2	`layer2`	middle layer of triplet
l_3	`layer3`	outer layer of triplet
$d\theta$	`dThetaCut`	quality measure on $\frac{\theta'}{\theta}$ Equation (7.4)
$d\phi$	`dPhiCut`	quality measure on $\lvert\phi' - \phi\rvert$ Equation (7.5)
d_0	`d0`	maximum Transverse Impact Parameter
z_0	`z0`	maximum Longitudinal Impact Parameter
p_θ	`dThetaWindow`	prediction window for θ in Algorithm 9.6
p_ϕ	`dPhiWindow`	prediction window for ϕ in Algorithm 9.8
s_z	`pairSpreadZ`	neighborhood size in z for pair building Algorithm 9.4
s_ϕ	`pairSpreadPhi`	neighborhood size in ϕ for pair building Algorithm 9.4
σ_z	`sigmaZ`	extra tolerance for z-range prediction Algorithms 9.5 / 9.9
σ_ϕ	`sigmaPhi`	extra tolerance for ϕ-range prediction Algorithms 9.5 / 9.10

Table C.3.: Parameters to define a layer triplet for triplet finding. All parameters are per layer combination.

Execution Configuration

Parameter	Technical	Remarks
$\mathcal{T}$	`threads`	work-group size
$\mathcal{E}$	`eventGrouping`	number of events to process concurrently
	`useCPU`	force using CPU instead of GPGPU
	`verbose`	verbose output
	`config`	configuration file to use

Table C.4.: Execution parameters for triplet finding.

Appendix D

List of Figures

1.1 Overview of particle acceleration facilities and experiments at LHC [116]. 10
1.2 Schematic of the CMS detector [44]. 11
1.3 Coordinate system as employed by the CMS collaboration. 12
1.4 Major components of the CMS silicon tracker [44]. 13
1.5 One quarter of the CMS silicon strip tracker [108]. 15
1.6 Schematic CMS detector slice with passing particles [17] 16
1.7 Pile-up events in the CMS detector. 18

2.1 Schematic view of the WLCG grid tier hierarchy including CMS Tier 1 sites and German CMS Tier 2 centers [156]. 20
2.2 CMSSW process containing a path with two sequences, each containing several modules [44]. 22
2.3 Schematics of the CMS trigger and data acquisition system [44]. 23
2.4 Muon reconstruction steps in CMS [2] 25
2.5 Distinction of seeding in the barrel and the endcap region. 30
2.6 Search range restriction for pair and triplet finding in CMSSW. 30

3.1 Comparison of peak performance of CPUs and GPUs. 36
3.2 Schematic of the GPU pipeline. 38
3.3 Schematic of NVIDIA's Kepler architecture. 40
3.4 Schematic of the OpenCL platform, execution and memory model. . . . 41
3.5 Mapping of index space to memory locations. 43

4.1 Illustration of two widely used neighborhood functions for Cellular Automata. 47
4.2 Five detector layers with CA cells defined via segments. 48
4.3 CA-based track formation. 50
4.4 CA-based track selection. 50
4.5 Illustration of a MWPC and its application in the ALICE TPC endplates. 52

5.1 Illustration of a k-d tree for two-dimensional space points [24]. 56
5.2 Illustration of a quadtree for two-dimensional space points [24]. 58
5.3 Illustration of a R-tree for two-dimensional MBRs [111]. 59

5.4 A grid data structure as uniform overlay over two-dimensional space. . . 60

6.1 High level processing steps of OpenCL-based CA track finding. 64

7.1 Schematic angular criteria to discriminate valid and fake triplets. 68
7.2 Schematic of the Transverse Impact Parameter d_0. 70
7.3 Calculating d_0 based on the point of closest approach $\mathbf{p}_{ca}$ of the particle's trajectory. 71

8.1 Layered construction of the CMS inner tracking system. 74
8.2 Schematic drawing illustrating the compressed `DetUnit` radius storage. 74
8.3 Occupancy of the PXB and TIB during $t\bar{t}$ events over z and ϕ. 76
8.4 Event hit data organization on the compute device 77
8.5 Schematic of three-dimensional index space. 79

9.1 Schematic of the balanced trees employed in the two phase prefix sum algorithm by Blelloch [27]. 84
9.2 Schematic of ϕ-slicing of detector induced by the the grid data structure. 87
9.3 Schematics for calculating the maximum $\Delta_\phi = |\alpha - \beta| + \gamma$ for pair finding. 90
9.4 Calculating the feasible z-range for the inner hit of a hit pair. 90
9.5 Illustration of ϕ-wraparound. 92
9.6 Prediction of the feasible z-range of the third hit based upon a straight line extrapolation. 96

10.1 Study of the relative performance of CPU and GPU used in the runtime evaluation. 103
10.2 Study of the longitudinal bending of particle tracks. 104
10.3 Study of the transverse bending of particle tracks. 105
10.4 Study of the transverse impact parameters of particle tracks. 105
10.5 Consecutive application of filter criteria in the pixel layers. 106
10.6 Consecutive application of filter criteria in the pixel layers. 106
10.7 Overview of physics performance for all considered layer combinations for QCD events. 108
10.8 Efficiency, fake rate and clone rate for QCD events in the pixel layers 1-2-3. 108
10.9 Efficiency, fake rate and clone rate for QCD events in the combined pixel and silicon strip layers 3-4-5. 109
10.10 Overview of physics performance for all considered layer combinations for $t\bar{t}$ events. 110
10.11 Physics performance for muon events over number simulated tracks. . . 111
10.12 Kernel time of triplet finding for varying work-group sizes. 114
10.13 Speedup over work-group size. 114
10.14 Runtime per event for concurrent triplet finding in many events. 115
10.15 Influence of grid granularity on runtime and data volume. 116
10.16 Share of individual processing steps of total runtime. 117

10.17 Runtime of OpenCL-based triplet finding in comparison to the initial seeding step of CMSSW. 118

A.1 Efficiency, fake rate and clone rate for QCD events layers *1-2-3*. 125
A.2 Efficiency, fake rate and clone rate for QCD events in layers *2-3-4*. . . . 125
A.3 Efficiency, fake rate and clone rate for QCD events in layers *3-4-5*. . . . 126
A.4 Efficiency, fake rate and clone rate for QCD events in layers *4-5-8*. . . . 126
A.5 Efficiency, fake rate and clone rate for $t\bar{t}$ events layers *1-2-3*. 127
A.6 Efficiency, fake rate and clone rate for $t\bar{t}$ events in layers *2-3-4*. 127
A.7 Efficiency, fake rate and clone rate for $t\bar{t}$ events in layers *3-4-5*. 128
A.8 Efficiency, fake rate and clone rate for $t\bar{t}$ events in layers *4-5-8*. 128

B.1 Kernel time of *grid building* for varying work-group sizes. 129
B.2 Kernel time of *pair generation* for varying work-group sizes. 130
B.3 Kernel time of *triplet prediction* for varying work-group sizes. 130
B.4 Kernel time of *triplet filtering* for varying work-group sizes. 130
B.5 Kernel time of individual processing steps for varying grid granularities. 131
B.6 Contribution of individual processing steps for *coarse* grid configuration. 132
B.7 Contribution of individual processing steps for *medium* grid configuration. 133
B.8 Contribution of individual processing steps for *fine* grid configuration. . 134
B.9 Runtime for muon track sample for *coarse* grid configuration. 135
B.10 Runtime for muon track sample for *medium* grid configuration. 135
B.11 Runtime for muon track sample for *fine* grid configuration. 136
B.12 Runtime for muon track sample for *super fine* grid configuration. 136
B.13 Kernel time for processing multiple events concurrently. 137
B.14 Transferred data volume and bandwidth for processing multiple events. 137
B.15 Contribution of individual processing steps for *small* events. 138
B.16 Contribution of individual processing steps for *large* events. 139

Appendix E

List of Tables

1.1 Sub-detectors of the CMS tracker with their abbreviations and comprised layers [44, 169]. 14

2.1 Configuration of the CMS seeding in the iterative tracking steps [162, 170]. 29

3.1 Mapping of OpenCL entities to CPU and GPU components [150]. . . . 42

9.1 Expensive pair building operations. 92
9.2 Expensive triplet prediction operations. 97
9.3 Expensive triplet filtering operations. 100

10.1 CPU and GPU used for runtime evaluations. 102
10.2 Cutoff values for the $d\phi$, $d\theta$ and d_0 filter criteria. 107
10.3 Number of combinations processed by CMSSW and different configurations of OpenCL-based triplet finding. 111
10.4 Configuration of the grid data structure and resulting usage of *local* memory in grid building and pair generation kernels. 113

C.1 Parameters for the grid data structure. 141
C.2 Parameters for event data loading. Parameters marked with * are only valid for simulated data. 141
C.3 Parameters to define a layer triplet for triplet finding. All parameters are per layer combination. 142
C.4 Execution parameters for triplet finding. 142

List of Algorithms

2.1 CMSSW pair finding algorithm for the barrel region. 31
2.2 CMSSW triplet finding algorithm for the barrel region. 32
6.1 Data flow of OpenCL triplet finding algorithm. 65
8.1 Ex-situ algorithm for grid data structure construction. 78
9.1 Generic two-pass algorithm with oracle. 82
9.2 Prefix sum kernel. 85
9.3 Recursive prefix sum computation. 86
9.4 Grid-based pair building *count* kernel. 88
9.5 Prediction of z-ϕ-range in pair building *count* kernel. 91
9.6 Angle-based prediction of the z-range. 93
9.7 Branch-less min/max determination. 94
9.8 Angle-based prediction of the ϕ-range. 94
9.9 Extrapolation-based prediction the z-range. 95
9.10 Extrapolation-based prediction of the ϕ-range. 97
9.11 Triplet prediction *count*/*store* kernel. 98
9.12 Triplet filtering *count* kernel. 99
9.13 Hamming weight calculation [109]. 100

Appendix F

Acronyms

ALICE A Large Ion Collider Experiment. pp. 11, 39, 48, 51, 52, 143

API Application Programming Interface. pp. 39, 40

ASIC Application-specific Integrated Circuit. pp. 14, 23

ATLAS A Toroidal LHC Apparatus. pp. 11, 19

AVX Advanced Vector Extensions. pp. 37

CA Cellular Automaton. pp. 1, 2, 35, 39, 47–53, 57, 63–65, 67, 81, 121, 143, 144

CERN Organisation Européenne pour la Recherche Nucléaire. pp. iii, 1, 9, 19, 20

CMS Compact Muon Solenoid. pp. iii, iv, 1, 2, 9, 11–26, 29, 33, 37, 48, 52, 53, 55, 67, 73–75, 77, 81, 87, 95, 101, 107, 109, 121, 122, 143, 144, 147

CMSSW CMS Software Framework. pp. vii, 21, 22, 27, 30–32, 36, 38, 53, 55, 57, 63, 69, 95, 103, 111, 112, 118, 143, 145, 147, 149

CPU Central Processing Unit. pp. iii, iv, vii, 23, 35–37, 39–42, 44, 53, 60, 63, 81, 102, 103, 113–115, 117, 118, 121, 122, 129, 130, 132–134, 138, 139, 142–144, 147

CSC Cathode Strip Chamber. pp. 16, 17

CTF Combinatorial Track Finder. pp. 26

DAQ Data Acquisition. pp. 21, 23, 24, 33

DC Data Centre. pp. 19

DT Drift Tube. pp. 16, 24

EB ECAL Barrel. pp. 15

ECAL Electromagnetic Calorimeter. pp. 15, 24, 26

EDM Event Data Model. pp. 21

EE ECAL Endcap. pp. 15

FED Front End Driver. pp. 15, 23, 24

FPGA Field-programmable Gate Array. pp. 23

FPU Floating Point Unit. pp. 37

GPGPU General Purpose Graphical Processing Unit. pp. 35, 37, 39, 43, 51, 75, 83, 142

GPU Graphical Processing Unit. pp. iv, vii, 35, 36, 38–44, 53, 56–60, 63, 74, 81, 84, 102, 103, 113–118, 121, 122, 129, 130, 132–134, 138, 139, 143, 144, 147

HB HCAL Barrel. pp. 16

HCAL Hadronic Calorimeter. pp. 16, 26

HE HCAL Endcap. pp. 16

HEP High-Energy Physics. pp. 21, 27, 48

HF Forward HCAL. pp. 16, 18

HI Heavy Ion. pp. 11, 52

HLT High Level Trigger. pp. 21, 23, 24, 39, 48, 51

HO Outer HCAL. pp. 16

IOV Interval of Validity. pp. 24

IP Interaction Point. pp. 12, 25, 26, 69

IS Instruction Set. pp. 37

LEAR Low Energy Antiproton Ring. pp. 9

LEP Large Electron-Positron Collider. pp. 9

LHC Large Hadron Collider. pp. iii, vii, 1, 2, 9–11, 13, 19, 48, 52, 121, 143

LHCb LHC-beauty. pp. 11

LHCOPN LHC Optical Private Network. pp. 19

LIP Longitudinal Impact Parameter. pp. 142

MBR Minimum Bounding Rectangle. pp. 58, 59, 143

MC Monte Carlo. pp. 20, 24, 33, 34

MIMD Multiple Instruction Multiple Data. pp. 37, 44

MoU Memorandum of Understanding. pp. 20

MWPC Multiwire Proportional Chamber. pp. 51, 52, 143

OpenCL Open Computing Language. pp. iv, vii, 35, 38, 40–44, 53, 56–60, 63–65, 73–75, 81, 86, 99–102, 107, 108, 111–113, 115, 116, 118, 121, 122, 143–145, 147, 149

PXB Pixel Barrel. pp. 14, 76, 104, 106–110, 144

PXF Pixel Forward. pp. 14

QBP Queen Bee Problem. pp. 50, 51

QCD Quantum Chromodynamics. pp. 101, 104, 108–110, 122, 125, 126, 144, 145

RPC Resistive Plate Chamber. pp. 17

SDK Software Development Kit. pp. 35, 39, 86, 102, 112

SIMD Single Instruction Multiple Data. pp. 37, 38, 41, 115

SLC Scientific Linux CERN. pp. 21, 102

SMP Symmetric Multiprocessing. pp. 36

SMX Streaming Multiprocessor. pp. 39, 40, 42–44, 84, 86

SPMD Single Program Multiple Data. pp. 41

SSE Streaming SIMD Extensions. pp. 37

TBB Threading Building Blocks. pp. 63

TEC Tracker End Cap. pp. 14

TIB Tracker Inner Barrel. pp. 14, 76, 104, 107–110, 144

TID Tracker Inner Disk. pp. 14

TIP Transverse Impact Parameter. pp. 25, 26, 69–71, 90, 142, 144

TOB Tracker Outer Barrel. pp. 14, 107, 109, 110

TPC Time Projection Chamber. pp. 51, 52, 143

USM Unified Shader Model. pp. 39

WLCG Worldwide LHC Computing Grid. pp. iv, 19–21, 35, 40, 53, 63, 121, 143

Appendix G

Bibliography

[1] I. Abt et al. „CATS: a cellular automaton for tracking in silicon for the HERA-B vertex detector“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 489.1-3 (Aug. 2002), pp. 389–405. ISSN: 0168-9002.

[2] D. Acosta et al., eds. *CMS Physics: Technical Design Report Volume 1: Detector Performance and Software.* Technical Design Report CMS. CERN-LHCC-2006-001; CMS-TDR-8-1. Geneva, CH: CERN, 2006. ISBN: 9290832681.

[3] Advanced Micro Devices, Inc. *AMD Showcases Wide-Ranging Motherboard Support For Dual-Core AMD64 Processors.* press release. May 2005.

[4] L. Agostino et al. „Commissioning of the CMS High Level Trigger“. In: *Journal of Instrumentation* 4.10 (2009), P10005.

[5] W. Akman et al. „Geometric computing and uniform grid technique“. In: *Computer-Aided Design* 21.7 (Sept. 1989), pp. 410–420. ISSN: 0010-4485.

[6] ALICE Collaboration. „ALICE HLT High Speed Tracking on GPU“. In: *IEEE Transactions on Nuclear Science* 58.4 (Aug. 2011), pp. 1845–1851. ISSN: 0018-9499.

[7] J. Allison et al. „Geant4 developments and applications“. In: *Nuclear Science, IEEE Transactions on* 53.1 (Feb. 2006), pp. 270–278. ISSN: 0018-9499.

[8] ALPHA Collaboration. „Trapped antihydrogen“. In: *Nature* 468 (Dec. 2010), pp. 673–676.

[9] J. Alwall et al. „MadGraph 5: going beyond“. English. In: *Journal of High Energy Physics* 2011.6 (June 2011), pp. 1–40.

[10] G. M. Amdahl. „Validity of the single processor approach to achieving large scale computing capabilities“. In: *Proceedings of the AFIPS Spring Joint Computer Conference.* AFIPS ’67 (Spring). Atlantic City, New Jersey: ACM, 1967, pp. 483–485.

[11] Q. An et al. „Elucidation of the dynamics for hot-spot initiation at nonuniform interfaces of highly shocked materials“. In: *Phys. Rev. B* 84 (22 Dec. 2011), p. 220101.

[12] Apple Inc. *Apple Previews Mac OS X Snow Leopard to Developers.* press release. June 2008.

[13] Apple Inc. *OpenCL Parallel Prefix Sum (aka Scan) Example.* OpenCl examples. Sept. 2009.

[14] ATLAS Collaboration. „Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC“. In: *Physics Letters B* 716.1 (Sept. 2012), pp. 1–29. ISSN: 0370-2693.

[15] D. A. Bader, B. M. Moret, and P. Sanders. „Algorithm Engineering for Parallel Computation“. In: *Experimental Algorithmics.* Ed. by R. Fleischer, B. Moret, and E. M. Schmidt. Vol. 2547. Lecture Notes in Computer Science. Berlin: Springer, 2002, pp. 1–23.

[16] M. Bähr et al. „Herwig++ physics and manual“. English. In: *The European Physical Journal C* 58.4 (Dec. 2008), pp. 639–707. ISSN: 1434-6044.

[17] D. Barney. *CMS slice.* CMS 5581-v1. Sept. 2011.

[18] G. Baur et al. „Production of antihydrogen“. In: *Physics Letters B* 368.3 (1996), pp. 251–258. ISSN: 0370-2693.

[19] G. L. Bayatyan et al. *CMS computing: Technical Design Report.* Technical Design Report CMS. Geneva, CH: CERN, 2005.

[20] R. Bayer and E. McCreight. „Organization and maintenance of large ordered indexes“. In: *Acta Informatica* 1.3 (1972), pp. 173–189. ISSN: 0001-5903.

[21] S. Beauceron. *The CMS High Level Trigger.* conference report CERN-CMS-CR-2012-355. Geneva: CERN, Nov. 2012.

[22] J. L. Bentley. „K-d trees for semidynamic point sets“. In: *Proceedings of the 6th Annual Symposium on Computational Geometry.* Berkley, California, USA: ACM, 1990, pp. 187–197. ISBN: 0-89791-362-0.

[23] J. L. Bentley. „Multidimensional binary search trees used for associative searching“. In: *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782.

[24] M. Berg et al. *Computational Geometry.* Berlin: Springer, 1997.

[25] T. Berners-Lee. „Information Management: A Proposal“. project proposal. Mar. 1989.

[26] I. Bird. „Computing for the Large Hadron Collider“. In: *Annual Review of Nuclear and Particle Science* 61.1 (Nov. 2011), pp. 99–118.

[27] G. Blelloch. „Scans as primitive parallel operations“. In: *IEEE Transactions on Computers* 38.11 (Nov. 1989), pp. 1526–1538. ISSN: 0018-9340.

[28] M. Blum et al. „Time bounds for selection“. In: *Journal of Computer and System Sciences* 7.4 (Aug. 1973), pp. 448–461. ISSN: 0022-0000.

[29] S. Borkar. „Thousand core chips: a technology perspective“. In: *Proceedings of the 44th annual Design Automation Conference.* San Diego, California: ACM, 2007, pp. 746–749. ISBN: 978-1-59593-627-1.

[30] R. Brun and F. Rademakers. „ROOT - An object oriented data analysis framework“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 389.1-2 (Apr. 1997), pp. 81–86. ISSN: 0168-9002.

[31] O. S. Brüning et al., eds. *LHC Design Report.* Vol. 1. Geneva, CH: CERN, 2004.

[32] B. Cabral, N. Cam, and J. Foran. „Accelerated volume rendering and tomographic reconstruction using texture mapping hardware“. In: *Proceedings of the 1994 Symposium on Volume Visualization.* VVS ’94. Tysons Corner, Virginia, USA: ACM, 1994, pp. 91–98. ISBN: 0-89791-741-3.

[33] D. A. Carr, C. Paszko, and D. Kolva. *SeqNFinD: A GPU Accelerated Sequence Analysis Toolset Facilitates Bioinformatics.* Nature Methods - Application Notes. Aug. 2011.

[34] D. Cederman and P. Tsigas. „GPU-Quicksort: A Practical Quicksort Algorithm for Graphics Processors“. In: *Journal of Experimental Algorithms* 14 (Jan. 2010), 4:1.4–4:1.24. ISSN: 1084-6654.

[35] CERN. *CERN awards major contract for computer infrastructure hosting to Wigner Research Centre for Physics in Hungary.* press release. May 2012.

[36] CERN. *CERN Data Centre passes 100 petabytes.* press release. Feb. 2013.

[37] CERN. *The first LHC protons run ends with new milestone.* press release. Dec. 2012.

[38] CERN Communication Group. *LHC - The Guide.* Brochure. CERN-Brochure-2009-003-Eng. Geneva, CH, Feb. 2009.

[39] Y. Chao. *Minimum-Bias and Underlying Event Studies at CMS.* preprint 0810.4819. arXiv, 2008.

[40] C. Cheshkov. „Fast Hough-transform track reconstruction for the ALICE TPC“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 566.1 (Oct. 2006), pp. 35–39. ISSN: 0168-9002.

[41] CMS Collaboration. *Measurement of CMS Luminosity.* Tech. rep. CMS-PAS-EWK-10-004. Geneva, CH: CERN, 2010.

[42] CMS Collaboration. *Measurement of the ttbar production cross section in the all-jet final state in pp collisions at sqrt(s) = 7 TeV.* preprint 1302.0508. CERN-PH-EP-2012-358. arXiv, Feb. 2013.

[43] CMS Collaboration. „Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC“. In: *Physics Letters B* 716.1 (Sept. 2012), pp. 30–61. ISSN: 0370-2693.

[44] CMS Collaboration. „The CMS experiment at the CERN LHC. The Compact Muon Solenoid experiment“. In: *Journal of Instrumentation* 3 (Aug. 2008). Also published by CERN Geneva in 2010, S08004.

[45] J. Cornwall. „Efficent multiple pass, multiple output algorithms on the GPU“. In: *Proceedings of the 2nd IEE European Conference on Visual Media Production, 2005.* New York, NY, USA: IEEE, 2005, pp. 255–264.

[46] N. Cuntz et al. „GPU-based Dynamic Flow Visualization for Climate Research Applications“. In: *Simulation und Visualisierung 2007.* Erlangen: SCS Publishing House e.V., 2007, pp. 371–384.

[47] L. Dagum and R. Menon. „OpenMP: an industry standard API for shared-memory programming“. In: *Computational Science and Engineering* 5.1 (Jan. 1998), pp. 46–55. ISSN: 1070-9924.

[48] P. Du et al. „From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming“. In: *Parallel Computing* 38.8 (Aug. 2012), pp. 391–407. ISSN: 0167-8191.

[49] C. Eck et al., eds. *LHC computing Grid: Technical Design Report.* Technical Design Report LCG. Geneva, CH: CERN, 2005.

[50] R. Eckhardt. „Stan Ulam, John von Neumann, and the Monte Carlo Method“. In: *Los Alamos Science* special issue (1987), pp. 131–143.

[51] L. R. Evans and P. Bryant. „LHC Machine“. In: *Journal of Instrumentation* 3 (Aug. 2008). This report is an abridged version of the LHC Design Report (CERN-2004-003), S08001.

[52] J. Fang, A. Varbanescu, and H. Sips. „A Comprehensive Performance Comparison of CUDA and OpenCL“. In: *Proceedings of the International Conference on Parallel Processing.* New York, NY, USA: IEEE, 2011, pp. 216–225.

[53] R. Finkel and J. Bentley. „Quad trees a data structure for retrieval on composite keys“. In: *Acta Informatica* 4.1 (1974), pp. 1–9. ISSN: 0001-5903.

[54] M. Flynn. „Some Computer Organizations and Their Effectiveness“. In: *IEEE Transactions on Computers* C-21.9 (Sept. 1972), pp. 948–960. ISSN: 0018-9340.

[55] I. Foster, C. Kesselman, and S. Tuecke. „The Anatomy of the Grid: Enabling Scalable Virtual Organizations“. In: *International Journal of High Performance Computing Applications* 15.3 (2001), pp. 200–222.

[56] C. Foudas et al. „The CMS tracker readout front end driver“. In: *IEEE Transactions on Nuclear Science* 52.6 (Dec. 2005), pp. 2836–2840. ISSN: 0018-9499.

[57] W. R. Franklin. „Adaptive Grids For Geometric Operations“. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 21.2 (Oct. 1984), pp. 160–167.

[58] K. A. Frenkel. „Evaluating two massively parallel machines“. In: *Communications of the ACM* 29.8 (Aug. 1986), pp. 752–758. ISSN: 0001-0782.

[59] R. Frühwirth. „Application of Kalman filtering to track and vertex fitting“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 262.2-3 (Dec. 1987), pp. 444–450. ISSN: 0168-9002.

[60] R. Frühwirth, A. Strandlie, and W. Waltenberger. „Helix fitting by an extended Riemann fit“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 490.1-2 (Sept. 2002), pp. 366–378. ISSN: 0168-9002.

[61] R. Frühwirth and M. Regler, eds. *Data analysis techniques for high-energy physics.* 2. ed. Cambridge monographs on particle physics, nuclear physics, and cosmology 11. Cambridge, UK: Cambridge University Press, 2000. ISBN: 0-521-63219-6; 0-521-63548-9.

[62] R. Frühwirth et al. „A review of fast circle and helix fitting“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 502.2-3 (Apr. 2003), pp. 705–707. ISSN: 0168-9002.

[63] J. Fung and S. Mann. „OpenVIDIA: parallel GPU computer vision“. In: *Proceedings of the 13th Annual ACM International Conference on Multimedia.* MULTIMEDIA ’05. Hilton, Singapore: ACM, 2005, pp. 849–852. ISBN: 1-59593-044-2.

[64] W. W. L. Fung et al. „Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow“. In: *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture.* Washington, DC, USA: IEEE Computer Society, 2007, pp. 407–420. ISBN: 0-7695-3047-8.

[65] L. Georghiou. „Global cooperation in research“. In: *Research Policy* 27.6 (Sept. 1998), pp. 611–626. ISSN: 0048-7333.

[66] P. Gepner and M. Kowalik. „Multi-Core Processors: New Way to Achieve High System Performance“. In: *Proceedings of the International Symposium on Parallel Computing in Electrical Engineering.* New York, NY, USA: IEEE, 2006, pp. 9–13.

[67] D. Giordano and G. Sguazzoni. „CMS reconstruction improvements for the tracking in large pile-up events“. In: *Journal of Physics: Conference Series* 396.2 (2012), p. 022044.

[68] A. Glazov et al. „Filtering tracks in discrete detectors using a cellular automaton“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 329.1-2 (May 1993), pp. 262–268. ISSN: 0168-9002.

[69] S. Gorbunov et al. „Fast SIMDized Kalman filter based track fit“. In: *Computer Physics Communications* 178.5 (Mar. 2008), pp. 374–383. ISSN: 0010-4655.

[70] S. J. Gortler. *Foundations of 3D computer graphics.* Cambridge, MA, USA: MIT Press, 2012. ISBN: 978-0-262-01735-0; 0-262-01735-0.

[71] D. Griffiths. *Introduction to Elementary Particles.* Weinheim, DE: John Wiley & Sons, Sept. 2008.

[72] K. Grimm et al. „Methods to quantify the performance of the primary vertex reconstruction in the ATLAS experiment under high luminosity conditions“. In: *Journal of Physics: Conference Series* 396.2 (2012), p. 02204. ISSN: 1742-6596.

[73] J. L. Gustafson. „Reevaluating Amdahl's law“. In: *Communications of the ACM* 31.5 (May 1988), pp. 532–533. ISSN: 0001-0782.

[74] A. Guttman. „R-trees: a dynamic index structure for spatial searching“. In: *ACM SIGMOD Record* 14.2 (June 1984), pp. 47–57. ISSN: 0163-5808.

[75] R. W. Hamming. „Error detecting and error correcting codes“. In: *The Bell System Technical Journal* 29 (1950), pp. 147–160. ISSN: 0005-8580.

[76] T. D. Han and T. S. Abdelrahman. „Reducing branch divergence in GPU programs“. In: *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*. GPGPU-4. Newport Beach, California: ACM, 2011, 3:1–3:8. ISBN: 978-1-4503-0569-3.

[77] M. Hansroul, H. Jeremie, and D. Savard. „Fast circle fit with the conformal mapping method“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 270.2-3 (July 1988), pp. 498–501. ISSN: 0168-9002.

[78] M. J. Harris et al. „Physically-based visual simulation on graphics hardware“. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. Saarbrucken, Germany: Eurographics Association, 2002, pp. 109–118. ISBN: 1-58113-580-7.

[79] T. Hauth, V. Innocente, and D. Piparo. „Development and Evaluation of Vectorised and Multi-Core Event Reconstruction Algorithms within the CMS Software Framework“. In: *Journal of Physics: Conference Series* 396.5 (2012), p. 052065.

[80] T. Hauth et al. „Parallel track reconstruction in CMS using the cellular automaton approach“. In: *Proceedings of the 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013)*. in preparation. 2013.

[81] J. Hegeman. *Public CMS Luminosity Information*. Feb. 2013.

[82] B. Hegner, P. Mato, and D. Piparo. „Evolving LHC Data Processing Frameworks for Efficient Exploitation of New CPU Architectures“. In: *Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. to appear. New York, NY, USA: IEEE, 2012.

[83] V. L. Highland. „Some practical remarks on multiple scattering“. In: *Nuclear Instruments and Methods* 129.2 (Nov. 1975), pp. 497–499. ISSN: 0029-554X.

[84] W. D. Hillis and J. G. L. Steele. „Data parallel algorithms“. In: *Communications of the ACM* 29.12 (Dec. 1986), pp. 1170–1183. ISSN: 0001-0782.

[85] V. Hindriksen. „NVIDIA's Industry-Leading “Support” For OpenCL“. In: *Stream Computing Performance Engineers* (Sept. 2012).

[86] A. Holzner. *78 reconstructed vertices in event from high-pileup run 198609*. CMS-PHO-EVENTS-2012-006. Sept. 2012.

[87] IBM. *IBM Launches World's Most Powerful Server: "Regatta" Transforms the Economics of UNIX Servers at Half the Price of Competition*. press release. Oct. 2001.

[88] Intel Cooperation. *Intel Announces New Pentium(R) III Brand for Next Generation Processors.* press release. Jan. 1999.

[89] Intel Cooperation. *Intel Architecture Instruction Set Extensions Programming Reference.* Tech. rep. 319433-014. Intel Cooperation, Aug. 2012.

[90] Intel Cooperation. *Intel Brings 'Eye Candy' to Masses with Newest Laptop, PC Chips.* press release. Jan. 2011.

[91] Intel Cooperation. *Intel Core i7-3960X Processor Extreme Edition (15M Cache, up to 3.90 GHz).* specification. Nov. 2011.

[92] Intel Cooperation. *Intel Releases MMX (TM) Technology Details to Software Community to Drive New Multimedia, Game and Internet Applications.* press release. Mar. 1996.

[93] C. D. Jones et al. „The New CMS Event Data Model and Framework“. In: *Proceedings of the 15th International Conference on Computing in High Energy and Nuclear Physics.* Ed. by S. Banerjee. Trieste, IT: ICTP-OEA, 2006, p. 242.

[94] C. Jones. „Study of a Fine Grained Threaded Framework Design“. In: *Journal of Physics: Conference Series* 396.2 (2012), p. 022027.

[95] M. Joselli et al. „A Neighborhood Grid Data Structure for Massive 3D Crowd Simulation on GPU“. In: *Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on.* New York, NY, USA: IEEE, 2009, pp. 121–131.

[96] R. E. Kalman. „A New Approach to Linear Filtering and Prediction Problems“. In: *Journal of Fluids Engineering* 82.1 (Dec. 1960), pp. 35–45.

[97] R. E. Kalman and R. S. Bucy. „New Results in Linear Filtering and Prediction Theory“. In: *Jou* 83.1 (Mar. 1961), pp. 95–108.

[98] J. Kalojanov and P. Slusallek. „A parallel algorithm for construction of uniform grids“. In: *Proceedings of the Conference on High Performance Graphics 2009.* New Orleans, Louisiana: ACM, 2009, pp. 23–28. ISBN: 978-1-60558-603-8.

[99] A. H. Karp and H. P. Flatt. „Measuring parallel processor performance“. In: *Communications of the ACM* 33.5 (May 1990), pp. 539–543. ISSN: 0001-0782.

[100] H. Kästli et al. „Design and performance of the CMS pixel detector readout chip“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 565.1 (Sept. 2006), pp. 188–194. ISSN: 0168-9002.

[101] Khronos Group. *The Khronos Group Releases OpenCL 1.0 Specification.* press release. Dec. 2008.

[102] Khronos Group. *The Khronos Group Releases OpenCL 1.2 Specification.* press release. Nov. 2011.

[103] E. Kilgariff and R. Fernando. „The GeForce 6 series GPU architecture“. In: *ACM SIGGRAPH 2005 Courses.* SIGGRAPH '05. Los Angeles, California: ACM, 2005.

[104] J. Kim and B. Nam. „Parallel multi-dimensional range query processing with R-trees on GPU“. In: *Journal of Parallel and Distributed Computing* (2013). in press. ISSN: 0743-7315.

[105] W. Kim and M. Voss. „Multicore Desktop Programming with Intel Threading Building Blocks“. In: *IEEE Software* 28.1 (Jan. 2011), pp. 23–31. ISSN: 0740-7459.

[106] D. Kirk and W.-m. W. Hwu. *Programming massively parallel processors : a hands-on approach.* Amsterdam: Elsevier, Morgan Kaufmann, 2010. ISBN: 978-0-12-381472-2.

[107] I. Kisel. „Event reconstruction in the CBM experiment“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 566.1 (Oct. 2006), pp. 85–88. ISSN: 0168-9002.

[108] K. Klein. „Lessons learned during CMS tracker end cap construction“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 579.2 (Feb. 2007), pp. 731–735. ISSN: 0168-9002.

[109] D. E. Knuth. *The Art of Computer Programming: Bitwise Tricks & Techniques; Binary Decision Diagrams.* Vol. 4, Fascicle 1. Upper Saddle River, NJ, USA: Addison-Wesley, 2009.

[110] M. Konecki. „Meeting on Pixel Triplet Seeding“. private communication. Sept. 2012.

[111] S. de Konink and R. Baca. *R-tree example.* wikimedia commons. Apr. 2010.

[112] E. Kozdrowicki and D. Theis. „Second Generation of Vector Supercomputers“. In: *Computer* 13.11 (1980), pp. 71–83. ISSN: 0018-9162.

[113] N. V. Krasnikov and V. A. Matveev. „Physics at the large hadron collider“. In: *Physics of Particles and Nuclei* 28.5 (Sept. 1997), pp. 441–470.

[114] G. Lamanna, G. Collazuol, and M. Sozzi. „GPUs for fast triggering and pattern matching at the CERN experiment NA62“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 628.1 (Feb. 2011), pp. 457–460. ISSN: 0168-9002.

[115] M. Lapka. *CMS General Brochure.* brochure. CMS Document 4263-v1. Mar. 2011.

[116] C. Lefèvre. *The CERN accelerator complex. Complexe des accélérateurs du CERN.* CERN-DI-0812015. Dec. 2008.

[117] N. Leischner, V. Osipov, and P. Sanders. „GPU sample sort“. In: *Proceedings of the 2010 IEEE International Symposium on Parallel Distributed Processing.* New York, NY, USA: IEEE, 2010, pp. 1–10.

[118] D. Luebke and G. Humphreys. „How GPUs Work“. In: *Computer* 40.2 (Feb. 2007), pp. 96–100. ISSN: 0018-9162.

[119] L. Luo, M. D. F. Wong, and L. Leong. „Parallel implementation of R-trees on the GPU“. In: *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC).* New York, NY, USA: IEEE, 2012, pp. 353–358.

[120] W. A. Maniatty, B. Szymanski, and T. Caraco. „Parallel Computing With Generalized Cellular Automata“. In: *Parallel and Distributed Computing Practices* 1.1 (1998), pp. 31–50.

[121] Y. Manolopoulos et al. *R-Trees: Theory and Applications*. Berlin: Springer, 2006.

[122] P. Martin et al. „Algorithmic strategies for optimizing the parallel reduction primitive in CUDA“. In: *High Performance Computing and Simulation (HPCS), 2012 International Conference on*. New York, NY, USA: IEEE, 2012, pp. 511–519.

[123] J. N. Marx and D. R. Nygren. „The Time Projection Chamber“. In: *Physics Today* 31.10 (Oct. 1978), pp. 46–53.

[124] M.Górski. „10 years of WLCG“. In: *Proceedings of the Cracow Epiphany Conference*. Jan. 2013.

[125] C. Moler. „Matrix Computation on Distributed Memory Multiprocessors“. In: *Proceedings of the First Conference on Hypercube Multiprocessors*. Philadelphia, PA, USA: Society for Industrial & Applied Mathematics, 1985, pp. 181–195.

[126] G. E. Moore. „Cramming more components onto integrated circuits“. In: *Electronics* 38.8 (Apr. 1965), pp. 4–7.

[127] S. Morein et al. „Graphics processing architecture employing a unified shader“. 6897871 (Ontario, CA). May 24, 2005.

[128] A. Munshi, ed. *The OpenCL Specification*. Beaverton, OR, USA: Khronos Group, Nov. 2012.

[129] J. von Neumann. *Theory of Self-reproducing Automata*. Ed. by A. W. Burks. Urbana, IL, USA: University of Illinois Press, 1966.

[130] M. E. J. Newman and G. T. Barkema. *Monte Carlo Methods in Statistical Physics*. Oxford, UK: Clarendon Press, 2004. ISBN: 0-19-851797-1; 0-19-851796-3.

[131] J. Nickolls et al. „Scalable Parallel Programming with CUDA“. In: *Queue* 6.2 (Mar. 2008), pp. 40–53. ISSN: 1542-7730.

[132] A. Nowak. „Intel OpenCL Compiler“. private communication. Mar. 2013.

[133] D. Nuzman, I. Rosen, and A. Zaks. „Auto-vectorization of interleaved data for SIMD“. In: *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’06. Ottawa, Ontario, Canada: ACM, 2006, pp. 132–143. ISBN: 1-59593-320-4.

[134] NVIDIA Cooperation. *NVIDIA CUDA Toolkit v5.0 Release Notes*. release notes. Oct. 2012.

[135] NVIDIA Cooperation. *NVIDIA Introduces GeForce GTX TITAN: DNA of the World’s Fastest Supercomputer, Powered by World’s Fastest GPU*. press release. Feb. 2013.

[136] NVIDIA Cooperation. *NVIDIA OpenCL Best Practices Guide*. white paper. Aug. 2009.

[137] NVIDIA Cooperation. *NVIDIA Tesla GPU Computing Processor Ushers In the Era of Personal Supercomputing.* press release. June 2007.

[138] NVIDIA Cooperation. *NVIDIA's Next Generation CUDA Compute Architecture: Kepler TM GK110.* white paper. Santa Clara, CA, USA, 2012.

[139] J. D. Owens et al. „A Survey of General-Purpose Computation on Graphics Hardware“. In: *Computer Graphics Forum* 26.1 (Mar. 2007), pp. 80–113. ISSN: 1467-8659.

[140] S. Pennycook et al. „An investigation of the performance portability of OpenCL“. In: *Journal of Parallel and Distributed Computing* (2012). in press. ISSN: 0743-7315.

[141] H. Perez-Ponce et al. „Implementing Geant4 on GPU for medical applications“. In: *Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference.* New York, NY, USA: IEEE, 2011, pp. 2703–2707.

[142] D. Pestre and J. Krige. „Some Thoughts on the Early History of CERN“. In: *Big Science: The Growth of Large-Scale Research.* Ed. by P. L. Galison. Stanford, CA, USA: Stanford University Press, 1992. Chap. 3, pp. 78–99.

[143] J. Pike. „Text compression using a 4 bit coding scheme“. In: *The Computer Journal* 24.4 (1981), pp. 324–330.

[144] M. Pioppi. *Iterative Tracking.* internal note CMS-IN-2007-065. Geneva, CH: CERN, Nov. 2007.

[145] R. L. Rardin and R. Uzsoy. „Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial.“ In: *Journal of Heuristics* 7.3 (2001), pp. 261–304.

[146] I. Reid. *Update on kd-tree triplet generator.* slides. PH-DPG Tracking Meeting. Aug. 2012.

[147] T. G. Rokicki. *An Algorithm for Compressing Space and Time.* article. Dr. Dobbs. Apr. 2006.

[148] K. Rose. „Deterministic annealing for clustering, compression, classification, regression, and related optimization problems“. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2210–2239. ISSN: 0018-9219.

[149] H. Samet. „Spatial data structures“. In: *Modern Database Systems: The Object Model, Interoperability and Beyond.* Ed. by W. Kim. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 361–385. ISBN: 0-201-59098-0.

[150] J. van der Sanden. „Evaluating the Performance and Portability of OpenCL“. master thesis. Eindhoven, NL: Eindhoven University of Technology, Aug. 2011.

[151] P. Sanders. „Algorithm Engineering - An Attempt at a Definition“. In: *Efficient Algorithms.* Ed. by S. Albers, H. Alt, and S. Näher. Vol. 5760. Lecture Notes in Computer Science. Berlin: Springer Berlin Heidelberg, 2009, pp. 321–340.

[152] P. Sanders and S. Winkel. „Super Scalar Sample Sort“. In: *Algorithms - ESA 2004.* Ed. by S. Albers and T. Radzik. Vol. 3221. Lecture Notes in Computer Science. Berlin: Springer, 2004, pp. 784–796. ISBN: 978-3-540-23025-0.

[153] A. dos Santos et al. „kD-Tree Traversal Implementations for Ray Tracing on Massive Multiprocessors: A Comparative Study“. In: *Proceedings of the 21st International Symposium on Computer Architecture and High Performance Computing.* New York, NY, USA: IEEE, 2009, pp. 41–48.

[154] A. Satpathy. „Overview and status of the CMS silicon strip tracker“. In: *Journal of Physics: Conference Series* 110.9 (2008), p. 092026.

[155] F. Sauli. *Principles of operation of multiwire proportional and drift chambers.* Tech. rep. CERN-77-09. Geneva, CH: CERN, 1977.

[156] A. Scheurer. „German Contributions to the CMS Computing Infrastructure“. In: *Journal of Physics: Conference Series* 219.6 (2010), p. 062064.

[157] O. Seiskari, J. Kommeri, and T. Niemi. *GPU in Physics Computation: Case Geant4 Navigation.* preprint 1209.5235. arXiv, Sept. 2012.

[158] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. „The R+-Tree: A Dynamic Index for Multi-Dimensional Objects“. In: *Proceedings of the 13th International Conference on Very Large Data Bases.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, pp. 507–518. ISBN: 0-934613-46-X.

[159] S. Sengupta et al. „Scan primitives for GPU computing“. In: *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware.* San Diego, California: Eurographics Association, 2007, pp. 97–106. ISBN: 978-1-59593-625-7.

[160] M. H. Seymour. „Searches for new particles using cone and cluster jet algorithms: a comparative study“. English. In: *Zeitschrift für Physik C Particles and Fields* 62.1 (1994), pp. 127–138. ISSN: 0170-9739.

[161] G. Sguazzoni, K. Stenson, and G. Cerati. *Tracking in 2013: how to get ready for 2015.* slides. Special CMS Week Tracking POG Meeting. Dec. 2012.

[162] G. Sguazzoni et al. *Description and Performance of the CMS Track and Primary Vertex Reconstruction.* Analysis Note CMS AN-2011/172. Draft. Geneva, CH: CMS Collaboration, May 2011.

[163] C. A. Shaffer and H. Samet. „Optimal quadtree construction algorithms“. In: *Computer Vision, Graphics, and Image Processing* 37.3 (Mar. 1987), pp. 402–419. ISSN: 0734-189X.

[164] J. Shen et al. „Performance Gaps between OpenMP and OpenCL for Multi-core CPUs“. In: *Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW).* New York, NY, USA: IEEE, 2012, pp. 116–125.

[165] J. Shin, M. Hall, and J. Chame. „Superword-level parallelism in the presence of control flow“. In: *Proceedings of the International Symposium on Code Generation and Optimization.* New York, NY, USA: IEEE, Mar. 2005, pp. 165–175.

[166] A. R. Smith. „Two-dimensional formal languages and pattern recognition by cellular automata“. In: *Proceedings of the 12th Annual Symposium on Switching and Automata Theory*. New York, NY, USA: IEEE Computer Society, 1971, pp. 144–152.

[167] W. Smith et al., eds. *CMS TriDAS project: Technical Design Report, Volume 1: The Trigger Systems*. Technical Design Report CMS CERN-LHCC-2000-038; CMS-TDR-6-1. Geneva, CH: CERN, 2000.

[168] T. Söstrand et al. „High-energy-physics event generation with Pythia 6.1“. In: *Computer Physics Communications* 135.2 (Apr. 2001), pp. 238–259. ISSN: 0010-4655.

[169] D. Sprenger. „Track-based Alignment of a CMS Tracker Endcap“. diploma thesis. Aachen: RWTH Aachen University, July 2008.

[170] K. Stenson. *Iterative Tracking*. CERN. Oct. 2011. URL: `https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideIterativeTracking`.

[171] J. E. Stone, D. Gohara, and G. Shi. „OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems“. In: *Computing in Science and Engineering* 12.3 (2010), pp. 66–73.

[172] A. Strandlie, J. Wroldsen, and R. Frühwirth. „Treatment of multiple scattering with the generalized Riemann sphere track fit“. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 488.1-2 (Aug. 2002), pp. 332–341. ISSN: 0168-9002.

[173] A. Strandlie et al. „Particle tracks fitted on the Riemann sphere“. In: *Computer Physics Communications* 131.1-2 (Sept. 2000), pp. 95–108. ISSN: 0010-4655.

[174] M. Stratmann and T. Worsch. „Leader election in d-dimensional CA in time diam log(diam)“. In: *Future Generation Computer Systems* 18.7 (Aug. 2002), pp. 939–950. ISSN: 0167-739X.

[175] H. Sutter. „The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software“. In: *Dr. Dobb‘s Journal* 30.3 (Mar. 2005), pp. 202–210.

[176] UA1 Collaboration. „Experimental observation of isolated large transverse energy electrons with associated missing energy at $\sqrt{s} = 540GeV$“. In: *Physics Letters B* 122.1 (Feb. 1983), pp. 103–116. ISSN: 0370-2693.

[177] UA2 Collaboration. „Observation of single isolated electrons of high transverse momentum in events with missing transverse energy at the CERN pp collider“. In: *Physics Letters B* 122.5-6 (1983), pp. 476–485. ISSN: 0370-2693.

[178] W. Waltenberger, R. FrÃ¼hwirth, and P. Vanlaer. „Adaptive vertex fitting“. In: *Journal of Physics G: Nuclear and Particle Physics* 34.12 (2007), N343.

[179] R. Wilkinson, B. Hegner, and C. D. Jones. „Usage of the Python programming language in the CMS experiment“. In: *Journal of Physics: Conference Series* 219.4 (2010), p. 042026.

[180] T. Willhalm and N. Popovici. „Putting Intel Threading Building Blocks to Work“. In: *Proceedings of the 1st International Workshop on Multicore Software Engineering*. Leipzig, Germany: ACM, 2008, pp. 3–4. ISBN: 978-1-60558-031-9.

[181] WLCG Collaboration. *Memorandum of Understanding*. CERN-C-RRB-2005-01. Mar. 2011.

[182] E. Yusov and V. Turlapov. „GPU-optimized efficient quad-tree based progressive multiresolution model for interactive large scale terrain rendering“. In: *Proceedings of the 17th International Conference on Computer Graphics and Vision (GraphiCon 07)*. 2007, pp. 23–27.

[183] J. Zhang, S. You, and L. Gruenwald. „Parallel quadtree coding of large-scale raster geospatial data on GPGPUs“. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. Chicago, Illinois: ACM, 2011, pp. 457–460. ISBN: 978-1-4503-1031-4.

[184] K. Zhang et al. „GPU accelerate parallel Odd-Even merge sort: An OpenCL method“. In: *Proceedings of the 15th International Conference on Computer Supported Cooperative Work in Design*. New York, NY, USA: IEEE, 2011, pp. 76–83.

[185] K. Zhou et al. „Real-time KD-tree construction on graphics hardware“. In: *ACM Transactions on Graphics* 27.5 (Dec. 2008), 126:1–126:11. ISSN: 0730-0301.

Acknowledgements

First and foremost I would like to thank Prof. Günter Quast for welcoming me to his research group, giving me the opportunity to work on this fascinating topic and his excellent supervision and counseling. I would like to extend my gratitude to Prof. Peter Sanders for taking over the computer science-related supervision of the work. His invaluable input and motivation helped attaining the results that could be presented in this thesis.

Furthermore, I would like to thank Thomas Hauth for all his efforts. Without his supervision, the fruitful discussions and his input and feedback, this work would have not been possible. I would like to extend my thanks and appreciation to Dennis Schieferdecker for his supervision, insights and feedbacks that were always of great value to me.

At EKP, I would like to thank the entire group for providing a productive and fun working atmosphere. Particularly, I would like to recognize Joram Berger, Fred-Markus Stober, Manuel Zeise, Max Fischer and Marcus Schmitt for their physical and computing input. I would like to thank Georg Sieber for proofreading my thesis. Moreover, my thanks go to Corinna Günth, Dominik Haitz, Oliver Oberst, Klaus Rabbertz, Raphael Friese, Thomas Müller and Fabio Colombo.

At CERN, I thank Vincenzo Innocente for his guidance and vision. Additionally, I would like to recognize Marcin Konecki for his insights into the CMSSW track reconstruction code and Danilo Piparo for his ideas and feedback.

At ITI, I thank Vitaly Osipov for his input on GPGPUs.

Last but not least, I would like to express my deepest gratitude to my girlfriend Margit for all her support and patience during the last months. Finally, I would like to extend my sincerest thanks and appreciation to my parents, Manuel and Stefanie, that supported me in all my endeavors.

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und nur die angegebenen Hilfsmittel verwendet zu haben.

Daniel Funke

Karlsruhe, den 28. Juni 2013