Department of Physics and Astronomy

University of Heidelberg

Master thesis

in Physics

submitted by

Carsten Grzesik

born in Görlitz

June 2016

Online Track Reconstruction On Graphics Processing Units For The MuPix-Telescope

This Master thesis has been carried out by Carsten Grzesik

at the

Intitute of Physics

under the supervision of

Prof. Dr. André Schöning

Spurrekonstruktion auf Grafikprozessoren für das MuPix-Teleskop

Für die Suche nach dem Zerfall $\mu^+ \to e^+e^-e^+$, der die Leptonenfamilienzahl verletzt, benötigt das Mu3e-Experiment einen präzisen Spurdetektor der bei hohen Teilchenraten betrieben werden kann. Die geplante Ereignisrate produziert dabei eine enorme Menge an Daten, die aktuell mit keinem Speichermedium gespeichert werden kann. Deshalb muss die Datenrate im laufenden Experiment um einen Faktor 1000 reduziert werden. Dies wird mit einer Online-Spurrekonstruktion erreicht, die auf Grafikprozessoren (GPUs) ausgeführt wird und interessante Ereignisse zum Speichern auswählt.
Als einen Schritt zur finalen Detektorauslese wird in dieser Arbeit eine erste Implementation einer Online-Spurrekonstruktion auf GPUs für das MuPix-Teleskop vorgestellt, welches ein Strahlteleskop mit Sensorprototypen für den Pixeldetektor des Mu3e-Experiments ist. Im Zuge dessen wurde ein Spurrekonstruktionsalgorithmus für das Teleskop auf GPUs implementiert und die Kommunikation zwischen einer feldprogrammierbaren Logikgatter-Anordnung (FPGA) und der GPU im Auslesecomputer entwickelt. Der FPGA wird für die Steuerung und Auslese der Pixelsensoren im Teleskop genutzt. Ein Vorsortier-Algorithmus wurde in der FPGA Firmware implementiert um den GPU-Algorithmus effizient nutzen zu können. Insgesamt lief das beschriebene System erfolgreich mit simulierten Daten und während zwei Teststrahlkampagnen am DESY- und MAMI-Beschleuniger.

Online Track Reconstruction On Graphics Processing Units For The MuPix Telescope

To search for the lepton flavour violating decay $\mu^+ \to e^+e^-e^+$ with a sensitivity four magnitudes better than the current limit, the Mu3e experiment requires an accurate tracking detector running at high rates. The desired event rate creates a high amount of data, that can not be stored by any current data storage device. Thus the data has to be reduced by about a factor 1000 during the experiment's operation. The reduction is achieved by an online track reconstruction algorithm executed on Graphics Processing Units (GPU) selecting interesting events to store.
On the way to the final detector readout, this thesis presents a first implementation test of online tracking on GPUs for the MuPix telescope which is a beam telescope consisting of prototype sensors for the Mu3e pixel detector. A tracking algorithm for the telescope was implemented on GPUs along with the development and testing of the communication between a Field Programmable Gate Array (FPGA) and the GPU in the readout PC. The FPGA controls and reads out the pixel sensors of the telescope. Since the GPU algorithm needs an additional pre-sorting of the data to run efficiently, it has been implemented in the FPGA firmware. This system was running successfully with simulated data and at testbeams at DESY and MAMI.

Contents

Part I

Introduction

1 Overview

The search for physics beyond the Standard Model is one of the main challenges in modern particle physics. The Standard Model of Particle Physics describes the fundamental constituents of matter and their interactions very well but cannot explain all phenomena observed so far. One of these phenomena is the mixing of lepton flavour states, which has been observed for neutrinos. The Standard Model has been expanded to describe lepton flavour violation in the neutrino sector. However, mixing phenomena have not been observed for charged leptons. It is heavily suppressed for interactions via neutrino mixing. Thus, any observation of charged lepton flavour violation is a clear indicator for new physics.
The proposed Mu3e experiment will search for the charged lepton flavour violating decay of a positive muon into two positrons and an electron. To this aim, a continuous, high rate muon beam is used to stop up to $2 \cdot 10^9$ muons per second. The decay products are tracked by a cylindrical spectrometer, consisting of a high granular silicon pixel, scintillating fibre and tile detector, placed around the muon stopping target. Background processes have to be suppressed by excellent spatial, momentum and timing resolution, to achieve the proposed sensitivity for the branching ratio of 10^{-16} at 90% confidence level. The pixel detector is made of novel, thin High Voltage Monolithic Active Pixel Sensors (HV-MAPS) to reconstruct the vertex and momentum of the low momentum decay particles ($E < 54\,\mathrm{MeV}$).
The data acquisition system uses no hardware trigger to read out the detector and handles data rates in the order of $1\,\mathrm{Tbit\,s^{-1}}$. For such high data rates it is not possible to write the whole data to storage. Therefore, an online event selection by a complete track reconstruction is needed to reduce the incoming data by about a factor 1000 while the experiment is running. This triggers the need of high-performance computing techniques to be used in the filter farm PCs. Graphics Processing Units (GPU) provide a highly parallel computing architecture suitable for this task. Currently, an algorithm for parallel track reconstruction is developed for the Mu3e experiment.
To test the feasibility of an online track reconstruction system for the Mu3e tracking detector, this thesis investigates an implementation of a GPU-based online track reconstruction for the MuPix telescope. The HV-MAPS prototypes for the Mu3e pixel detector, called MuPix, are arranged in a four layer telescope for testbeam characterization measurements of the sensors and integration tests with components of the final Mu3e readout system. Especially the direct communication between a Field Programmable Gate Array (FPGA) and the GPU is developed and tested. The FPGA interfaces the telescope sensors with a readout PC. Simulated data was used to study the performance of a GPU in context of an online reconstruction and two testbeam campaigns served as a first implementation test of the system.
At the beginning, the physical background is described. The SM and lepton flavour violation are briefly explained to motivate the Mu3e experiment, which is described in chapter 3. A description of the computation technologies, used in the scope of this thesis, is provided in chapter 4. The MuPix telescope setup, readout concept and the used track fit is shown in chapter 5. Chapter 6 explains the implementation of the track fit for a GPU. It concentrates on the effects of the parallel structure of the GPU on the track fit implementation. Then, the analysis of simulated data with the GPU track reconstruction is shown in chapter 7. In the following chapters 8 and 9, the results of the testbeam campaigns at DESY and MAMI are discussed. The last chapters give a short summary about the studies and an outlook on further developments on the MuPix telescope online track reconstruction on GPUs.

2 Theoretical Background

2.1 The Standard Model

Currently, the Standard Model of Particle Physics (SM) [1] is the best theory to describe the constituents of matter and their interactions. It is a quantum field theory describing the electromagnetic, weak and strong interactions and includes all known fundamental particles. Figure 2.1 shows the building blocks of the SM and their interrelationship.

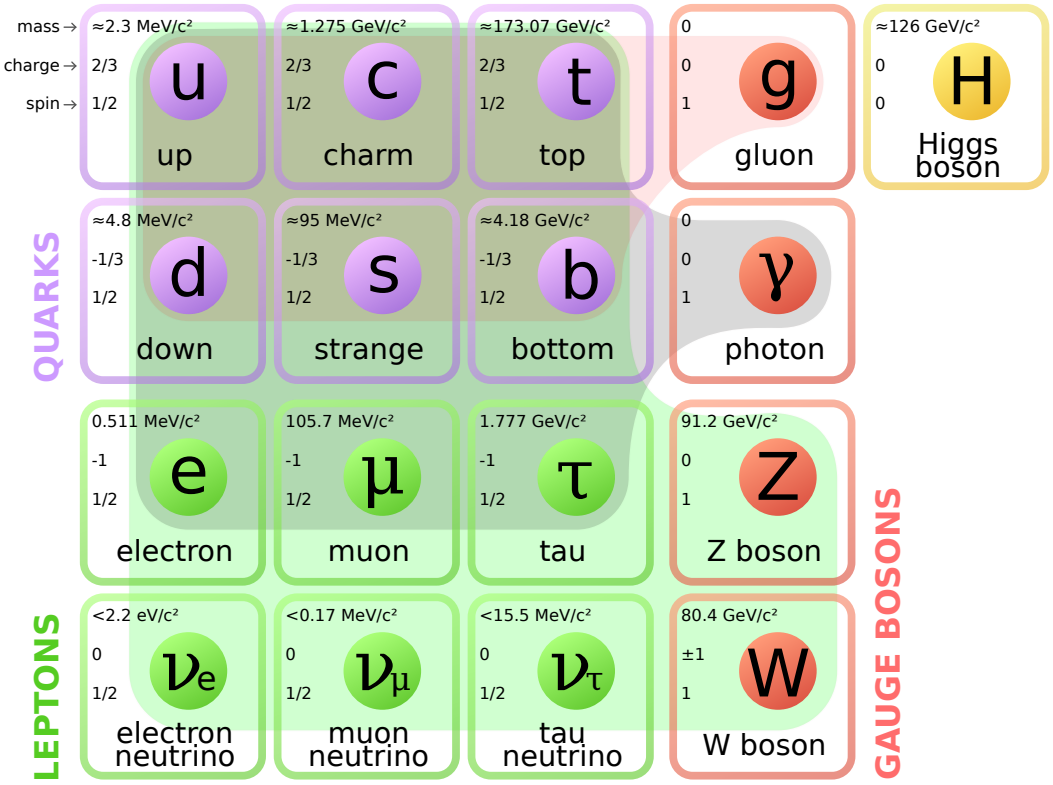


Figure 2.1: Elementary particles in SM physics [2].

There are twelve spin-$\frac{1}{2}$ particles called fermions making up matter, their oppositely charged antiparticles and another five spin-1 particles called bosons which mediate the forces. The fermions are grouped by the interactions they take part in. Undergoing only electromagnetic and weak interactions the leptons are again grouped in three flavors. Each lepton flavor contains an electrically charged fermion and its neutral counterpart, the neutrino that only interacts weakly. In the SM the lepton flavor is a conserved quantity and neutrinos are massless.

The other group of fermions, the so called quarks, are the only particles that interact strongly, because they carry a charge called color. Quantum Chromodynamics (QCD) describes the strong interaction that does not allow free, colored particles. Therefore, they always appear as color neutral bound states, called hadrons. This is called color confinement and originates from the nature of the mediating particles of the strong force. The bosons associated with the strong interaction are called gluons. Since they also carry color charge themselves, they

can not be detected as free particles in nature and interact strongly among each other. This is considered to be the reason for the confinement. In contrast to the electromagnetic force, this results in direct proportionality of the force between two colored objects to their distance.
Due to their electric charge quarks are also involved in electromagnetic interaction, which is mediated by the massless and neutral photon. The weak interaction is the only one that has massive mediating bosons, the $W^{\pm}$ and Z bosons. They couple to all fermions and have electric charge of ± 1 and 0 respectively. Due to the relatively high mass of the bosons (c.f. figure 2.1) the interaction is short ranged. The weak interaction is the only one which can change the quark flavour, which is the type of a quark associated with its mass. This flavor changing mechanism is described by the Cabibbo–Kobayashi–Maskawa (CKM) matrix.
In the SM it is forbidden to change the lepton flavor. Due to the discovery of neutrino mixing, which is the transition of neutrinos between different flavors, the SM has been adapted to it. This lepton flavor violation (LFV) between the neutrino flavors is described by the Pontecorvo-Maki-Nakagawa-Sakata matrix (PMNS). It implies a non-zero mass of the neutrinos and also enables lepton flavor violation for the charged leptons (cLFV), but with a tiny branching fraction, which makes it impossible to measure.
The last boson discovered is the long ago predicted, so-called Higgs particle. It was predicted in 1964 by three groups lead by R.Brout, F.Englert [3], P.Higgs [4], G.Guralnik, C.Hagen and T.Kibble [5] and discovered in 2012 by the LHC experiments ATLAS and CMS [6, 7]. Via the so called Higgs mechanism, the gauge bosons acquire mass by coupling to the Higgs boson.
Although the SM is a comprehensive tool to explain most of the phenomena of particle physics, there are still observations not yet explained. Conceptually it can not integrate the fourth fundamental force, gravity. Therefore, theorists search for unified theories. Besides the missing link to gravity there are other phenomena not described in SM, e.g. the properties of dark matter and dark energy, as well as the mechanism that produced the imbalance of matter and antimatter in the universe, called baryogenesis, and the strong CP problem, which is the problem of a non observed, combined charge conjugation and parity violation in the strong sector, although it is not forbidden in the SM. A lot of theoretical frameworks exist to solve these problems, e.g string theory, supersymmetric theories or loop quantum gravity. To prove or exclude models beyond the SM (BSM) two different approaches are followed by experimental physicist. On the one hand one can directly search for new particles at high energies. On the other hand one can conduct precision experiments to measure model-dependent parameters. Examples of the first kind are the LHC experiments, measurements to find charged lepton flavor violation can be assigned to the second group.

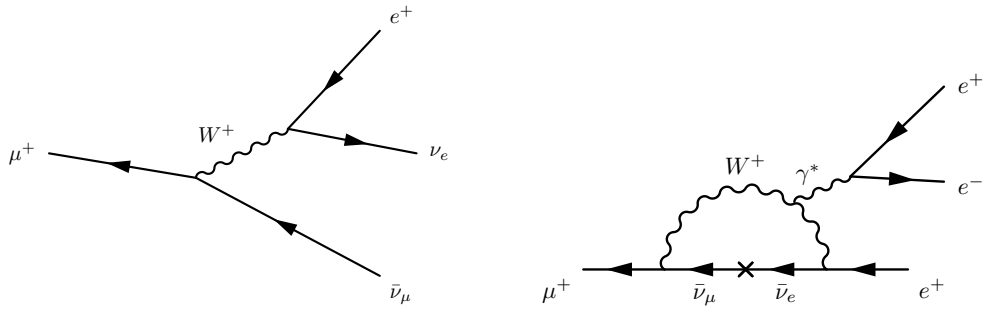
2.2 Muon Decays

The muon decay will be described as a possible cLFV process in the following sections.

2.2.1 Muon Decay in the Standard Model

The dominant process in the muon decay is the so called Michel decay. Figure 2.2a shows the Feynman diagram of such a process, with a muon decaying weakly into an electron, a muon neutrino and an electron anti-neutrino. The lepton flavor is conserved in this process, because the muon neutrino keeps the muon lepton flavor and the electronic lepton number $(+1)$ of the electron and the anti-neutrino (-1) vanish. So we have a muonic lepton number of $+1$ and an electronic lepton number of 0 in the initial and final state.
By neutrino mixing, a lepton flavor violating decay of the muon is possible in the SM. One possible process is the decay of a muon into two positrons and an electron. Starting with a positively charged anti-muon the decay products then are two positrons (anti-electrons) and an electron. Figure 2.2b shows such a decay for which the branching ratio (BR) is suppressed

to $< 10^{-54}$ by the mass ratio $\frac{(\Delta m_\nu^2)^2}{m_W^4}$ [8]. This is not detectable by current means but there exist different BSM models that lead to an enhanced BR for this cLFV process.



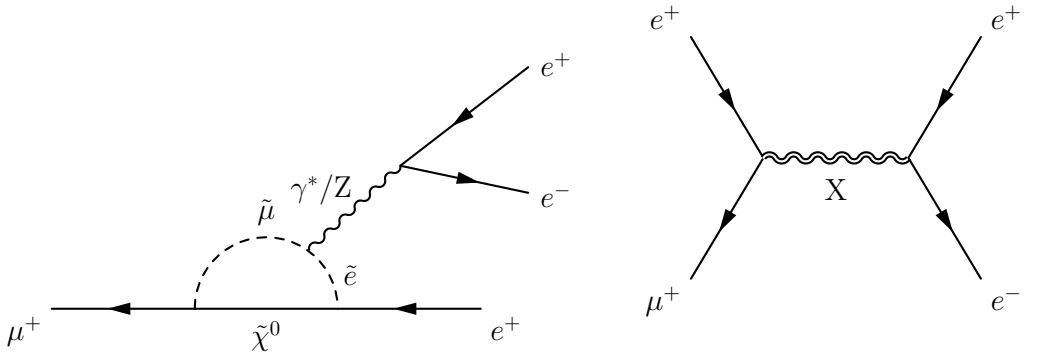
(a) Michel decay: $\mu^+ \to e^+ \tilde{\nu}_\mu \nu_e$.

(b) $\mu^+ \to e^+ e^- e^+$ decay by neutrino oscillation.

Figure 2.2: Feynman diagrams of SM muon decays.

2.2.2 Muon Decay in Models beyond the Standard Model

A detection of such a decay of a muon into three electrons would be a clear sign for BSM physics and can provide validation or exclusion of theories predicting enhanced BR values. Figure 2.3 shows two possible processes for the decay $\mu^+ \to e^+ e^- e^+$ in BSM models, which can enhance the BR. The process could be mediated by supersymmetric particles that change lepton flavor in the superpartner, which are the non-observed partners of SM particles in SUSY models, regime (c.f. figure 2.3a). Another possibility are tree level decays mediated by a new particle like doubly charged Higgs particles, R-parity violating scalar neutrinos or heavy vector bosons.



(a) SUSY decay.

(b) Tree diagram with a new mediating particle.

Figure 2.3: $\mu^+ \to e^+ e^- e^+$ Feynman diagrams in BSM models.

2.3 Experimental Situation in charged Lepton Flavor Violation Physics

Measurements to search for cLFV processes have already been started in the 1950s [8]. There was a variety of experiments to search for cLFV processes and many future experiments are

planned to explore BSM physics. Up to now none of these processes has been detected, only upper limits exist for the BR. The historical pathway of limits on the BR (at 90% confidence level (C.L.)) for various decay channels of muon and tauon not conserving the lepton flavor are given in figure 2.4.

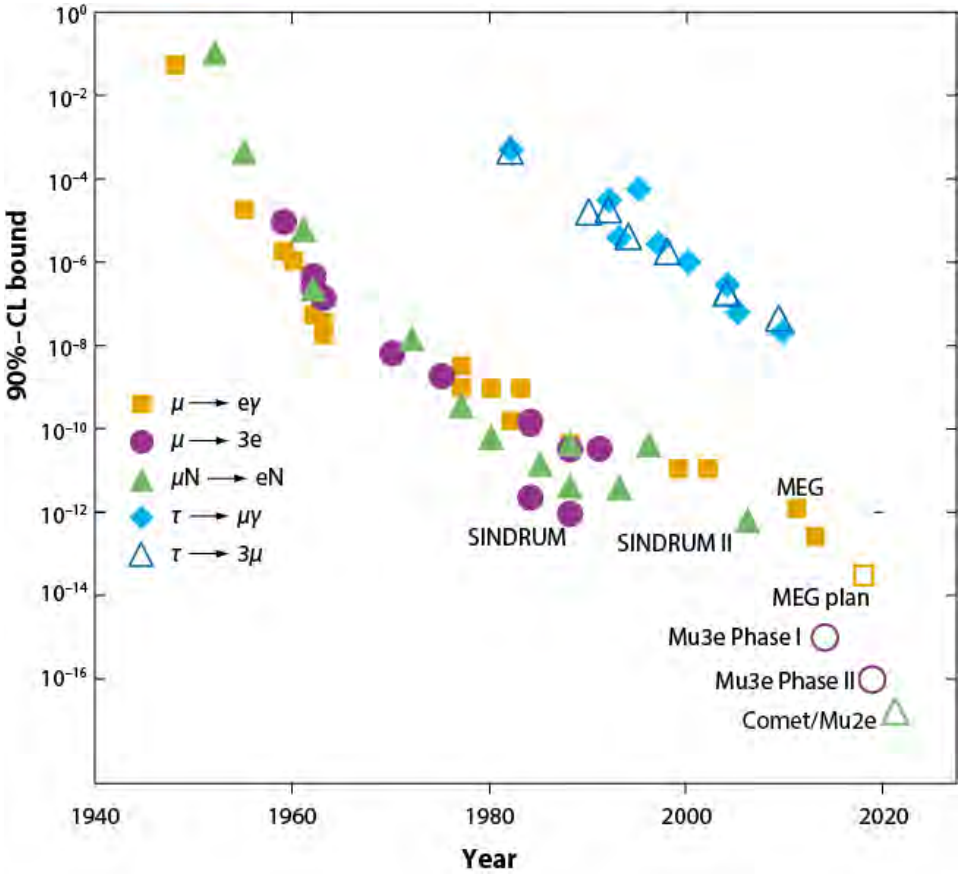


Figure 2.4: History of cLFV searches and prospects for future experiments, adapted from [9].

The most recent values for the muon decays $\mu \to e\gamma$, $\mu \to 3e$ and $\mu N \to eN$ come from the MEG, SINDRUM and SINDRUM II experiments, respectively.

MEG Experiment The MEG experiment is build to search for the decay $\mu^+ \to e^+\gamma$. Since this is a two particle decay, the two main parts of the detector are arranged back to back to detect the positron and the photon. Just recently the limit on the BR of the $\mu^+ \to e^+\gamma$ decay was set to $< 4.2 \cdot 10^{-13}$ (90% C.L.) [10]. This decay mode is especially sensitive to new heavy particles that mediate a LFV dipole coupling, e.g in SUSY models (cf. figure 2.3a). Requiring an on-shell photon it is less or even not sensitive to tree level models (cf. figure 2.3b), Z-penguin or box diagrams.

SINDRUM Experiment The SINDRUM experiment searched for the decay $\mu^+ \to e^+e^-e^+$ from 1983 to 1986. No signal was found which lead to the still leading limit on the BR $< 1.0 \cdot 10^{-12}$ (90% C.L.) [11]. For gauge boson mediated LFV, e.g. SUSY models, this decay mode is more than two orders of magnitude less sensitive than $\mu^+ \to e^+\gamma$, but enables a lot more models at tree level, as discussed in section 2.2.2.

Conversion Experiments Another possible cLFV process is the direct conversion of a muon into an electron $\mu \to e$. To conserve energy in this process the vicinity of a nucleus is

required, which means $\mu N \to eN$. Several experiments have been searching for this process, with various kinds of nuclei. The SINDRUM II collaboration sets the strongest limit to BR $< 7 \cdot 10^{-13}$ (90% C.L.) using gold atoms as target [12]. In general, this decay can proof the same models as $\mu \to eee$ with the addition of possible quark interactions, e.g. leptoquark models.

τ Decays Like in the muon decay, cLFV can also occur in decays of the heavier tauon. A lot of channels in the tauon decay, enabling cLFV, have been explored by B-factories. For most of the channels the limits on branching ratios have been set to a few 10^{-8} [13].

3 The Mu3e Experiment

The Mu3e experiment aims to measure the cLFV decay $\mu^+ \to e^+e^-e^+$ with a sensitivity on the BR of 10^{-16} (90% C.L.). This is four orders of magnitudes lower than the current limit measured by the SINDRUM experiment[11]. High particle rates are necessary to observe the required number of muon decays to reach the sensitivity goal in a reasonable amount of time. Modern silicon pixel detectors in combination with scintillating fibres and tiles make it possible to achieve precise spatial and timing information at the high rates.

3.1 The $\mu \to eee$ Decay

The required muon rate is in the order of 10^9 muons per second on the target and will be available at the planned High Intensity Muon Beam (HIMB) at Paul Scherrer Institute (PSI) in Switzerland. Muons coming from a beam line are stopped in a target where they decay at rest. To measure the momentum of the electrically charged decay products and the sign of their charge the detector is placed in a homogeneous magnetic field aligned to the beam direction. For a signal of the desired $\mu^+ \to e^+e^-e^+$ decay the detector has to measure two positrons and an electron coming from the target region. Other processes can produce such a detector response as well and contribute to the background.

3.1.1 Signal Process

Two positrons and one electron have to come from a common point in the target, where the muon decay took place, the so called primary vertex. The invariant mass calculated from the decay particle four momenta has to be equal to the rest mass of the muon, as given in the equation 3.1.

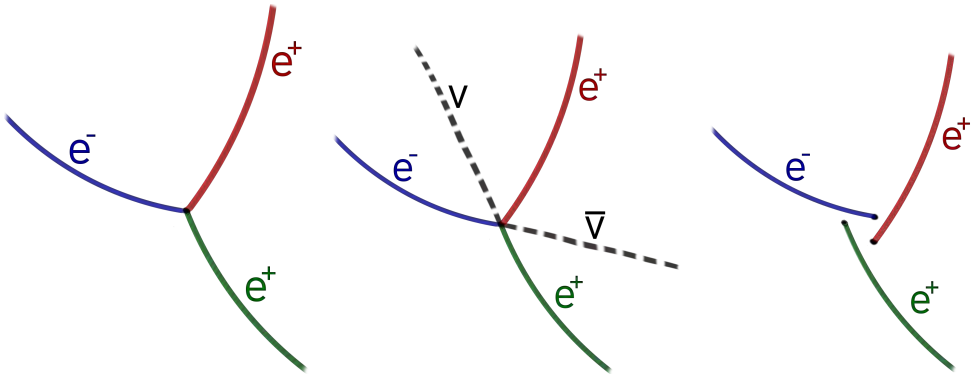
$$m_\mu = \left| \sum_{i=1}^{3} \mathbf{p}_i \right| \tag{3.1}$$

$\mathbf{p}_i$ is the four-momentum of decay particle i and $m_\mu = 105.7\,\text{MeV}$ in natural units. Assuming a muon decaying at rest in the target, the momenta $\vec{p}_i$ have to sum up to zero. These two facts characterize the signal process in the detector. Due to limited resolution in reconstructing the vertex position, momentum and energy, and timing of the decay particles, background processes can potentially produce a similar signature. The final sensitivity is determined by the ability to suppress these backgrounds.

3.1.2 Background Processes

There are mainly two types of processes contributing to the background in Mu3e. One is given by radiative SM decays where neutrinos are not detected and the other is contributing via multiple decays that have spatially close vertices. Figure 3.1 shows schematic topologies of the signal decay and the two types of background.

Internal conversion Decay SM muon decays with a photon undergoing internal conversion to an e^+e^- pair are considered in this type: $\mu^+ \to e^+e^-e^+\bar{\nu}_\mu\nu_e$. The only chance to distinguish this from the signal is the missing energy of the three decay particles carried away by the two neutrinos. Figure 3.2 shows the required energy resolution to suppress the background due to internal conversion as a function of the BR.



(a) Signal (b) Radiative Decay (c) Combinatorial Background

Figure 3.1: Schematic of signal and background processes.

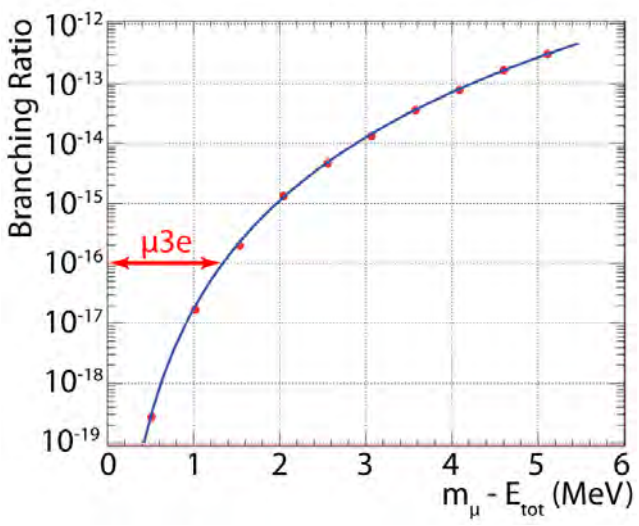


Figure 3.2: Branching ratio of the radiative decay $\mu^+ \to e^+e^-e^+\bar{\nu}_\mu\nu_e$ to $\mu^+ \to e^+\bar{\nu}_\mu\nu_e$ depending on the missing energy of the three decay particles to the muon rest mass [14].

Combinatorial Background The main contribution to this type of background comes from the Michel decay $\mu^+ \to e^+\bar{\nu}_\mu\nu_e$. But it does not provide a negatively charged decay particle, an electron. However, by falsely reconstructing multiple vertices from different processes into one vertex this can give a contribution to the background. Electrons can originate from radiative decays. like $\mu^+ \to e^+\gamma\bar{\nu}_\mu\nu_e$ where the photon creates an e^+e^- pair, or Bhabha scattering. Bhabha scattering is the process of a positron scattered off an electron, which can occur in any detector material and frees a bound electron. To reduce this background a good vertex resolution is required, which is reached by the spatial resolution of the detector and by minimizing the scattering in the detector.

3.2 The Mu3e Detector

As described in section 3.1, the detector requires the capability of a high rate measurement (up to 2 GHz muon decay rate) with an adequate spatial and time resolution and a minimum amount of material to reduce the background contribution, as discussed in section 3.1.2. The detector design is specified to meet these requirements [15].

3.2.1 Detector Layout

The basic detector layout is shown in figure 3.3. The full detector has an overall length of 2 m and a diameter of 18 cm.

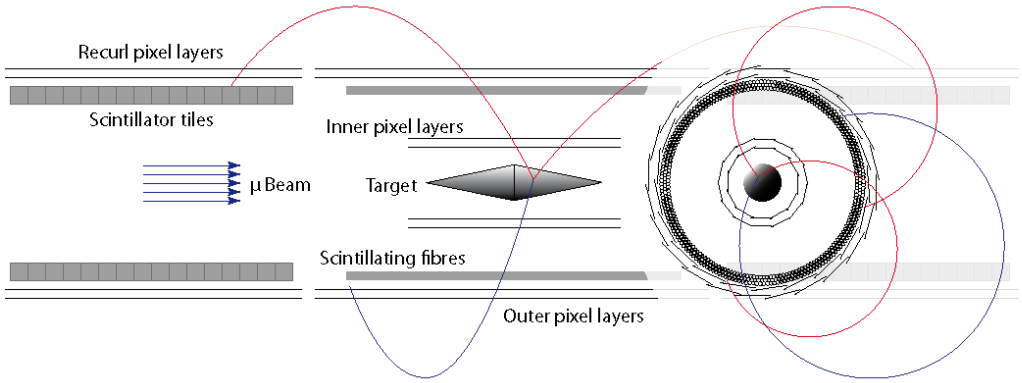


Figure 3.3: Schematic of the full detector (cut along the beam axis (left) and transversely (right)) with the muon beam from the left, pixel, scintillating fibre and tile detectors. Exemplary tracks are given in blue and red. The whole detector is placed in a homogeneous magnetic field along the beam direction[15].

The detector will be placed in a muon beam pointing to the hollow double cone target. Here the muons decay after being stopped by interaction with the target material. The shape of the target is chosen such that the muons are stopped with wide range in r- and z-direction. Passing through a homogeneous magnetic field of 1 T the electrically charged decay particles get deflected. Tracks are measured by two double layers of silicon pixel detectors. Between the second and third layer there is a scintillating fibre detector with better timing resolution than the pixel detector. Particles leaving the fourth layer of the pixel detector re-curl into the active region again. They can enter the central detector part again or one of the four recurl stations. The recurl stations consist of two layers of pixel sensors and a layer of scintillating tiles inside the pixel layers to stop the particles and measure the time with an even better resolution ($\mathcal{O}(100\,\text{ps})$) compared to the fibre detector. The recurl station pixel detectors improve the momentum resolution.
Since the energy of the resulting electrons is smaller than 54 MeV the particles get strongly deflected by multiple Coulomb scattering in the detector material. The amount of material has to be kept as low as possible because this limits the momentum resolution. Therefore, the silicon sensors are thinned to 50 µm and thin flexprint cables are used to connect detector components.. A global helium gas flow is used to cool the detector. Helium combines high heat conductivity and low multiple scattering and was therefore chosen for the cooling of the pixel detector[16], which dissipates up to $400\,\text{mW}\,\text{cm}^{-2}$ of heat.
This thesis is closely related to testing and reading out the pixel detector, therefore it will be described in more detail in the following section.

3.2.2 The Pixel Detector

For the final experiment the pixel detector consists of one center barrel part with four layers of silicon detectors and four extension barrels of two layers each up- and downstream. The central station has two layers of detectors close to the target to determine the vertex position of decay particle tracks and the outer layers are used to measure the momentum of

the outgoing particles. With the help of the recurl stations the momentum resolution can be greatly improved due to the long lever arm. With $\mathcal{O}(1\,\mathrm{m}^2)$ of active area the pixel detector will close to 300 million pixels in the end.
The sensors are thinned to 50 µm and the support structure for the sensors is made of 25 µm thick polyimide foil that is glued to plastic end pieces, as shown in figure 3.4.

(a) Inner layers. (b) Part of an outer layer.

Figure 3.4: Pictures of mechanical prototypes for the Mu3e pixel detector build from polyimide foil with glass plates representing the sensors.

The outer layers feature a V-shape folding (cf. figure 3.4b) to enhance stability and enable the possibility of an extra helium flow for cooling. Since the global flow of helium heats up towards the end of the detector the extra cooling flow is inserted in the other direction in the V-shaped channels under the chips.
The pixel sensors will be built in the novel High-Voltage Monolithic Active Pixel Sensor (HV-MAPS) technology [17].

HV-MAPS The current technology for pixel detectors, used e.g. in the large LHC experiments (ATLAS, CMS), consists of a segmented sensitive material (e.g. silicon diodes) and a chip for the readout electronics which is connected via bump-bonds. These hybrid sensors feature a complex production process to connect all pixels to the readout chip and the bump-bonds add material to the detector. Since it is typically a heavy metal bond to conduct the analog signal to the digitalization part, they add a significant amount of multiple scattering.
The idea to have the analog diode and the readout logic in the same chip is realized in Monolithic Active Pixel Sensors (MAPS). Because they can be produced using commercially available CMOS processes, as used in the multimedia industry for example, they feature very small structure sizes and relatively low production costs. Bump-bonds are not required, which reduces multiple scattering. However the charge collection in the diode relies on diffusion, which makes it slow. Hybrid sensors use a reverse-bias applied to the diodes to collect charges by drift, which is faster.
HV-MAPS combines these two features, being a MAPS that uses reversed-bias diodes for charge collection. Figure 3.5 shows a sketch of an HV-MAPS.
It features a p-doped substrate with n-doped wells and p-doped islands in the n-wells to implement the CMOS logic. The reverse-bias voltage of up to 85 V is applied between the substrate and the n-wells and creates a fully depleted region. A particle passing through this region creates electron-hole pairs which are collected at the electrodes via drift which is faster than diffusion and reduces the occurrence of clustered hits. This analog signal is amplified in the pixel and digitized in the periphery on the edge of the chip. So each chip has a purely digital output. The thickness of the depletion zone of ∼10 µm allows to thin the sensor to about 50 µm.

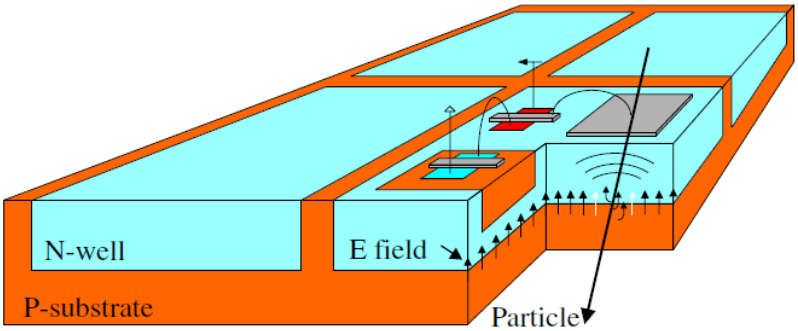


Figure 3.5: Sketch of an HV-MAPS chip with analog and digital part in one chip [17].

MuPix Prototypes HV-MAPS is the technology chosen for the Mu3e experiment. Therefore, the sixth version of sensor prototypes is currently characterized [18]. It has a 32×40 pixel matrix with $103 \times 80\,\mu m^2$ pixel size and was thinned to a minimum of $50\,\mu m$. The readout of the chip is described in section 5.3.

3.2.3 Readout System

The Mu3e experiment will feature a triggerless readout system for all subdetectors with a push architecture. This means that all parts of the detector send data continuously to the data acquisition system (DAQ). The structure of the DAQ for the Mu3e experiment is shown in figure 3.6.

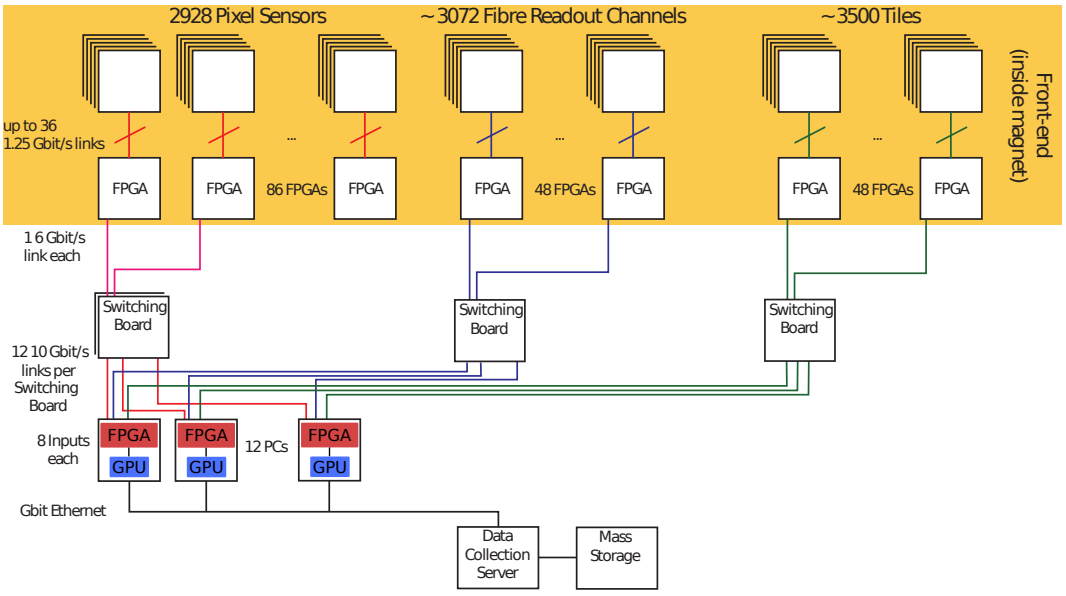


Figure 3.6: Schematic overview of the Mu3e readout scheme [19].

Front-end FPGAs collect the data of multiple detector channels (<100) locally in the detector and send it off to the switching boards. Multiple switching boards merge data from their associated front-end boards and schedule the distribution to the filter farm PCs. The filter farm reduces the amount of data by a factor ~ 1000 to be able to write it to the data storage. They get the data of the whole detector for a specific time slice and perform an online track and vertex reconstruction to select interesting events to be stored.

Online Reconstruction The full amount of data produced by the Mu3e detector can not be written to any data storage. Therefore, the data has to be reduced by selecting possible signal events. An algorithm implemented on a GPU reconstructs tracks from the hit data of the whole detector.
The filter farm GPUs receive data from an FPGA, connected to the switching boards.

4 Computing Technology

Scientific research in general and particle physics in particular have moved to putting high demand on data processing and computation to solve extensive problems in manageable time and with affordable energy consumption. To accommodate this demand, researchers have come from serial computing to high-performance computing (HPC). When the heat dissipation problem stopped the increase of clock rates on microprocessors in the early 2000s, multi-core systems were introduced to keep up with Moore's law. Moore's law predicts the increase in transistor count on integrated circuits since 1965 [20] and therewith the increase in computing performance. On the one hand, the development of multi-core systems enabled the possibility of executing several tasks in parallel, but on the other hand it made the efficient use of the provided resources more complex. Parallel tasks can easily be executed serially, but for purely serial problems it can be hard or even impossible to parallelize them. Besides the usage of multi-core central-processing units (CPU), other types of computing and electronic devices can be used for HPC, e.g. application-specific integrated circuits (ASIC), field-programmable gate arrays (FPGA) and Graphics Processing Units (GPU). In the scope of this thesis the latter two will be considered, since they will be greatly used in the Mu3e readout system. Before computation can be executed on data it has to be transported to the computation units. To use an advanced data processing the data transmission system has to keep up with its performance.

4.1 Data Transmission

Data transmission is the transport of information using a physical observable as the transport medium. For particle physics experiments it is crucial to transmit the measured data reliably from the detector to the data storage or analysis system via a suitable readout system. With increasing event rates and high granularity detectors the amount of data to be transmitted increases. Therefore, the amount of data that can be transmitted by the readout system in a given time, the data bandwidth, is an important parameter that has to fit the requirements of the detector. The data bandwidth is limited by the transmission method that is used and depends on the transmission medium, e.g. electrical or optical signals, and other parameters, e.g. encoding, power consumption, noise, radiation hardness and material budget. Typically one has to find a compromise between them. In the following, the data transmission types that are applied in the scope of this thesis are explained.

4.1.1 Peripheral Component Interconnect Express (PCIe)

PCIe is a serial data bus that is commonly used in computers to transmit data from expansion units to the CPU (and the main memory). The topology is a serial point-to-point connection over switches instead of a common bus, which enables parallel communication between separate devices. The standard defines the data encoding, bus protocol as well as the connection slot and power supply of the periphery devices. Between the different nodes the signal is transferred using a differential pair for transmitting and receiving per lane. The number of lanes per connector implemented in PCIe is 1,2,4,8,16 (denoted by x1, x2, x4, x8, x16, respectively). All of the three versions available so far are hot-plugable and mainly differ by the data rate per lane. PCIe version 2.0 and 3.0 play a roll in this thesis and feature $5.0\,\mathrm{Gbit\,s^{-1}}$ and $8.0\,\mathrm{Gbit\,s^{-1}}$ raw data rate. Since the encoding overhead in PCIe 3.0 (128b/130b encoding) is reduced compared to PCIe 2.0 (8b/10), this together results in a nearly doubled usable data bandwidth.

4.1.2 Direct Memory Access

Direct Memory Access (DMA) is a feature of the PCIe bus that allows devices to write to the PC main memory and other memories without interaction of the CPU. Normally, the CPU sends a read request to a PCIe device, asking it to send a limited sized package. It has to wait for the package to arrive and writes it to the main memory. This is called polling [21].

DMA is controlled by the DMA controller without interference of the CPU, except for an initialization of the memory region to write to. The CPU allocates a region in the main memory and assigns it to the DMA controller, which can independently schedule the data transfer. Theoretically, the write process from the device to the main memory can continue forever. The data bandwidth can be increased by the reduction of control overhead and the bypass of the CPU, which also leaves it free for other tasks. GPUs make use of this technique to copy data from and to main memory.

4.1.3 Low Voltage Differential Signaling (LVDS)

LVDS is a hardware standard for electrical signal transmission that uses two signal traces per channel. With the downside of doubling the amount of signal lines it gives some advantages over single ended transmission types. It is low powered and has good signal quality in terms of low crosstalk and reduced sensitivity to disturbances on the two lines.

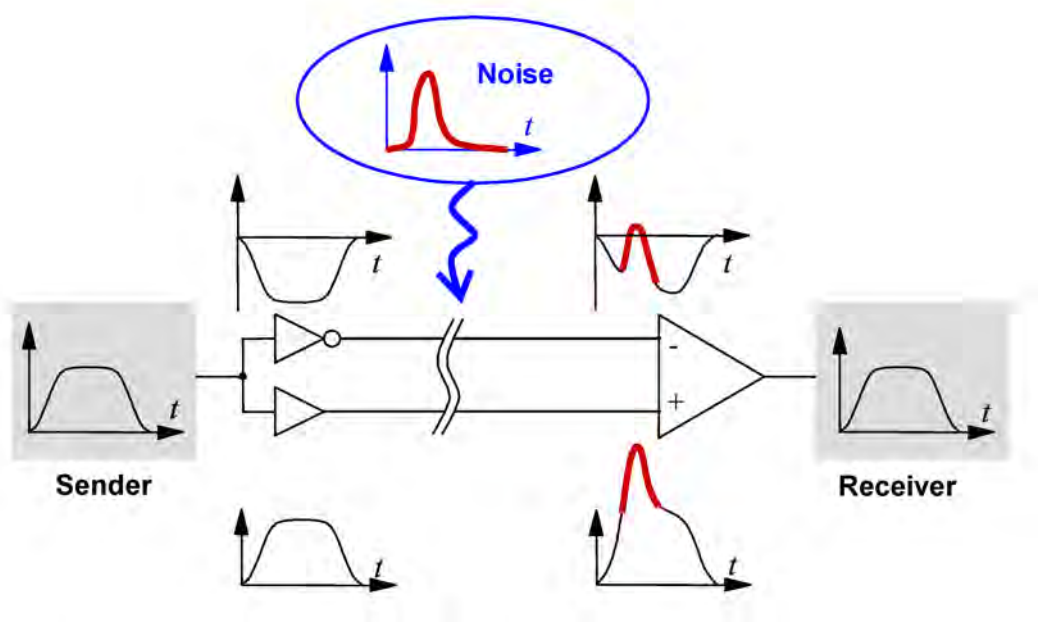


Figure 4.1: Influence of common mode noise to a differential signaling line [22].

The two lines of a channel are therefore driven with opposite polarity by a current mode driver with a constant current of 3.5 mA, resulting in a relatively low voltage drop of 350 mV on the 100 Ω terminating resistor. A combination of low signal voltage and closely placed differential signal lines reduces the far field strength of the transmission line. Since the signal is always taken as the difference between the two lines of a channel, common mode noise, influencing both lines equally (e.g. electromagnetic radiation background) cancels on the receiving side, as illustrated in figure 4.1.

4.2 Field Programmable Gate Array (FPGA)

An FPGA is an integrated electrical circuit with a user programmable layout enabling high flexibility for fitting it to the desired application. Like ASICs, FPGAs can process signals in a highly parallel structure using parallel resources without being bound to a specific behavior due to a predefined design. With lower efficiency in terms of resource usage, speed and energy consumption, but the ability of reprogramming the chip, FPGAs are best suited for prototyping or specific, small series applications. FPGAs mainly consist of three building blocks:

- Logic elements (LE)
- Interconnection network between the LEs
- Input/Output (I/O) ports

The LE's main components are a lookup table, that contains the truth table for the logic function of the LE and a register to hold the state of the LE. The in- and outputs of different LEs can be connected via a programmable wire array, which also allows to connect the I/O ports to the periphery. With this flexible architecture FPGAs can be used for every task that is computable in the limits of available LEs. FPGAs can be combined with any specialized hardware, especially with transceivers for high-speed signal transmission. This allows for an excellent use of FPGAs for signal transmission and processing in a physics experiment. Thus, an FPGA development board by Altera, providing many I/O connections, transceivers and application specific hardware along with the FPGA chip is used in the readout of the MuPix telescope (cf. section 5.3) and custom-made boards will be an integral part of the Mu3e readout system, as described in section 3.2.3.

4.3 Grapical Processing Unit (GPU)

GPUs were developed to compute the display output of a computer. The video game industry pushes the development of high performance GPUs for a wide market. This makes them powerful and reasonably priced. Due to the high number of pixels on a computer screen and the independence of these pixels, GPUs have a highly parallel structure. Making use of this special hardware for computations not related to computer graphics is referred to as general purpose computing on graphics processing units (GPGPU). It is especially desirable for parallel algorithms and problems that can be described by the single instruction multiple thread (SIMT) model, which means that an algorithm acts on multiple sets of data with the same set of operations. In comparison to single instruction multiple data (SIMD) it allows for different sets of registers, addresses and branching flow paths between the threads (the concept of a thread is explained in section 4.3.2), but is not meant to handle completely different tasks as the simultaneous multithreading model (SMT).
Figure 4.2 shows the main differences in execution with the three parallelism models.
So SIMT is somewhere in the middle between SIMD and SMT. Branching in the execution flow is possible but will not perform well if the ratio of branched execution count to SIMD-like executions is high. The differences in hardware design and execution flow between CPUs and GPUs is discussed in the following section.

4.3.1 Hardware Difference between GPU and CPU

CPU and GPU are both microprocessors to execute tasks on a computer. Figure 4.3 shows the main difference between a CPU and a GPU chipset.

CPU To satisfy the requirements of an operating system that has several, very different tasks to be executed at the same time, the CPU chip is largely occupied by cache memory and control units. Cache is memory situated close to the execution units which makes it

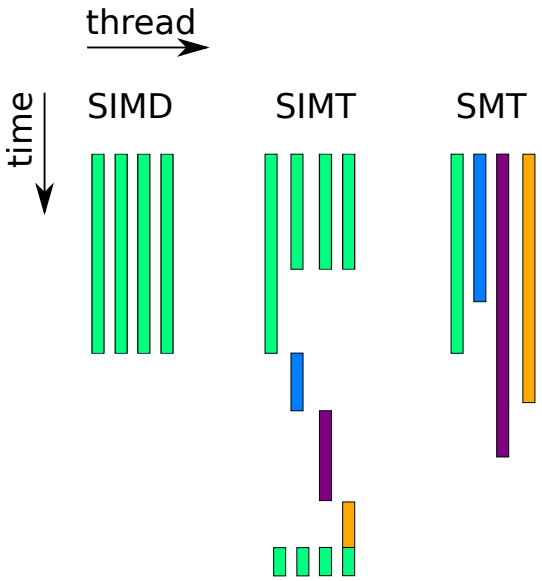


Figure 4.2: Schematic of an execution flow example for four threads in SIMD, SIMT and SMT. The bars represent processes executed in different threads with same color denoting the same instructions on multiple data. Different colors mean completely different processes, that can also belong to different programs. One can see that SIMD does not allow for any branching between the threads, while SIMT allows branching but the hardware units can only execute one process at a time. SMT can execute totally different processes at the same time. In this example: SIMD executes the green process in parallel, SIMT starts with same green process in parallel, branches into the different tasks that are executed individually and merges back on the green task again. The SMT executes completely different tasks in parallel.

fast to access but due to its position on-chip and fast implementation it is more expensive than separate memory blocks. Due to costs the fast cache is smaller than the main memory. Typically, cache is used in several levels with increasing size and decreasing data bandwidth for higher levels. Cache is mainly used to load data from the main memory before it is needed for computations on the CPU and to buffer intermediate results to minimize latency for memory accesses. In addition, the CPU chip contains a fairly large area for control units to schedule the different tasks to be executed on it. In this simplified description, the rest of the chip is used for arithmetic logic units (ALU) that execute the actual computation on the data types. In general, modern CPUs are optimized for SMT allowing additional SIMD instructions in the threads (cf. 4.2).

GPU In contrast to CPUs the GPU chip spends most of the available space on ALUs. Thus, the GPU features more execution units with the downside of reduced performance on scheduling of different tasks and the need for a data set that needs similar computations tasks on independent data elements.
Table 4.1 compares some hardware parameters for a current CPU and GPU model, that were used in the scope of this thesis.
The difference in the number of ALUs per compute core and the resulting total number of ALUs per chip shows the high parallelism of the GPU compared to the CPU. It also implies the need for different parallel computing architectures. The frequency with which the individual cores on the GPU work is a factor 3 lower compared to the CPU, which also features a lot more cache with a deeper hierarchy (3 levels versus 2 levels for the GPU). This enables more flexibility in the usage of cache on the CPU but also requires more effort in

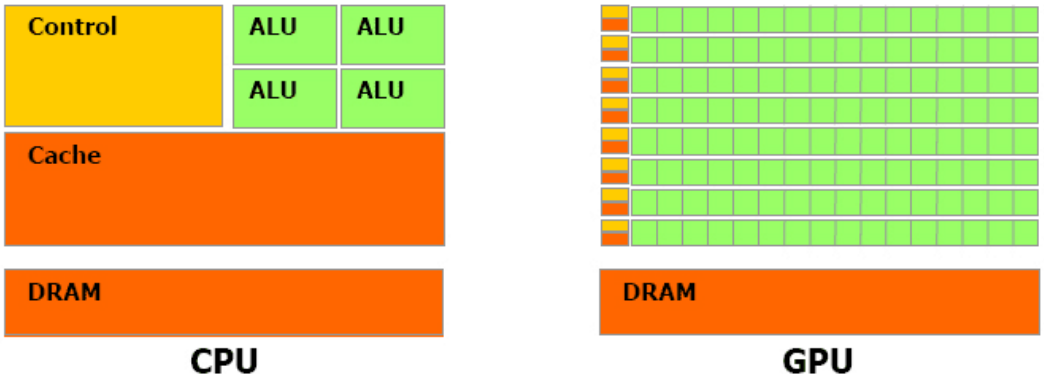


Figure 4.3: Schematic of the distribution of cache memory, control units and arithmetic logic units (ALU) on a CPU and a GPU chip [23].

	CPU (i7-5820K)	GPU (GTX 980)
Core count	6	16
ALU count	4 per core	128 per core
Frequency	3.6 GHz	1.2 GHz
Max. Cache size	15 MB (L3)	2 MB (L2)
Memory bandwidth	16 GB s^{-1}	224 GB s^{-1}
Memory capacity	16 GB (up to 64 GB)	4 GB

Table 4.1: Comparison of exemplified CPU and GPU specifications for the models used in this thesis [24, 25]. The cores on the GPU are called streaming multiprocessors (SM).

controlling and managing it. The memory on both processor types varies by capacity and size. While the GPU is equipped with less memory than a CPU it is faster in transferring data to the processor. But the data needs to be transferred to the GPU with a limited bandwidth of 16 GB s^{-1} for PCIe x16.

4.3.2 GPU Compute Model

In the following, the terms of the GPU compute model are explained [23].

Thread Threads are the smallest execution units of a program. They contain sequential compute instructions and are executed on the compute cores. The execution of multiple threads at a time is called multi-threaded execution (or multi-threading). As described in section 4.3.1, the core execution units of the GPU, called streaming multiprocessors (SM), can be compared to the cores of a CPU. In contrast to the CPU cores, that execute one or two, streaming multiprocessors (SM) on the GPU handle up to several thousand threads at the same time. This is done by making use of the high number of ALUs in SMs and by thread scheduling. The latter enables the switching between threads, when an active thread is waiting for an instruction to be finalized. The processes in the threads are called kernels.

Thread Blocks, Grid The GPU threads are grouped in a three-dimensional array, its elements are called thread blocks. A two-dimensional array of these blocks form a grid, as shown in figure 4.4.
A GPU program is always executed in a grid, so all threads in a grid execute the same kernel function.

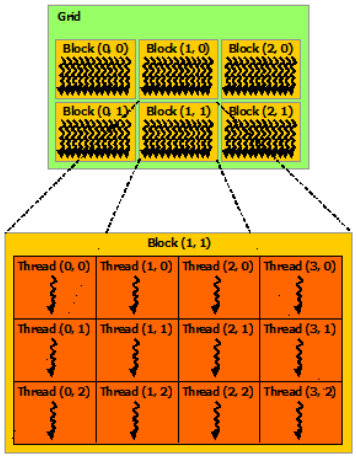


Figure 4.4: Schematic of the structure of threads and blocks in a grid on GPUs [23].

Warps The SMs in a GPU schedule a certain number of threads (16 or 32 depending on the GPU model, 32 for current versions) at the same time. This group is called a warp. It implies that the number of threads in a grid should be a multiple of 32 (16), because the threads missing to the next multiple of 32 are launched anyway, which can also lead to malfunctions (e.g. segmentation violations). A context switch is applied between warps if a warp is idle, with a maximum of 64 active warps per SM at a time to hide latency, especially for memory transactions.

4.3.3 GPU memory

The GPU features two different types of hardware memories, fast, small, low latency on-chip memory and a bigger, but slower off-chip memory, with high latency, implemented as double data rate synchronous dynamic random-access memory (DDR SDRAM). For the use in software, these hardware memory types scatter in different memory types again [26]. Figure 4.5 shows the layout of memory in the GPU and table 4.2 gives the size of the memory types on the NVIDIA GTX 980 GPU.

memory type	size
global, local, constant, texture memory	4 GB per GPU
shared memory	96 kB per SM
register	256 kB per SM

Table 4.2: Memory sizes on an NVIDIA GTX 980 GPU [28].

Registers The registers are allocated for each thread by the SM when launching and reside in the fastest on-chip memory with the lowest latency (just one clock cycle). They are used for variables declared in the kernel code. Since the number of registers is high but limited for each SM, the register usage in kernels has to be considered to allow for as many active threads as possible for the SM (high occupancy of the SM).

Local Memory Local memory is exclusively allocated for a thread and used to spill data that does not fit into registers or arrays for which the indexing is not known at compile time. This memory can use the cache hierarchy as global memory can.

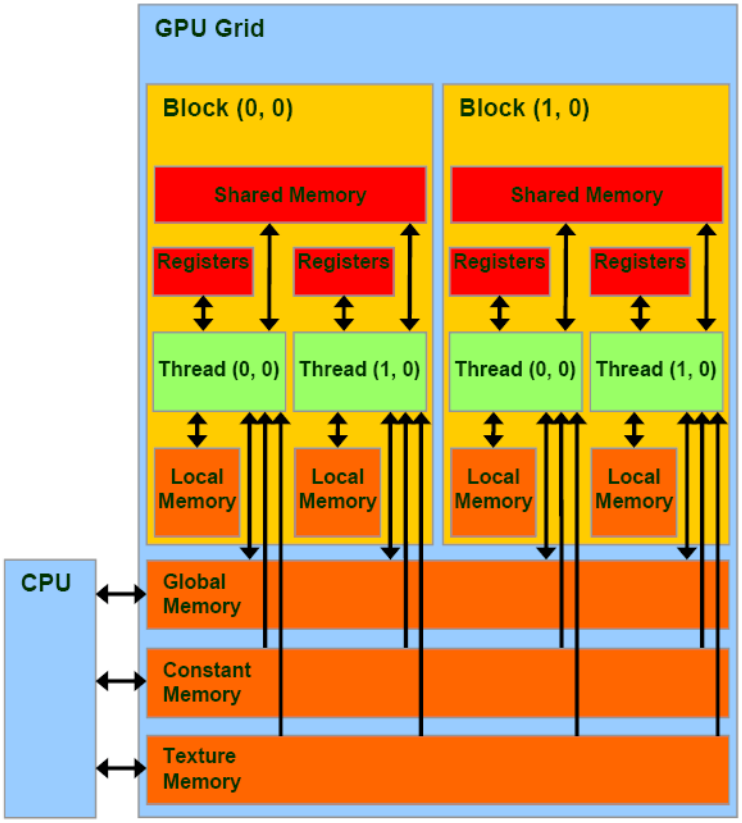


Figure 4.5: Layout of the GPU memory with memory types as available to the programmer. The four memory types, depicted orange in the schematic, reside in off-chip memory, while the rest is on chip [27].

Global Memory The biggest part of the available memory on the GPU is global memory, which uses the DDR SDRAM and can be cached. The whole memory space can be addressed by all threads of the GPU, which makes it necessary to care about data consistency. Since the DDR memory of the GPU is connected via a wide bus (128 bit for the NVIDIA GTX980) the SM always loads four 32 bit values from global memory in one memory clock cycle. If the requested data of consecutive threads resides also consecutively in memory the SM can distribute it to the according threads, as shown in figure 4.6.

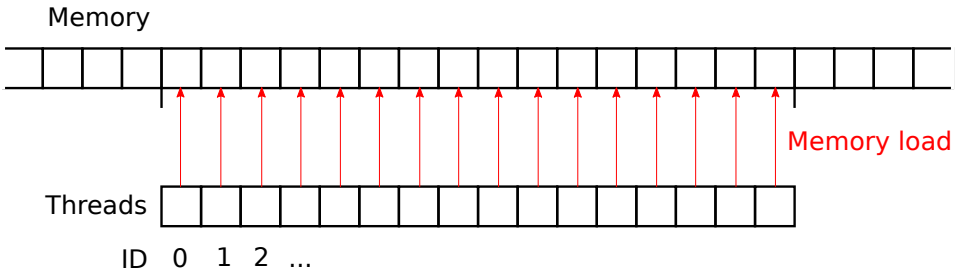


Figure 4.6: Schematic showing the principle of a coalesced memory access from the threads of an SM to memory positions (each rectangle in the top row depicts one memory position) aligned to the thread IDs.

This is referred to as coalescing and is essential for the efficient usage of the available memory bandwidth and especially important for memory bound algorithms.

Constant Memory This is read-only memory located in the same place as global memory, which means it can only be written to by the host CPU. The requests to that memory are cached using a strategy optimized for accesses of all threads to the same memory location.

Texture Memory Texture memory is similar to constant memory but with a cache strategy optimized for a two-dimensional access pattern.

Shared Memory Shared memory is fast on-chip memory in the SM, that can be used to communicate within a thread block. The usage has to be optimized to a high occupancy of the SM, like for registers. It is arranged in banks matching the number of threads in a warp (32 for modern NVIDIA GPUs). All banks can be accessed simultaneously by the threads if bank conflicts are avoided, which is only the case if all threads access the same bank or each thread accesses a bank with the matching ID.

4.3.4 Parallelizable Algorithms

To fit the resources of highly parallel hardware, algorithms have to be parallized. There are tasks that get easily parallelized and pure serial tasks for which it is impossible. The first type is called embarrassingly parallel problem. The latter one typically requires communication between the single threads or the result of the previous step is needed in the next one.

On GPUs, that use the SIMT model an execution step depending on the result of a different step causes branch divergence which results in threads simply waiting for other branches. If the divergence in a warp gets big, the performance is reduced and threads that do not terminate can block the whole warp to finish.

The speed-up of parallel algorithms compared to serial ones is limited by Amdahl's law [29]. It assigns the achievable speed-up for an algorithm to the portion of the algorithm that is fully parallel:

$$S = \frac{1}{1 - P + \frac{P}{S_P}}, \tag{4.1}$$

with the maximal speed-up S, P the fraction of time of the parallelizable part of the algorithm and S_P the speed-up of this fraction. The upper boundary in speed-up is given by P, not by S_P, as even for really high S_P values the total gain is low if P is small. So the parallelizable part of an algorithm needs to be high such that parallel execution is beneficial.

4.3.5 GPU Programming

With the parallel structure of the code and the need to control the GPU's behavior, especially for copying data between the GPU and main memory and starting the functions to be executed (so called kernels), a programming interface to the GPU is needed. There are different application programming interfaces (API) to control the GPU with a program written in a general programming language (e.g. C/C++, Python, Fortran). Despite the vendor specific APIs by AMD and NVIDIA, the main market holders in GPU production, there exists an open source project, OpenCL. Due to the perfect matching between API and hardware, the NVIDIA API, called CUDA, is used in the scope of this thesis for programming of NVIDIA GPUs. Since the company is interested in the GPGPU market, the CUDA API is a commonly used framework in research and other HPC applications. It features an intuitive interface and is expected to map the code efficiently to the GPU hardware. The framework features a set of libraries and programs for implementing GPU programs, e.g. compiler, debugging and profiling tools. The programs for this thesis are written in the C/C++ language with CUDA extensions.

4.4 Numeral Data Types

Since computers work on the basis of binary numbers, all values have to be encoded in a series of bits. For the integers this is done using the binary numeral system. Within this system, there exist signed integers that are symmetric around zero and unsigned ones which only implement zero and positive values. Integer arithmetic is easier to implement than floating point arithmetic, which can be compensated for by special compute units in the processors.
Floating point values are encoded using the IEEE 754 standard [30]. A floating point value encodes a sign, significant and exponent, as for example given in equation 4.2.

$$-1234.5 = \underbrace{-}_{\text{sign}} \underbrace{1.2345}_{\text{significant}} \times \underbrace{10}_{\text{base}}^{\overbrace{3}^{\text{exponent}}} \tag{4.2}$$

Integers are commonly implemented in words with 8 to 128 bits. Floating point values typically use 32 bits for single precision or 64 bits for double precision words. Table 4.3 shows the bit distribution of the components of a floating point word, for the two lengths.

C/C++ type name	word length	sign	exponent	significant
float	32	1	8	23
double	64	1	11	52

Table 4.3: Bit content to encode floating point values after [30].

One can directly see, that most of the extra bits in the double type are spent for an increased precision of the value (52 bits to 23 bits). The rounding required for every result of a floating point computation adds an uncertainty to this value. Therefore, floating point computation is not associative, which can lead to unexpected behaviour, especially for calculations with floating point values with widely spread exponents.
GPUs are developed to perform best on floating types (32 bit), since these are commonly used in graphics rendering. Therefore, the SMs contain more single precision than double precision units and achieve high throughput on single precision addition, subtraction, multiplication and multiply-add ($a \cdot b + c$) operations with low latency [31]. Division as well as double precision and some integer operations (multiplication, division, multiply-add) are disfavoured by the GPU.

Part II

Setup and Measurements

5 The MuPix Telescope

The MuPix Telescope is a beam telescope built to track high rates of charged, low momentum particles in testbeams. It uses the MuPix7 sensor, an HV-MAPS pixel sensor prototype.

5.1 Motivation

On the way to a new pixel sensor, a lot of testing, development and characterisation is needed to understand and develop the desired behaviour and features of the sensor. To characterize and test the HV-MAPS prototypes for the Mu3e experiment a beam telescope was build consisting of four layers of sensor prototypes. Three layers can be used to track particles passing through the telescope while the fourth one is considered as a device under test (DUT). This setup allows for efficiency measurements of the DUT under different conditions, as well as other studies with the prototype sensors, e.g. testing of the sensor readout, alignment and timing analysis. It is used as an integration test before building the first Mu3e Pixel detector module prototypes. Further details about the telescope and results of the measurements can be taken from [32, 33, 18].

5.2 Mechanics

To track straight tracks of beam particles, the telescopes basic setup is a parallel arrangement of four pixel sensors behind each other in the beam direction. To hold the sensors in parallel, allow for spatial adjustments in three dimensions and have a compact and movable setup, a commercial, optomechanical breadboard with compatible rails, pillars and custom-made, printed circuit boards (PCB) with matching holders are used. The whole setup, placed at a testbeam can be seen in figure 5.1.
In various testbeam campaigns the described setup has shown its portability, easy installation and ability to manually align the sensors in the 100 µm regime.

5.3 Electronics and Readout

In addition to suitable mechanics, the MuPix telescope consists of a set of commercially available and customized electronic components to implement the readout and control functions.
The MuPix7 sensor uses a digitization logic in each pixel and a finite-state machine in the inactive sensor part without the need of a separate readout chip (cf. section 3.2.2). A signal in the pixel, that surpasses the threshold value, triggers the storage of a counter value giving the timestamp of that hit. The pixel is inactive until it gets read. The readout state-machine in the sensor periphery cycles through a given readout sequence with a reference frequency (typically 62.5 MHz). Firstly, it pulls one hit from all columns of the pixel matrix to the periphery. Then it reads the hits from the bus consecutively and samples the data for putting it to the output until the bus is empty. After that it pulls the column data again if there are hits left. The data output is zero-suppressed, serialized and 8b/10b encoded [34, 35] with a maximum data rate of $1.6\,\mathrm{Gbit\,s^{-1}}$. While the hits are loaded to the periphery, the chip does not send data, but counter and control words to allow for synchronization on the receiving side.
The sensor is glued to a polyimide foil and bond wired to a PCB. This PCB has a hole at the sensors position, which is covered by the polyimide foil. The PCB allows to connect the chip

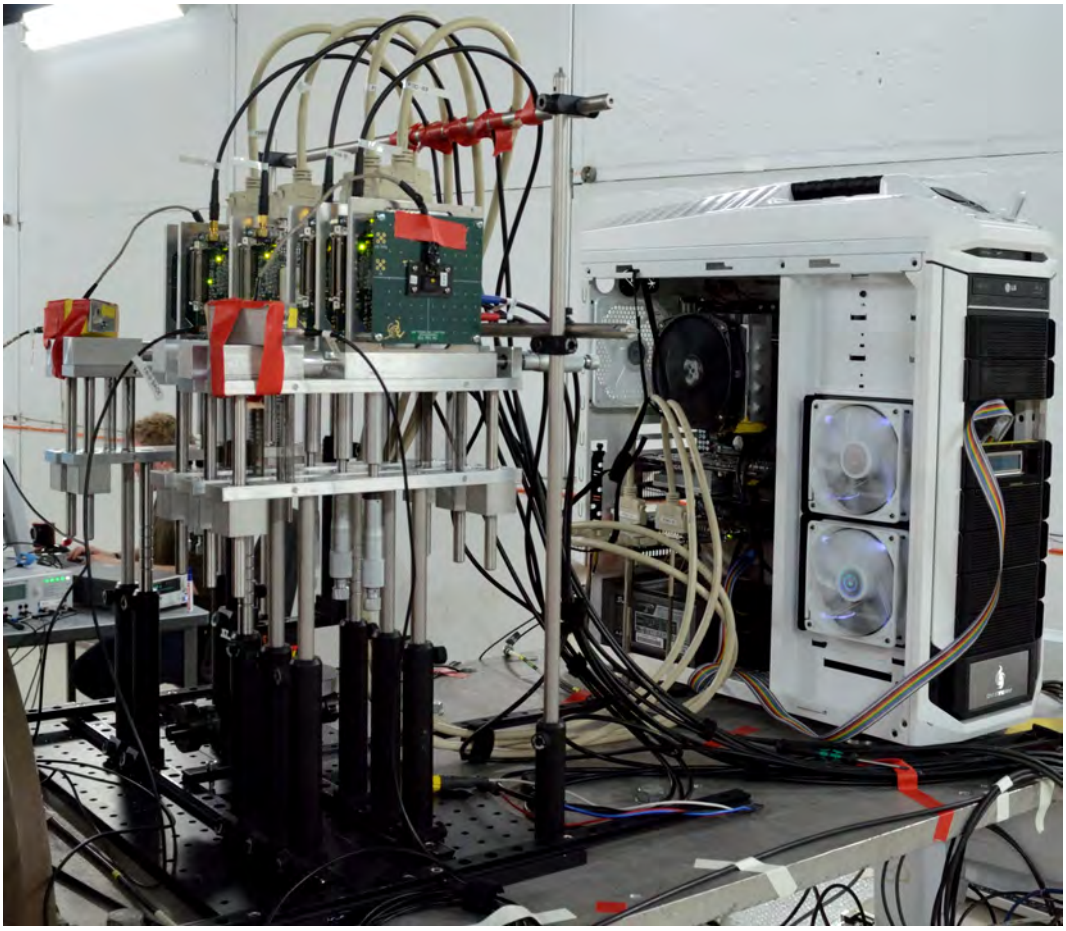


Figure 5.1: Picture of the MuPix telescope set up for a testbeam campaign. The two additional planes in the front and in the back hold scintillating detectors for a trigger setup.

to the periphery including the data connection, electrical power, high voltage and reference voltages by programmable digital-to-analogue converters (DACs). The PCB board with a mounted MuPix7 can be seen in figure 5.2.
LVDS transceivers on the PCB transform the serial data signal and send it to an FPGA via Small Computer System Interface (SCSI) cables. To connect the four SCSI cables of the telescope to the FPGA two custom adapter PCBs for the High Speed Mezzanine Card (HSMC) connector of the FPGA development board are used. Figure 5.3 shows the SCSI cables connected to the FPGA in the readout PC.
A commercial FPGA development board from Altera, featuring a Stratix IV chip and other useful hardware (e.g. memory, interfaces), is used to control the sensors, merge and further process the data of the four pixel sensors. Figure 5.4 shows the flow diagram of the FPGA firmware for the MuPix telescope readout.
The incoming data is 8b/10b decoded, unpacked and merged into one data stream. The data is not read out time sorted, so that later hits on the sensor can arrive earlier in the FPGA. Therefore, a hit sorting algorithm, using the hit timestamps, is implemented in the FPGA firmware [36]. It is implemented by a segmented ring buffer in FPGA memory assigned to the full range of the 8 bit hit timestamp. The write address to the memory is given by the timestamp of a hit and a 4 bit counter, allowing for a maximum of 15 hits per timestamp. A block is defined as a part of this memory for a certain number of timestamps assigned to it. The hits are written to the memory according to their time information and read by

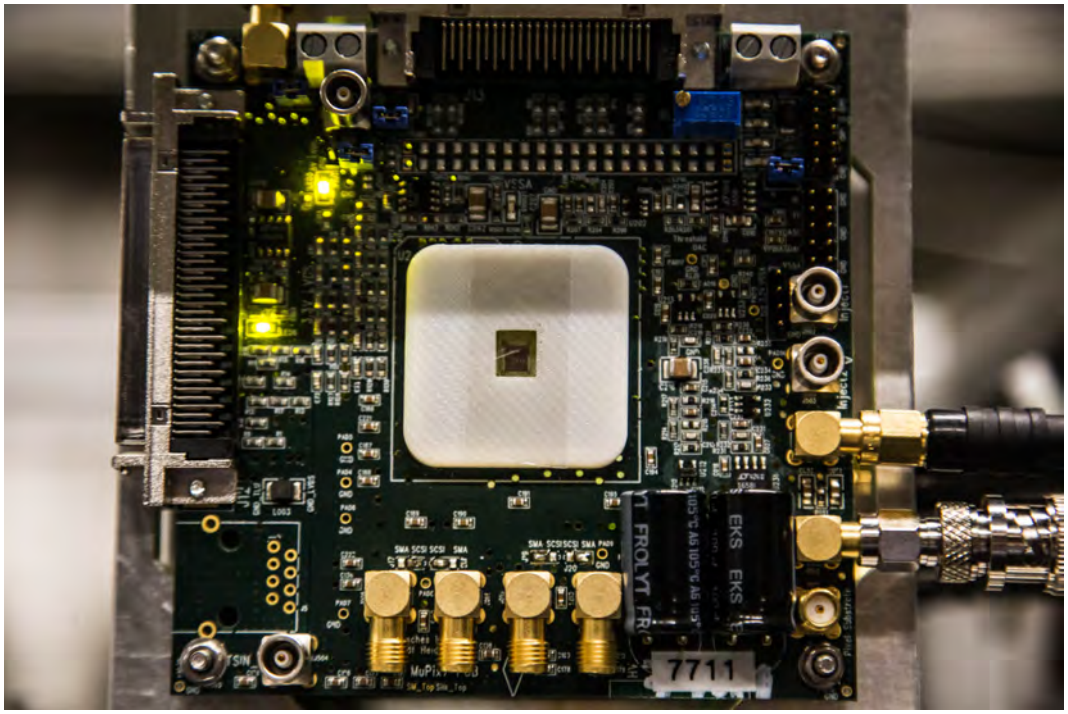


Figure 5.2: Picture of a MuPix sensor prototype on PCB mounted to the aluminium frame of the holder. The cables on the right connect the low voltage ($-5\,\mathrm{V}$) and high voltage ($\mathcal{O}(-80\,\mathrm{V})$).

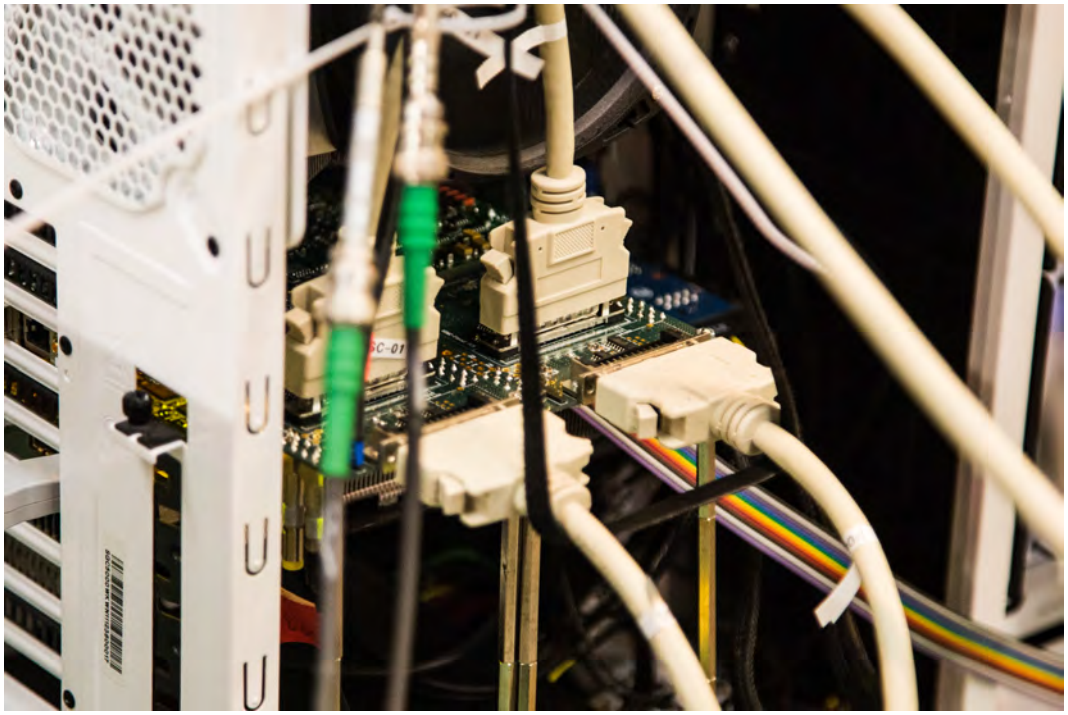


Figure 5.3: Picture of the four SCSI cables (beige) connected to the two adapter boards and the FPGA development board in the readout PC.

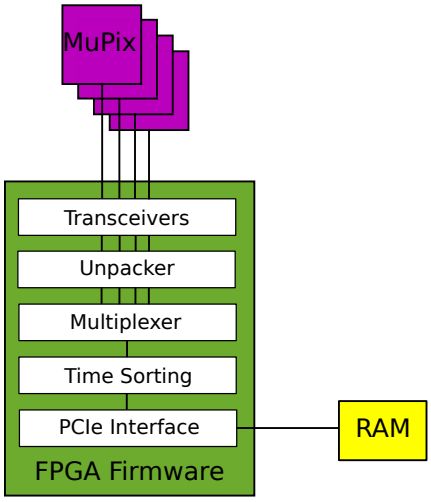


Figure 5.4: Flow diagram of the FPGA firmware for the MuPix telescope.

circulating through the buffer to get a time sorted data output. Read and write processes on the same block are avoided by allowing only written blocks to be read, as depicted in figure 5.5.

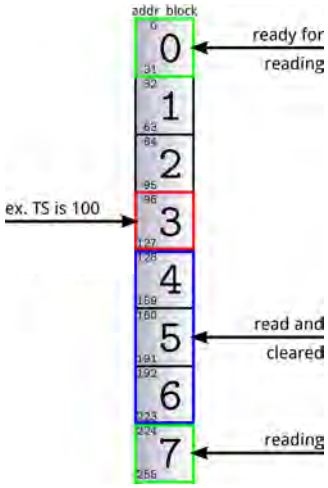


Figure 5.5: Schematic of the ring buffer implemented in the FPGA firmware to sort the incoming hit data according to their timestamps. The red block is currently written to because the incoming hit has timestamp (TS) of 100 and the block range is $96 \leq TS \leq 127$. the blue blocks are read and ready to write to and the green ones are ready for reading. Adapted from [37].

This is complex because after hits have been written to a certain block, hits can arrive that belong to blocks before that. Another problem occurs with overflowing the available space for one timestamp. This happens very rarely and the overall efficiency of the time sorting algorithm is above 99 % for beam rates up to 300 kHz. After the hits have been sorted they are transferred to the main memory of the readout via PCIe. Therefore, the firmware contains a PCIe interface to control the communication on the PCIe bus. Together with a custom driver for the PC's operating system the data transmission can use DMA (cf. section 4.1.2). The readout scheme for the telescope is shown in figure 5.6.
The readout PC is equipped with high-performance consumer hardware. An Intel Core i7 CPU is combined with an NVIDIA GTX 980 graphics card (see table 4.1 for some essential

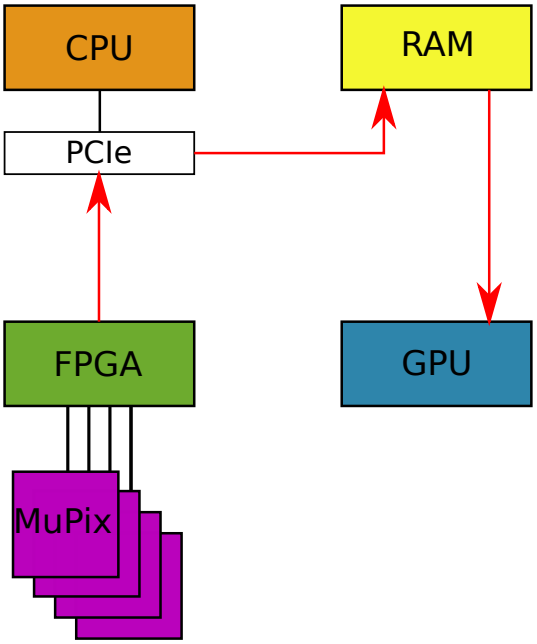


Figure 5.6: Schematic drawing of the MuPix telescope readout. Red arrows indicate DMA transmissions.

hardware specifications) and 16 GB DDR4-SDRAM main memory. To store measurement data 6 TB of hard disc drive space is available. It runs a Linux based operating system with custom control, readout and analysis software for the telescope. Having the data in the main memory allows for a variety of operations on it, implemented in software, e.g. monitoring, analysing and writing it to data storage devices. It also enables the make use of the GPU for online processing, as discussed in chapter 6.

5.4 Tracking

Tracking denotes the process of finding tracks in a series of hits from different detectors and fitting a track model to them. The track model used in the scope of this thesis is a straight track, which can be described by the following linear function with $\vec{x}(z)$ describing the x and y position as a function of the z position [32].

$$\vec{x}(z) = z \cdot \vec{a} + \vec{x}_0 \tag{5.1}$$

The parameters $\vec{a}$ and $\vec{x}_0$ describe the two dimensional slope and offset of the track function. This simple model is sufficient to describe the particle's trajectory through the telescope because the particle track is not bent by any electric or magnetic field and the pixel sensors are very thin (< 1.0 ‰ of a radiation length) reducing the influence of multiple scattering. The tracking scheme takes the hits from four planes of the telescope for a given time interval (frame) and combines all possible combinations of them to track candidates. For each track candidate it fits the track model and evaluates the χ^2 value (equation 5.2), which represents the sum of the squared residuals. Ignoring the multiple scattering effects here, the statistic is not expected to follow a real χ^2 distribution for the number of degrees of freedom. The χ^2 value of a track with parameters a_x and x_0 in x-direction and a_y and x_0 in y-direction is calculated by

$$\chi^2 = \sum_{i=1}^{n} \left(\frac{\Delta x^2}{\sigma_{x_i}^2} + \frac{\Delta y^2}{\sigma_{y_i}^2} \right) = \sum_{i=1}^{n} \left(\frac{\left(x_i - (z_i \cdot a_x + x_0)\right)^2}{\sigma_{x_i}^2} + \frac{\left(y_i - (z_i \cdot a_y + y_0)\right)^2}{\sigma_{y_i}^2} \right) \tag{5.2}$$

with the number of planes n, the residuals Δx, Δy and pixel resolutions $\sigma_{x_i} = \frac{\text{pixelsize}_x}{\sqrt{12}}$, Δy, $\sigma_{y_i} = \frac{\text{pixelsize}_y}{\sqrt{12}}$.

Track Fit To find the best estimate for parameter values of a track given a set of n hits from n planes of the telescope the χ^2 value needs to be minimized. The derivatives of 5.2 with respect to the track parameters give a set of linear equations.

$$\begin{aligned}
\frac{\partial \chi^2}{\partial x_0} &= \sum_{i=1}^{n} \frac{2}{\sigma_{x_i}^2} \cdot (x_i - (z_i \cdot a_x + x_0)) \\
\frac{\partial \chi^2}{\partial a_x} &= \sum_{i=1}^{n} \frac{2}{\sigma_{x_i}^2} \cdot z_i \cdot (x_i - (z_i \cdot a_x + x_0)) \\
\frac{\partial \chi^2}{\partial y_0} &= \sum_{i=1}^{n} \frac{2}{\sigma_{y_i}^2} \cdot (y_i - (z_i \cdot a_y + y_0)) \\
\frac{\partial \chi^2}{\partial a_y} &= \sum_{i=1}^{n} \frac{2}{\sigma_{y_i}^2} \cdot z_i \cdot (y_i - (z_i \cdot a_y + y_0))
\end{aligned} \tag{5.3}$$

Setting the derivatives to zero for minimization and rearranging, the equations give the following matrix form.

$$\begin{pmatrix} \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} (z_i \cdot x_i) \\ \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} (z_i \cdot y_i) \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} z_i & n & 0 & 0 \\ \sum_{i=1}^{n} z_i^2 & \sum_{i=1}^{n} z_i & 0 & 0 \\ 0 & 0 & \sum_{i=1}^{n} z_i & n \\ 0 & 0 & \sum_{i=1}^{n} z_i^2 & \sum_{i=1}^{n} z_i \end{pmatrix} \cdot \begin{pmatrix} a_x \\ x_0 \\ a_y \\ y_0 \end{pmatrix} \tag{5.4}$$

The inverted matrix gives an analytic expression for the track parameters. This gives a fast, analytic algorithm to reconstruct straight tracks from a set of telescope hits.
The χ^2 value is a good estimate for the quality of the track fit and is used to select real tracks from all the track candidates. Track candidates with a small χ^2 value are considered a good description of the real track. While looping through all track candidates in a certain frame, the algorithm decides whether a candidate is rejected or not by the following criteria. The χ^2 value has to be lower than a certain threshold and hits of this candidate have not been part of another track before. This requires for a sorting algorithm on the χ^2 value of all track candidates, which can be time consuming. An easy and fast shortcut, is to allow only one track per time-frame, which is applicable because the timestamp rate is high (typically 31.25 MHz) compared to the particle rate of the beam lines, so that the possibility of multiple tracks per frame is small.
A coordinate system was chosen with the z-axis parallel to the beam particle's direction of flight and x- and y-axis along increasing column and row addresses of the chip, respectively.

6 GPU-based Tracking for the MuPix Telescope

The MuPix telescope is a very useful tool to test the MuPix sensor prototypes but it can also be seen as the first test of a multiple sensor system leading to the Mu3e pixel detector. In this sense it can be used to set up and test different parts and techniques for the Mu3e detector readout. Many crucial aspects of the readout system can be tested before having built the whole detector in order to save time by developing the pixel detector and its readout system in parallel. One of these aspects was to validate the communication between an FPGA and a GPU, as needed for the online track reconstruction and event filtering in the Mu3e filter farm PCs. The MuPix telescope was used as a testbench for the implementation. Since the geometry of the final experiment and the telescope differ a lot, it is a completely different reconstruction algorithm to be implemented, but with the online tracking for the MuPix telescope different parts of the readout system can be developed and tested. This includes data transmission from the pixel sensors to the readout FPGA as well as soft- and firmware for the communication between FPGA, GPU and CPU.

6.1 Memory and Compute bound algorithms

Algorithms implemented on a computing device can be categorized by their characteristics in terms of resource usage. The two main resources of a compute processor are memory space to store the data set needed for the algorithm and the computation units to act on the data. The categorization is commonly done in memory- and compute bound algorithms. It denotes whether an algorithm's performance is limited by the memory space and bandwidth or the computation resources it uses. Thus, the optimization of an algorithm implementation has to target these primary bottlenecks. The optimum would be reached if the algorithm is right in the middle of the two special cases and fully utilizes both resources equally.
To shift a memory-bound algorithm to the optimum, one has to increase the compute load on a given data get loaded from memory. For a fixed instruction set of an algorithm this can be reached by, e.g. increasing the data bandwidth by improved access patterns, reducing the size of the data set and efficiently using the fast memory like cache. For compute-bound algorithms it adds up to more efficient instruction sets, exchanging computations for memory interactions or reducing the computation count by re-usage of intermediate results. To apply such optimizations a profound understanding of the hardware structure and the compute model on the target system is required as well as of the bottleneck of the algorithm.

6.2 Parallelization of the Telescope Tracking

The tracking algorithm needs to cover the following main steps for one track candidate: loading the hits from memory, transform the hits from a local to a global coordinate system, fit the track, calculate the χ^2 value and compare it to the best value in the frame so far to store the best track candidate.
In general, a lot of approaches can be chosen to parallelize an algorithm to efficiently execute it on a GPU. Since the execution on a GPU is done following the single instruction multiple thread (SIMT) scheme, as described in section 4.3, one has to find a compute task that is repeated multiple times on different sets of data. This task becomes the content of a thread on the GPU and the different data sets have to be stored in the GPU's memory. Clearly, the fitting of tracks to hits from different frames of the MuPix telescope, as described in section 5.4, can be parallelized this way.

6.2.1 Algorithm Implementation

The main idea of the parallelization of the track fitting algorithm is to fit all possible hit combinations of a time frame of the telescope in one thread, which means to fit a lot of frames in parallel on the GPU. Since it does not require any communication between the threads, this gives an easy, reliable and fast parallel implementation of the tracking for the telescope. Another approach is the fitting of only one track candidate in the scope of a thread, which is disfavored because it would increase the overhead for the thread handling (more threads need to start with less computations per thread) and requires a sorting of the χ^2 values after all track candidates for a frame have been fitted. Other parallelization schemes considered were expected to require either too complicated thread scheduling, memory access patterns, inter-thread communication, pre-processing of the data or sorting algorithms for a first implementation of the whole system. Besides performance considerations, the chosen scheme was also kept simple to retain deep insight in the first implementation of the chain containing the MuPix7 sensors, adapted FPGA firmware, PCIe driver for DMA and the GPU software. A thread of the GPU loops through all combinations of hits in a frame and calculates the χ^2 value for each track candidate (cf. paragraph 5.4) and compares it to a threshold value and the best value already obtained from the candidates fitted before in this frame. After all combinations are tested, the residual and χ^2 values are returned to the global memory of the GPU. To allow for an efficient usage of the available memory bandwidth the data is sorted to achieve a coalesced memory access, as described in section 4.3.3. In the following, the characteristics of the GPU tracking implementation are discussed, comparing the advantages and disadvantages.

Characteristics The main characteristics of the parallelization scheme are a simple layout of one frame per thread with a fixed amount of resources and the omitted communication between the threads. It was chosen such that it meets as many aspects of the performance guidelines in the NVIDIA programming guide [23] as possible.

Performing the track fits for all combinations of the hits of one frame in a thread gives the maximum amount of computation tasks on the same set of hits, which allows for an efficient use of the memory bandwidth, as described in section 6.2.2. This is a favoured characteristic because memory interactions have a high latency ($\mathcal{O}(500)$ clock cycles) compared to compute tasks, especially for 32 bit floating point values ($\mathcal{O}(30)$ clock cycles [31]).

The resources (e.g. number of threads, blocks and memory size) for the track fitting are constant during the execution to reuse them efficiently. Therefore, an execution of a fixed number of threads and the associated memory interactions between GPU and CPU memory are enclosed in a so called stream. Multiple streams can be scheduled simultaneously, which allows for copying the data from the CPU main memory to the GPU for a set of frames, while the computation of the frames before is currently done on the GPU. All resources assigned to a stream are completely exclusive for this stream. Figure 6.1 shows the usage of streams in the GPU tracking algorithm.

The streams are cycled in a ring structure with increasing time, so after all streams are executed, the first one is used again. A ring buffer in CPU RAM, used to accumulate data sets from the FPGA, is split in four parts. Each part of this buffer is assigned to a specific stream. One part is written to by the FPGA while another one is copied to the GPU. These two active parts have a stride of one, so the parts in between are inactive and ensure data integrity, so that memory regions read are not written to at the same time. The size of memory accessed by a thread is also kept constant, because it allows for reusing the allocated memory for the next cycle in this stream. The memory not needed for hit data in a frame is assigned with a special value by the FPGA, which can not occur in real data.

The communication between threads is prevented since tracks that are just reconstructed within a frame. Thus a specific memory entry is not used by different threads, which allows for coalesced memory transactions and worries about data integrity and overhead from inter-thread communication are not an issue.

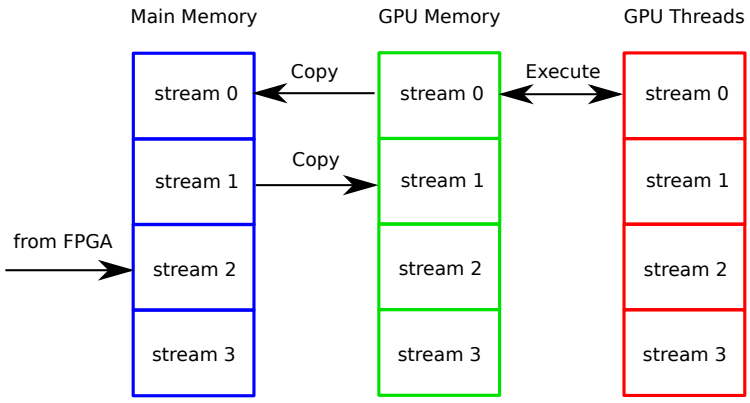


Figure 6.1: Principle of stream usage in the GPU tracking algorithm. Data comes from the FPGA and is written to main memory, e.g. to the part associated with stream 2. At the same time data is copied from main memory to GPU memory in stream 1 and stream 0 executes the fitting and writes back the data.

Disadvantages The downsides of the implemented parallelization of the telescope tracking for the GPU, with one frame per thread, are mainly the need for a pre-sorting of the telescope hits, overhead, as well as overflow from the constant problem size, ignoring tracks over frame boundaries and the number of tracks per frame being fixed to one.
The pre-sorting can not be easily come around, it is simply needed for a track reconstruction, either in software or firmware.
The same holds for the fixed size of the memory to store the hit data, which could potentially improve the efficiency of resource usage (memory space and data bandwidth) but requires a sophisticated algorithm to assign the memory to the threads each time a thread is launched. Tuning the available memory space for the frames, meaning the maximum number of hits in a frame, and the frame length in time should be suitable to meet the performance requirements. The optimal setting should achieve minimal memory space and avoid overflow in the frames for a specific hit rate on the sensors.
Depending on the frame length the ignoring of tracks that cross the frame boundary (e.g. timestamp counter on the chips switches while a passing particle is between two sensor planes) can significantly reduce the tracking efficiency by up to 50% [38]. Longer frames (containing higher number of hit timestamps) increase the efficiency but also increase the combinatorics. Optimizing the frame length together with the aforementioned memory size can potentially lead to a suitable tracking efficiency. The tracking efficiency could also be optimized by the use of overlapping frames, which means that more than one timestamp is considered with the first ones already used in a frame before and the last ones in the next frame. It would require a communication between the threads to prevent assigning a hit to multiple tracks. Allowing for only one track per frame opens the possibility to loose real tracks in the reconstruction. As discussed in 5.4 this is low. More tracks reconstructed per frame require a comparison of the track candidate with the previous ones and one has to ensure assigning hits to only one track. This requires additional memory and compute effort for the sorting algorithm.

6.2.2 Memory

Since the track fit for straight tracks in the telescope has an analytic solution and requires relatively few computations per track candidate, the fit algorithm is memory bound. Thus, the implementation was chosen such that it allows for an efficient use of the wide memory interface of the SMs to the GPUs RAM. As discussed in section 4.3.3, this is possible if consecutive threads request or write data from/to consecutive memory addresses at the same time. The idea is that different threads load hits from different time frames simultaneously

and write back the results the same way. So the hit data of a frame resides in strides of the number of frames in the global memory of the GPU, as shown in table 6.1.

memory address	0	1	2	...	31
hit.plane.frame	0.0.0	0.0.1	0.0.2	...	0.0.31
memory address	32	33	...	...	63
hit.plane.frame	1.0.0	1.0.1	...	...	1.0.31
...					
memory address	256	257	...		
hit.plane.frame	0.1.0	0.1.1	...		

Table 6.1: Address scheme for the hits in GPU global memory for 32 frames.

First, all first hits from the first plane of a number of frames are placed next to each other. Then the second hits follow and so on for the next planes. The column and row addresses, representing the x- and y-direction, are completely separated, creating two non-overlapping memory regions for the two axes. The number of frames in this scenario has to be a multiple of the warp size (cf. section 4.3.2), since threads in a warp execute memory interactions simultaneously. The data layout for the GPU tracking requires a sorting of the hits before the tracking can start. It is implemented on the FPGA, as discussed in section 6.3.1.
Hits are stored in the memory and transferred to the GPU in a raw format that encodes the row and column of the pixel matrix in 8 bit unsigned integer values each. To obtain physical values for the hits, the pixel pitch is multiplied to it at the beginning of the kernel function, as well as the plane distances and offsets as specified by the user.

Constant Memory Being read-only, constant memory is used for the geometry data, namely offsets of the planes in x and y-direction, the position of the planes in z-direction, the pixel pitch and a rotation of the DUT along the y-axis. It is written to the GPU at the initialisation of the GPU online tracking, along with the allocation of global memory and creation of the streams.

Registers The thread registers on the GPU are used to store four hits of a track candidate, intermediate values of the track fit and loop variables. Since the amount of registers is limited per SM the NVIDIA compiler can optimize the usage by moving it to off-chip memory and some values of the track fit algorithm are stored in shared memory instead of registers. Spreading the variables more widely over the available hardware resources allows the SM to schedule more warps simultaneously and thus increases the performance of the GPU tracking, as described in section 4.3.2.

6.3 Readout and Software Concept

The existing software and firmware package to control and readout the MuPix telescope has been modified to handle the GPU online track reconstruction. To this end, a sorting algorithm has been added to the FPGA firmware. A custom device driver to control the DMA engine has been used and a software to merge these functionalities with the GPU algorithms has been developed.

6.3.1 FPGA Firmware

The firmware of the FPGA in the readout PC was adapted to deliver the required data structure for the GPU track reconstruction. Therefore, in addition to the time sorting of the incoming hits, a sorting by the sensor plane was implemented. The plane sorting algorithm uses the time sorted data stream (cf. section 5.3) before it is sent off to the PCIe

interface. Two finite-state machines (FSM) are implemented to sort data in a dual-port random access memory (RAM) on the FPGA. Dual-port memory allows for read and write accesses on different memory addresses at the same time. One of the FSMs controls the writing to and the other one the reading from the RAM, as depicted in figure 6.2.

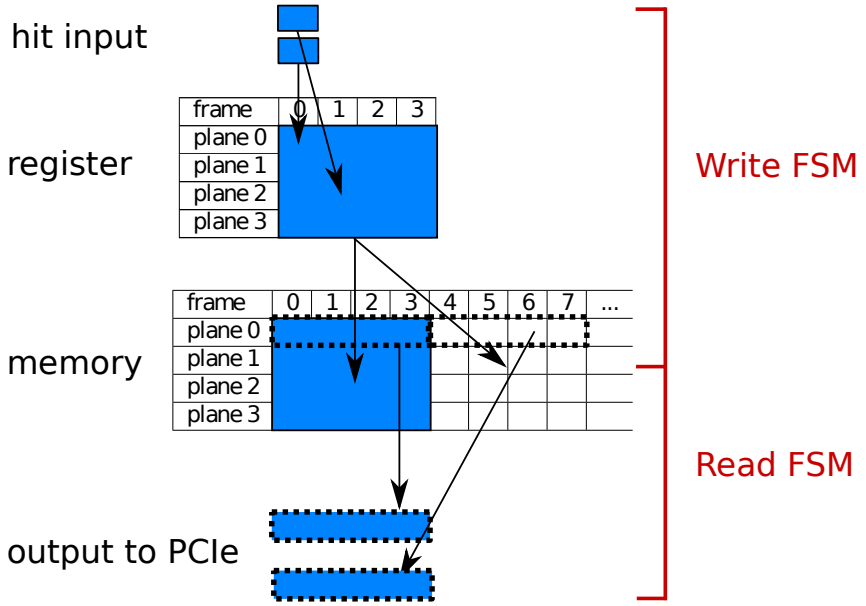


Figure 6.2: Simplified schematic of the working principle of the sorter FSMs. The time-sorted hits are stored in a register sorted by plane and filled in frames. The register takes four complete frames (size: 1024 bit) before it is written to the sorter memory in one write clock cycle. When a block in memory (32 frames) is completely written, the read FSM reads the hits from one plane in the first four frames (frame 0-3) and sets it to the output. In the next read clock cycle it reads the next frames (frames 4-7) and so on until the boundary of 32 frames is reached. Then the next plane is read.

Write FSM The writing part sorts the data by plane and time-frames and splits the data in column and row addresses. It receives the time-sorted data in 32 bit words for one hit and writes four frames to a register, which is written to RAM when it is full and initialized with a special word for empty hits. This special data word is recognized in the GPU later as the end of valid hits in a frame. It is needed because the hit count varies while the memory size is constant all the time. The register size is fixed to 8 hits in a plane per frame. Overflow is recognized and sets a flag readable by the readout software. Frames with less than 3 hits in the four planes are not useful for any track reconstruction, which is why they are thrown away.
The writing FSM uses a 125 MHz clock. In every clock cycle a new hit can be on the input. To enable writing of the complete register, it uses a 1024 bit wide memory interface.

Sorter Memory The RAM the sorter uses is divided in two parts for the column and row addresses. These parts are used as ring buffers with four blocks containing data for 32 frames.

Read FSM The read FSM loops through the ring buffer with a stride of four, the number of telescope planes, to obtain the structure described in section 6.2.2. It reads data from one plane but four frames in one clock cycle, continues on the next frames and switches the

planes after 32 frames have been transmitted. On the reading side the interface width is 128 bit running at 250 MHz frequency. The clock is delivered by the PCIe interface and also drives the DMA algorithm on the FPGA.

6.3.2 Device Driver

The FPGA controls and reads out the telescope. To access the control and readout functions from the PC an interface is needed. This is done with the help of a custom device driver for a Linux-based operating system. To communicate with the FPGA it allows to write to and read from registers via the PCIe link. The driver was available from the telescope software package already.
The driver handles the DMA for the data transmission. Basically, it allows to allocate memory that is accessible by the GPU and transfers the virtual memory addresses, as seen by the user programs, to physical addresses that represent the real position in the memory hardware. Since contiguous virtual memory is scattered to different physical addresses, the driver acquires a table that holds the starting addresses and sizes of the physical memory sections. The DMA engine in the FPGA gets this table during the initialization process, so that it can directly write to the physical addresses. Since the DMA happens without interference of the CPU, interrupt messages are sent to the processor. They inform the processor about the copy process. Every time 256 kB of data have been sent from the FPGA to the main memory an interrupt message is sent. The driver handles the interrupt messaging and informs a user program about the current memory position, the last 256 kB have been written to. The DMA functionality of the driver was developed in the scope of a PhD thesis [39].

6.3.3 Software

A program controls the MuPix telescope, starts the GPU online tracking and writes the data to storage at a testbeam. It was derived from the existing telescope software. The software makes use of the driver functions to set registers in the FPGA to control the voltage references of the MuPix sensor and the readout functionality [32]. In the initialization phase it sets all the essential parameters for the telescope sensors, readout and the GPU tracking. Also the allocation of the CPU ring buffer and the GPU memory is triggered before the start of the tracking. The streams described in section 6.2.1 loop through the following points until the run is stopped:

- FPGA writes data to the current part of the ring buffer in CPU RAM using DMA.
- After 32 interrupt messages arrived this memory region is approved to get copied to the GPU.
- GPU tracking kernels are started.
- Results are copied back to CPU RAM and stored in a binary file.

A program to write the recorded data into ROOT [40] histograms is used to monitor the data.

7 Simulation Studies

In addition to the studies in context of the testbeam campaigns (cf. chapters 8 and 9), the GPU tracking for the telescope has been tested with the use of simulated data sets. Being independent of recorded data from testbeam campaigns opens the possibility to test the limits of the GPU tracking implementation. It was used to study the execution times with fully filled frames to explore the performance of the GPU tracking and to evaluate the influence of 32 bit floating point values in the GPU tracking on the precision of the calculations.

7.1 Execution Time

With simulated hits that always fill the number of hits per plane in a frame n_{hits}, the time to execute the GPU tracking was measured. The execution time here and in the following is the time to copy the hit data from CPU main memory to the GPU, execute the GPU tracking kernels and copy back the results to the main memory. The results contain the residuals of the four planes (cf. section 8.2.1) and the χ^2 value for one track candidate per frame.
Figure 7.1 shows an exemplary measurement of the execution time per track candidate against n_{hits}.

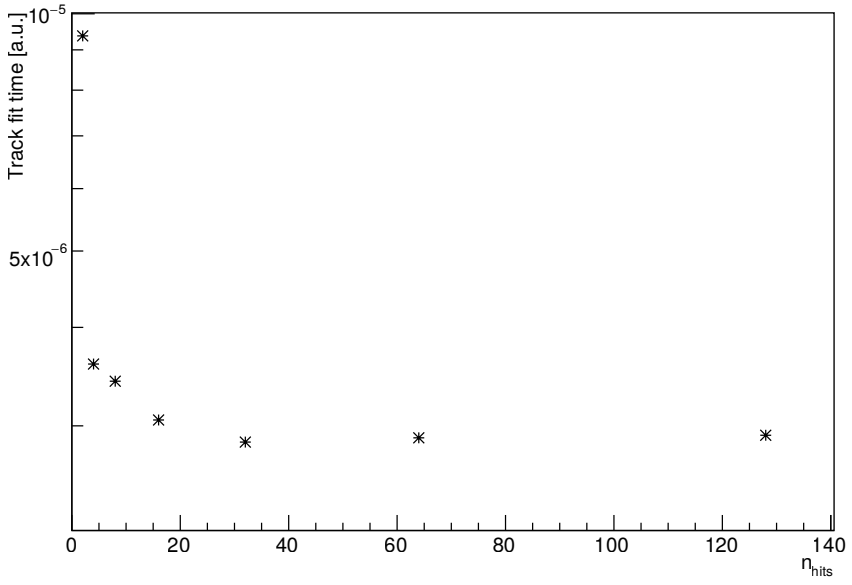


Figure 7.1: Plot of the execution time for one track candidate depending on the number of hits per plane in one frame n_{hits} using the GPU tracking.

The time converges to a lower limit for high n_{hits}, which suggests that the execution time is determined by the pure computation of the track fit. For $n_{hits} < 20$ the execution lasts longer since the portion of memory interactions on the total time increases significantly. According to section 6.1, the first case is compute-bound and the second is memory-bound.

The implementation for the testbeam campaigns mainly reside in the memory-bound regime (see figure 8.7 for the distribution of n_{hits} in the DESY data set.)
The maximal achieved rate for fitting one track candidate with $n_{hits} = 8$ is 670 MHz.

7.2 Floating Point Precision

The calculated χ^2 of the tracking algorithm is used to compare 32 bit bit floating point computation to the 64 bit case. For a GPU application the use of single precision values is favoured for performance reasons (cf. section 4.4). The effect on the precision of the resulting value is shown in figure 7.2, which shows the difference of the resulting χ^2 values computed with single and double precision variables on the GPU depending on the absolute value.

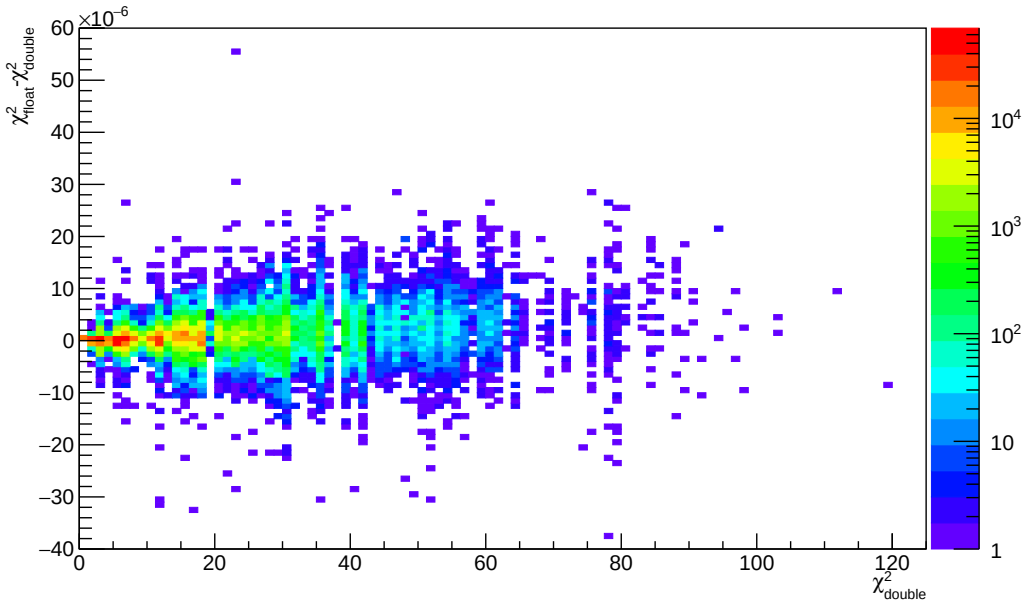


Figure 7.2: Differences in the resulting χ^2 values for the GPU track fit algorithm implemented using 32 bit floating point variables (float) versus 64 bit (double) applied to simulated data. The y-axis shows the difference, x-axis the absolute value (1 corresponds the size of a whole sensor) and the color denotes the number of entries.

The width of the distribution seems to increase for higher absolute values. This is expected since the significant in floating point values has a fixed length and the precision of a value is proportional to the value itself. The deviation of the single precision value to the double precision is on the 10^{-6} level. It suggests that the usage of 32 bit bit floating computation for the GPU tracking has no significant impact and can be used.

8 DESY Testbeam

In October 2015 the MuPix telescope was used at a one week testbeam campaign at the DESY II synchrotron in Hamburg (Germany). In DESY II electrons or positrons are accelerated to an energy of up to 7 GeV. To generate the secondary beam for the testbeam facility, Bremsstrahlung is created at a 7 µm carbon fiber target in the synchrotron and the photons are then converted to electrons and positrons in a secondary metal target. A dipole magnet followed by a collimator selects 1 to 6 GeV electrons to be conducted to the testbeam area. The whole electron production process for the testbeam at DESY II is illustrated in figure 8.1.

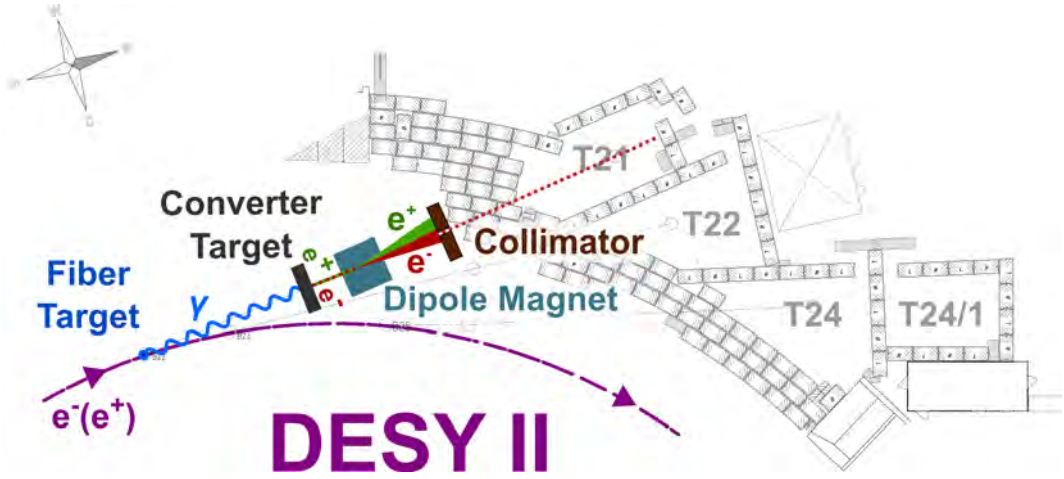


Figure 8.1: Schematic view of the DESY II beam generation for testbeam areas [41].

8.1 Setup

During the testbeam campaign, the MuPix7 prototypes were tested in a single setup and two telescopes. One telescope was mainly used for efficiency measurements and the implementation test of the GPU tracking, while the second telescope ran in parallel, placed behind the first one, to test the long term reliability of the whole telescope setup. Figure 8.2 shows the picture of the setup with the two telescopes.
To reduce the influence of multiple scattering the first three planes contained chips thinned to between 75 µm and 50 µm and all planes were placed as closely spread as possible (c.f table 8.1).

Layer	z-position [mm]
1	0
2	62
3	175
4	256

Table 8.1: Z-positions of the four planes for the DESY testbeam setup

A first version of the GPU tracking algorithm had been implemented for a first proof of principle run. Since the firmware to create frames for the GPU tracking was not yet ready,

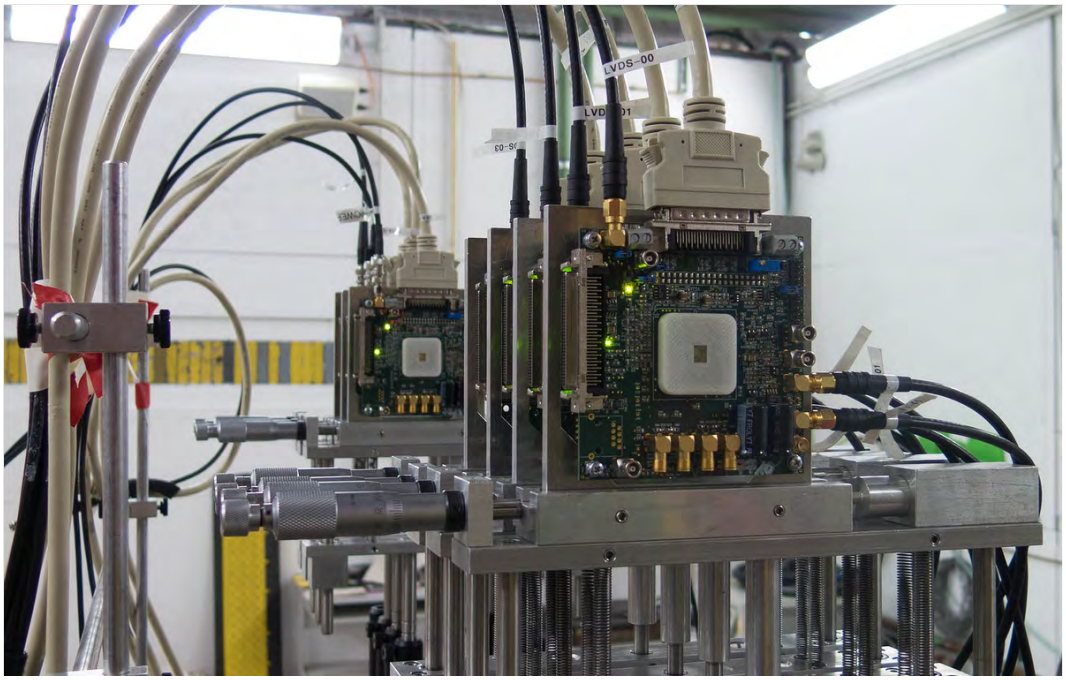


Figure 8.2: Picture of the two telescopes set up during the DESY testbeam campaign in October 2015.

software running on the CPU was used to sort the hit data according to the data structure for the GPU tracking as described in section 6.2.2. For the DESY testbeam this is not critical because the beam particle rate is fairly low ($\mathcal{O}(100\,\mathrm{Hz})$) and thus the sorting algorithm does not put a high load to the CPU. During the measurement, some issues have been discovered in the implementation of the track fitting algorithm on the GPU by comparing the results with the output of the implementation on the CPU, which is used for the analysis of the telescope data. These issues could be identified and fixed in a debugging process after the testbeam, using recorded data.

8.2 Analysis

The telescope data taken during this testbeam campaign was used to improve and study the track fitting algorithm on the GPU. Therefore, a specific run with a set of parameters for the sensors with a good efficiency and low noise to not create fake tracks was used. To validate the implementation of the track fitting algorithm on the GPU, the results were compared to the ones given by the implementation from the CPU analysis software. Another interesting aspect is the performance gain by using the GPU over the CPU for the same track fit algorithm. This is a rather unfair comparison because the analysis software is not optimized in any way, especially as it is not using multithreading methods while the programming of the GPU code was driven by performance considerations. In the following, the results of this offline analysis are shown.

8.2.1 Residual Distributions

A residual is the difference of a hit to a reconstructed intersection on the plane. The distribution of the residuals is used to compare the track fit implementations on the GPU and CPU. Figure 8.3 shows the distributions for both implementations with the same parameters:

- The same data file (run 313) for both with equal number of frames

- One track per frame (length: 32 timestamps · 32 ns ≈ 1 µs)
- maximum number of hits per frame and plane: 128 (only 8 in the final implementation with shorter frames)
- No cut on the χ^2 value
- 4 planes considered for the track fit

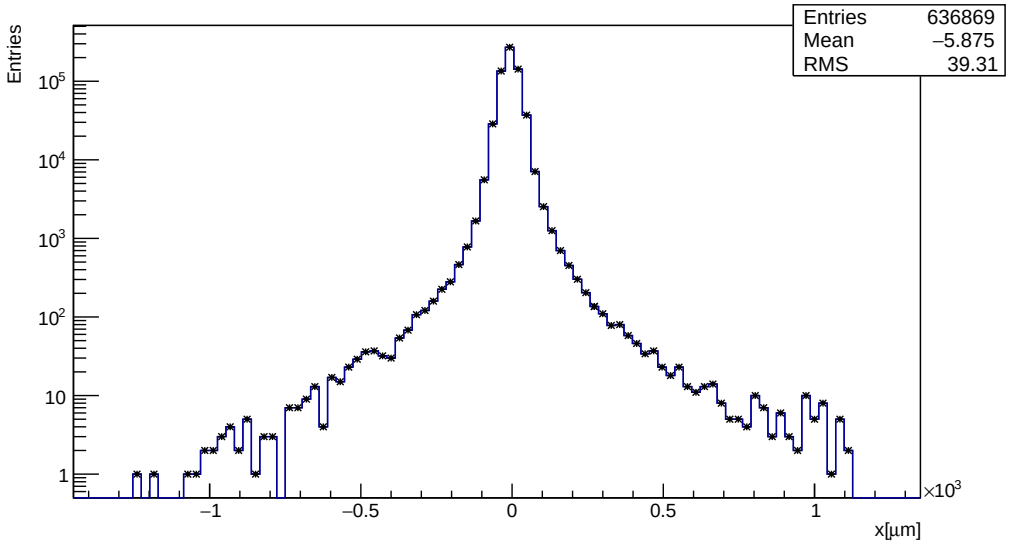


Figure 8.3: Residual distribution in x-direction of the first plane for GPU and CPU implementations of the telescope track fitting algorithm. The blue line is for the GPU version and black dots denote the CPU version.

By eye both distributions are identical, with the same binning, which implies the correct implementation of the track fitting algorithm on the GPU, especially with respect to the memory layout, array and thread indexing, as well as the kernel code itself. When comparing the values individually, slight differences show up between GPU and CPU implementation, which are discussed in section 8.2.3.
Figure 8.4 shows a set of residuals in x-direction for all planes of the telescope processed by the GPU using recorded hits from the testbeam. One can see the misalignment of the telescope by the shifts of the whole distributions and their asymmetries.
By fitting a gaussian function to the peak region one can obtain the offset of the plane. The width of the gaussian distribution in the peak region is determined by the intrinsic resolution of the pixel sensor ($\sigma_{pixel} = \frac{103\mu m}{\sqrt{12}} = 29.7\mu m$), a contribution from multiple scattering and the tracking resolution. The latter defines the precision of the track reconstruction and it depends on the z-position. Since the track fit takes all four planes into account it is biased towards narrower residual distributions. This can be seen by comparing the residual distributions of the outer (0,3) and inner (1,2) planes. The outermost planes have a higher impact on the reconstructed track because there is only one plane next to it. An inner plane has a constrain from the hits in the planes in front and back. The bias yields narrower residual distributions for the outer planes.

Unbiased Fit Along with the biased fit algorithm using the hits from all four planes to fit the track, an unbiased fit was implemented on the GPU and also tested with recorded testbeam data. It only takes into account three planes for the track fit and the fourth one is left over as a DUT. To get residual distributions for all four planes the fit has to be repeated

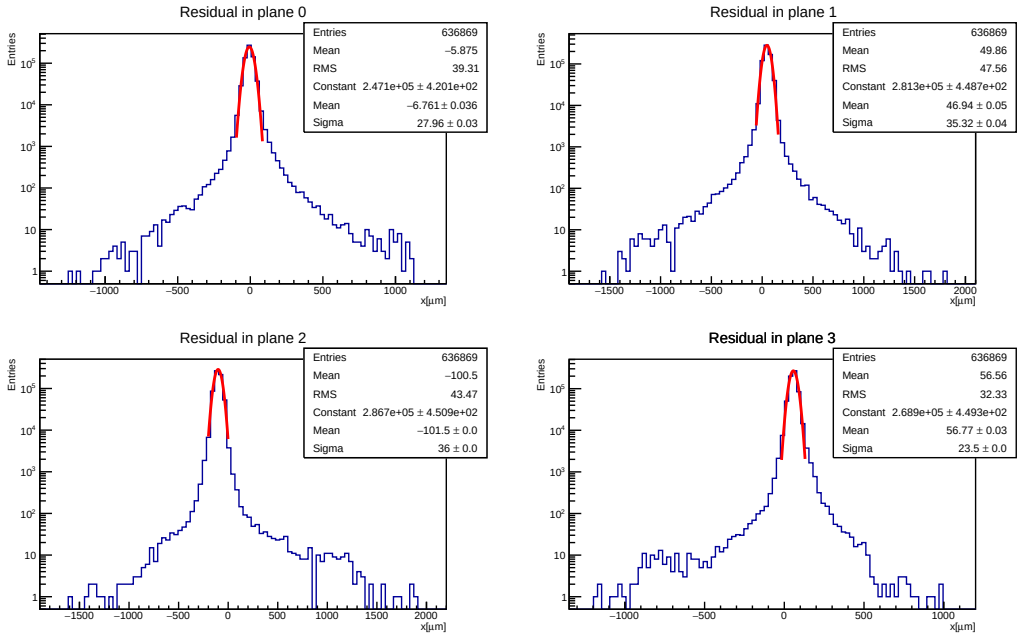


Figure 8.4: Residual distributions of the four planes of the telescope in x-direction processed on the GPU with recorded data from the DESY testbeam.

four times with each plane serving as the DUT once. This is essential to do efficiency measurements and get unbiased residual distributions for the DUT plane. Figure 8.5 shows the residual distributions in x-direction for the four planes using the unbiased fit with the same data file as before.
Despite the similar appearance of the distributions, they show some expected differences compared to the biased fit (cf. figure 8.4). The most obvious is the increased spread of the residual distributions. It comes from the fact that the hits of the DUT are not taken into account for the minimization in the track fit algorithm. The reconstructed track is not pulled towards the hits in the DUT. For the unbiased fit the inner planes have narrower distributions than the outer planes. Taking an outer plane as DUT increases the resolution for the tracking because the length in z-direction, where the track is constrained by hits, is reduced and there is no additional plane before (behind) the first (last) plane.

Chip Efficiency Measurement Given the residual distributions of the unbiased fit for a DUT plane and the total number of reconstructed tracks one can calculate the detection efficiency of the sensor in the DUT plane by comparing the number of hits in a certain search window to the total number of extrapolated hits. The online efficiency measurement requires communication between the different frames of a run to sum up the number of tracks and matched hits in the DUT. On the GPU this requires inter-thread communication which requires some effort to be implemented efficiently, e.g. via parallel reduction algorithms. A naive approach, implemented in CPU software, using the residual data of one plane from the GPU tracking and the number of tracks reconstructed from the other three planes, was tested with the testbeam data.

8.2.2 Telescope Alignment

One desired application of the online track reconstruction, required for an online analysis, is the alignment of the telescope. In first order this means finding the offsets between the individual planes. It can easily be achieved with the help of the residuals, using their offsets

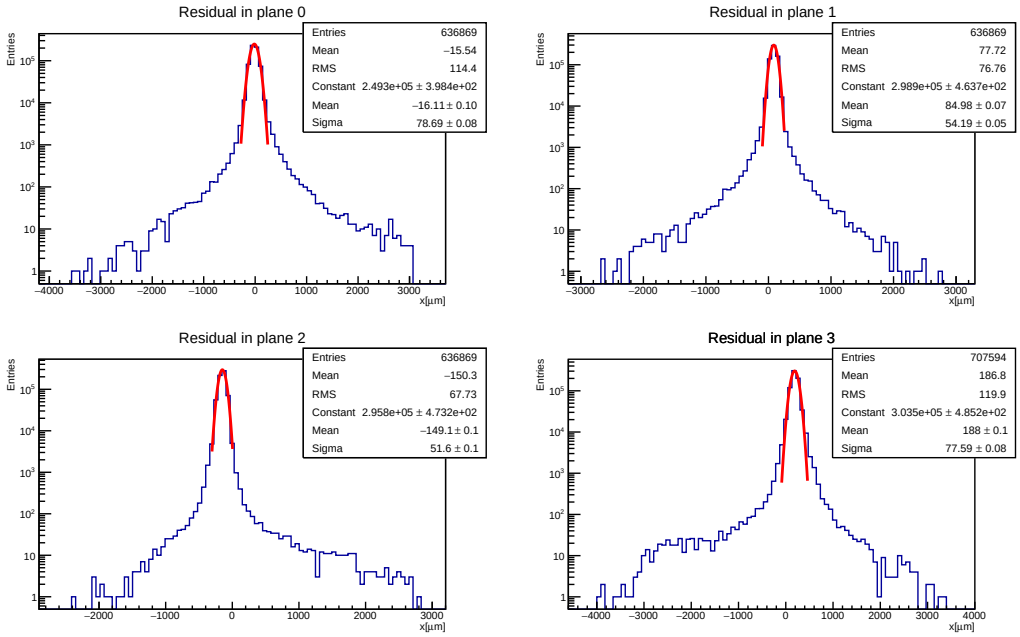


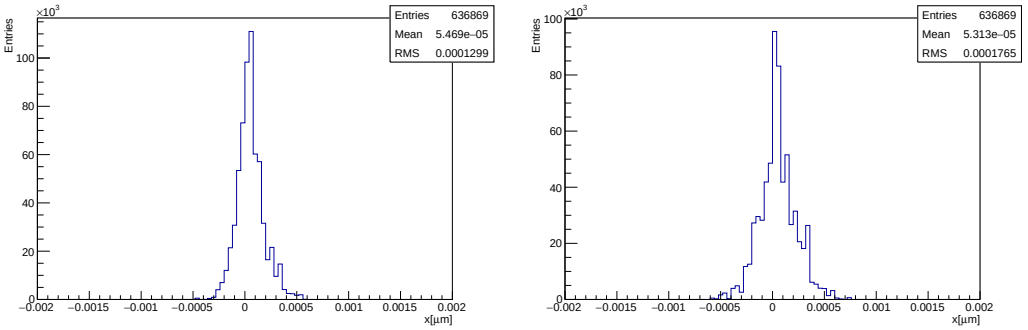
Figure 8.5: Residual distributions for the unbiased track fit in x-direction for the DESY data.

from zero. The idea is to fit the residual distributions with a Gaussian in the peak region, like in figure 8.4, after the tracks are fitted on the GPU, and use the offset parameter to put it into the next iteration of the track reconstruction. This can be repeated up to the desired precision and the final values can then be used in the following online analysis. It was studied using the aforementioned data set giving an increased alignment precision after a reasonable number of iterations. For a break condition that requires the the sum of all four plane residuals to be below 20 µm less than ten iterations were needed. This allows to align the telescope in minutes with a reasonable rate of beam particles.

8.2.3 Floating Point Precision

Taking the difference of the residuals computed by the CPU and GPU for each reconstructed track, differences in the execution between the two processors show up. Figure 8.6a shows the differences between the values of the distribution in figure 8.3 computed by GPU and CPU.

The absolute difference between the resulting values is below 1 nm and has no effect on the resolution. The first obvious reason for that is the different bit lengths of floating point values in both implementations. The CPU version uses 64 bit data words (C++: double type) to encode a value. Since the GPU is optimized to process 32 bit floating point values (C++: float type), the algorithm was implemented to use them. Another reason for the differences can be the different instruction sets used by the processors. The GPU e.g. makes regular use of the fused multiply-add instruction (FMA) which improves performance by calculating $a \cdot b + c$ in only one cycle, which also changes the rounding behavior for the floating point result, because it does not round the intermediate value [23]. The influence of the FMA on the precision of the floating point values of the residuals can be seen in figure 8.6b. Using the FMA in the GPU tracking gives a narrower distribution of the difference to the CPU reference value. The precision is increased by using FMA but it is a small effect. Again, the influence on the relative error by moving from 64 bit floating point values to 32 bit in the computation is on the 10^{-6} level, as suggested by the results in section 7.2.



(a) FMA used.

(b) FMA turned off.

Figure 8.6: Distribution of the differences between the residual values computed by GPU and CPU code with and without FMA instruction used on the GPU ($value_{CPU} - value_{GPU}$).

8.2.4 Performance

Obviously, an open question is the gain in performance by using the GPU for the track reconstruction over the CPU implementation.

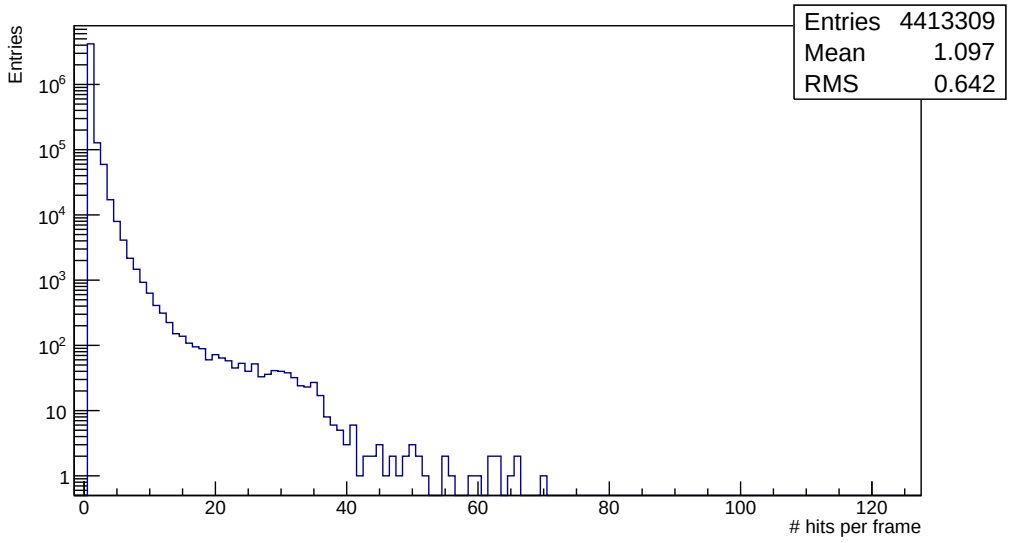
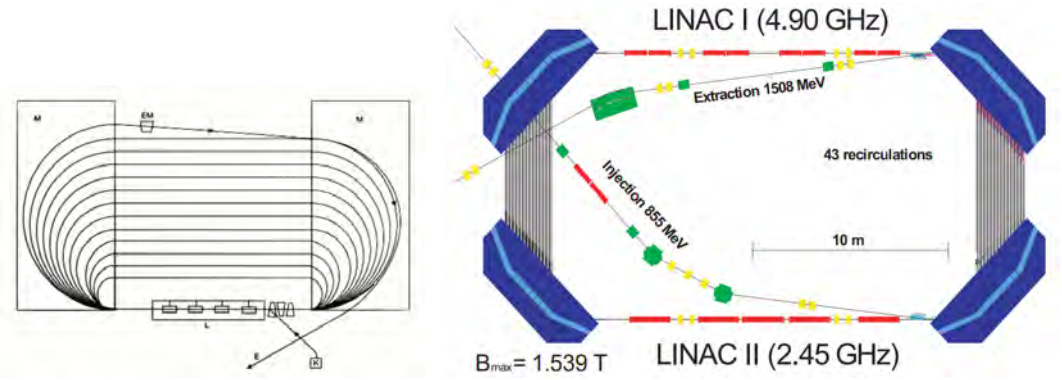


Figure 8.7: Hit multiplicity for the DESY run. There is a wide spread of number of hits in a frame with most of the frames filled very little.

Therefore, the time for the tracking algorithm execution, without file reading, writing and any output (e.g. ROOT histograms) was measured. While the GPU implementation is optimized by the parallelization of the algorithm, the CPU code is single-threaded and does not use any other special accelerating features. With this in mind, the speedup achieved with the GPU is about a factor four for the testbeam data run 313. This is not the ultimate speedup, but it is limited by the low occupancy of the planes at the DESY testbeam, which adds overhead to the GPU implementation. The overhead is mainly generated by the fixed memory size and branch divergence for frames with widely spread number of hits, which is the case for this data run, as shown in figure 8.7. The number of reconstructed tracks and the execution time on an NVIDIA GTX 980 GPU translates into a track fit rate of670 kHz.

9 MAMI Testbeam

From March 29th to April 7th 2016 a testbeam campaign at the Mainz Mictrotron (MAMI) was scheduled to measure the photon detection capability of the MuPix7 sensor prototype and the firmware implementation of the sorting for the telescope GPU tracking. MAMI uses three so called race-track microtrons and a harmonic double sided microtron to accelerate electrons to a maximum energy of 1.5 GeV with a linear microwave accelerator that is passed multiple times. This recirculation is achieved using two dipole magnets to bend the electron tracks by 180° each and multiple beamlines between the two magnets.



(a) Schematic of a racetrack microtron [42].

(b) Layout of the HDSM stage of the MAMI accelerator [43].

Figure 9.1: Electron accelerator types of MAMI.

The fourth stage, accelerating the electrons from 855 MeV to the maximum energy is the so called Harmonic Double Sided Microtron (HDSM) with 4 magnets and two linear accelerators. In the A2 setup, photons are produced by Bremsstrahlung in a metal target. After that, the electrons are deflected by a dipole magnet with plastic scintillators in the focal plane to measure the electron momentum. The beam electrons that do not scatter in the target are conducted into a beam dump. Knowing the electron energy in the beam and after the scattering in the target, the photon energy can be tagged. In figure 9.2 this photon tagging spectrometer is shown.
Figure 9.3 shows a plan of the MAMI accelerator halls.

9.1 Setup

The telescope was placed behind the tagger magnet of the A2 photon beam at the MAMI accelerator where the scattered electrons get tagged. Here the electrons come out of the tagger dipole magnet in a planar array because of the energy spread after scattering. The GPU tracking was tested with a primary beam energy of 1557 MeV. The tagger covers electron energies of about 93 % downto 5 % of the beam energy [46]. To get the straightest possible tracks in the telescope it was placed on the high energy side of the tagger output which resulted in electron energies between 70 % and 50 % of the primary energy. Figure 9.4 shows the final setup of the telescope with the tagger magnet in the background.
Due to the planar array of the electrons escaping the tagger magnet, the alignment procedure was more complicated than for small straight particle beams, where the initial setup gets fairly accurate by an optical alignment. It began with an educated guess for the initial

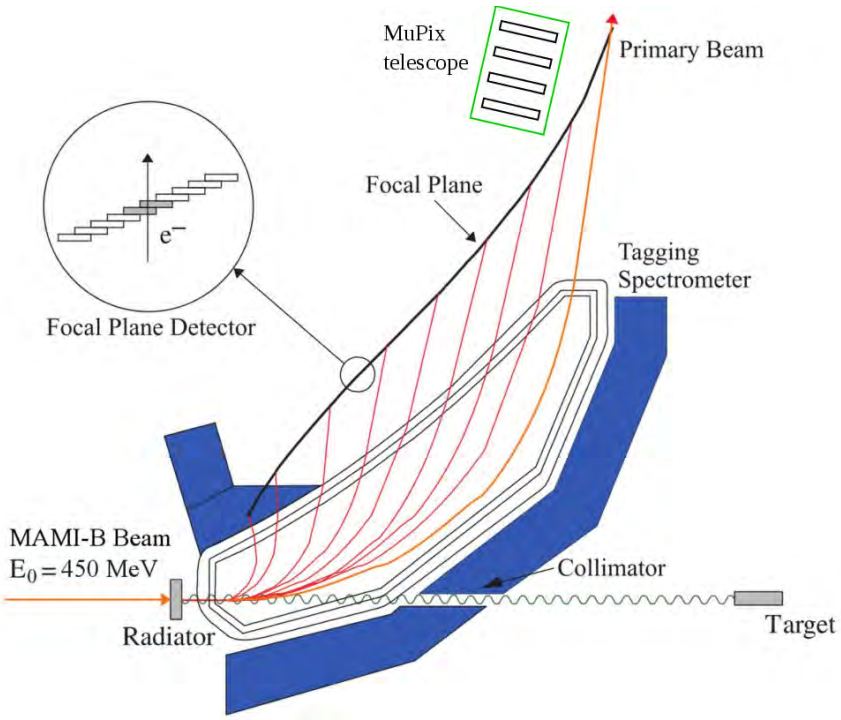


Figure 9.2: Schematic view of the photon tagging spectrometer at MAMI, adapted from [44]. Blue areas denote shielding and the white area in the center is the magnet. The primary electron beam enters from the left (orange arrow), the photons (wavy line) fly straight through the collimator and electron tracks (red) are bent by the magnetic field. Target is where the MuPix setup was situated and the telescope indicated by the green frame.

direction of the telescope, which was good enough to use correlations of hits in the sensor planes for the fine adjustments.
Since by that time the firmware implementation of the sorting algorithm for the GPU tracking was finalized, as described in section 6.3.1, it was the first time to test the GPU online track reconstruction with particle hits on the telescope. Apart from that, the setup was similar to the DESY testbeam in October 2015, as described in 8.1 with the z position of the planes given in table 9.1.

Layer	z-position [mm]
1	0
2	36
3	82
4	126

Table 9.1: Geometry for the telescope setup at MAMI testbeam.

9.2 Measurement

To test the GPU online tracking with a basic setup as described in chapter 6, it was configured with the control functions of the telescope software package followed by the usage of the GPU online tracking framework in the following state:

- Data sorting algorithm implemented in the readout FPGA

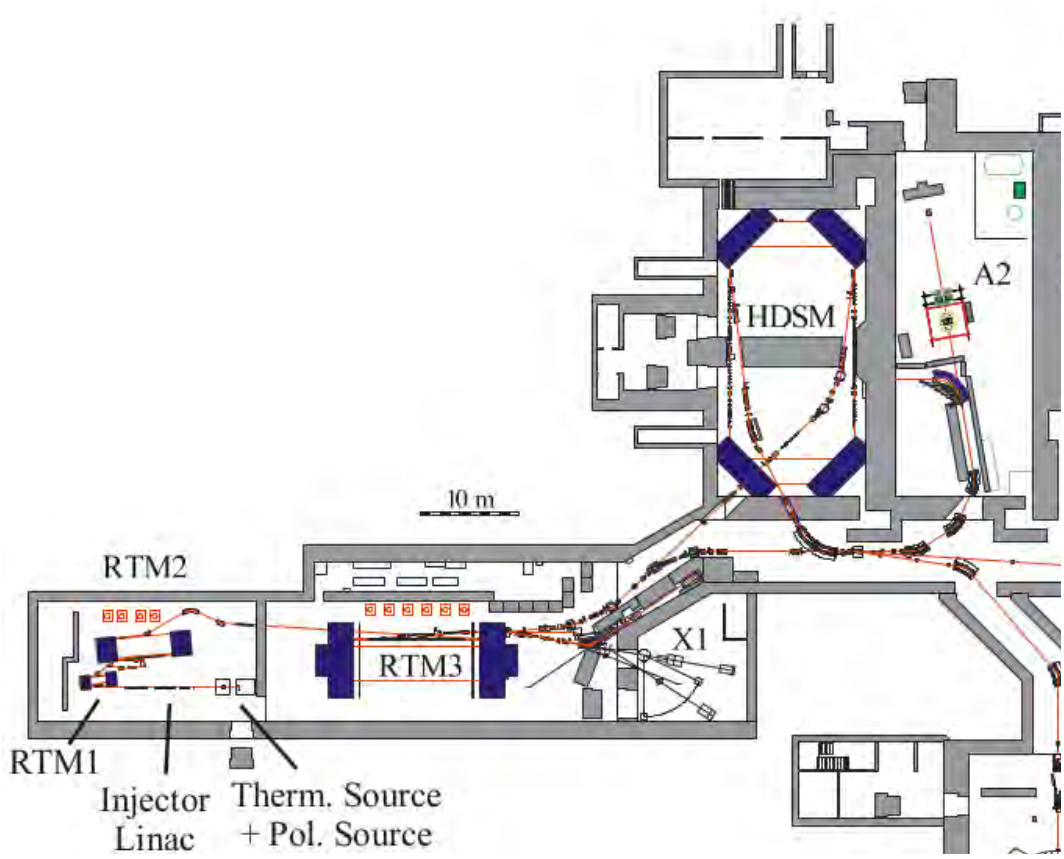


Figure 9.3: Floorplan of the MAMI accelerator with four mictrotrons, adapted from [45].

- DMA transmission to the main memory of the readout PC with driver described in section 6.3.2
- Copy a dataset of 2048 frames to the GPU
- Biased fit algorithm to calculate the residuals
- Copy back residuals of one track per frame to main memory and store them in a binary file

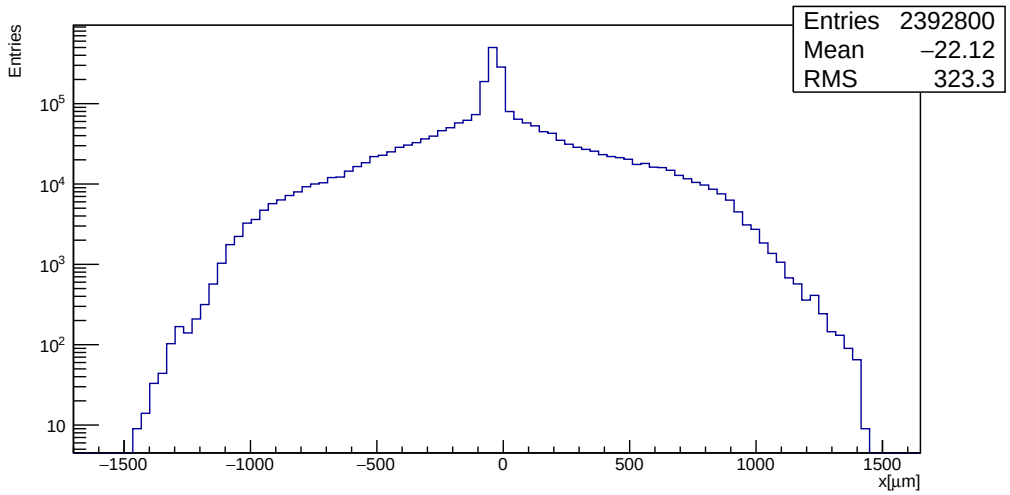
Using the readout FPGA firmware enabled the possibility to measure the residual distributions for the biased fit algorithm with a scan through the particle rate from about 0.5 kHz to 8.0 kHz with the sensors tuned to a noise rate of around 200 Hz. During this measurement the developed FPGA firmware and GPU software were running without any problems impeding data taking, but an unusually high background in the residual distributions was observed, which is explained in section 9.3.

9.3 Analysis

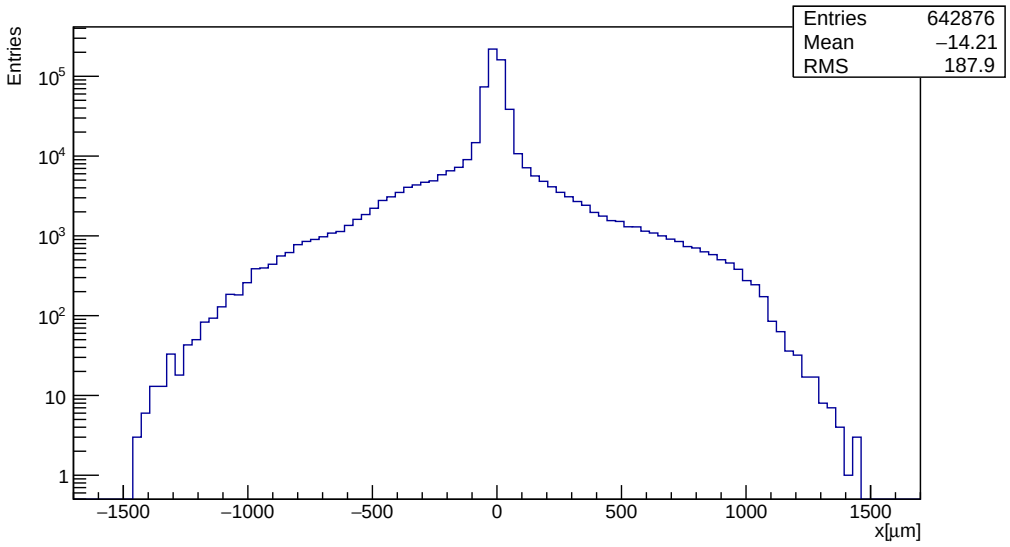
After the testbeam was finished, an analysis of the residual distributions was performed to validate the correct operation of the developed FPGA firmware and GPU software in combination. First of all, the distributions showed a relatively high background contribution which turned out to be a remnant in the GPU fitting program coming from the preceding debugging process. Figure 9.5a shows a residual distribution in x-direction for the first plane with a primary beam energy set to 1557 MeV and the highest measured rate.

Figure 9.4: MuPix telescope set up at MAMI testbeam in Spring 2016.

In principle, the distribution is expected to look similar to the distributions obtained from the DESY data (refer to figure 8.3), but the tracking software was falsely set to choose a random track from a frame instead of the track with minimal χ^2 value, which could explain the enhanced background distribution. Since it was not possible with the current software to store the raw hit data of the telescope during the GPU online track reconstruction, it was impossible to plot distributions with the correct algorithm afterwards. So to validate the results of this testbeam, the algorithm with random tracks was applied to the recorded data from the DESY testbeam to compare the residual distributions.
Figure 9.5 shows the residual distributions of the MAMI testbeam in x-direction for the first plane compared to the DESY data. Showing the same shape, the distributions suggest that the GPU online tracking framework seems to work as expected. The slight differences come from the different setups (e.g. offsets of the planes, beam energy). The different signal to background ratios can be described by the shape of the beam at MAMI. As described in section 9.1 the electron beam does not have a round profile but a band structure. Therefore, beam electrons can hit the frame or the PCBs of the telescope and scatter such that they hit the sensors. This gives additional background hits and enhances the background contribution.



(a) MAMI.



(b) DESY.

Figure 9.5: Residual distribution in x-direction for the first plane as computed by the GPU tracking algorithm at MAMI testbeam (top). The background contribution, in the tails is enhanced over the DESY case (c.f. 8.3) due to a software issue. The same software applied to the DESY data gives the distribution in the bottom.

Part III

Conclusion

10 Summary

The Mu3e experiment aims to search for the lepton flavour violating decay $\mu^+ \to e^+e^-e^+$ with a sensitivity of 10^{-16}. To reach this sensitivity in a reasonable time it has to run with high muon decay rates and needs to have precise spatial, momentum and timing resolution to tag background processes. The high event rate of up to 2 GHz translates in a high data rate for the data acquisition system (DAQ) in the order of $1\,\mathrm{Tbit\,s^{-1}}$, which cannot be stored by current storage devices. To reduce this data rate, an online track reconstruction on Graphics Processing Units selects interesting events to be stored.
As an integration step a similar technique was successfully implemented for the MuPix telescope. It contains the implementation of a pre-sorting algorithm on a Field Programmable Gate Array (FPGA), the test of the data transfer from the FPGA to the GPU and the implementation of a simple track fit algorithm in software executed on the GPU.
Within the boundaries of the algorithm characteristics on the GPU, it delivered the expected results when comparing them with results of the telescope analysis software on the CPU. This was done with data acquired at a DESY testbeam campaign and simulated events. The influence of the reduced floating point precision, used by the GPU, is on the 10^{-6} level for both cases. The real data confirmed the estimated effect obtained from the simulations and the observed differences are negligible. Thus, the GPU tracking algorithm can be used for testbeam measurements.
Comparing the performance of the GPU algorithm with the CPU gives a slight increase by a factor of four but one has to keep in mind that the CPU code was not optimized by any means. The performance of the GPU is currently limited by the data movement processes, not by the computation tasks. When the GPU is fully utilised by filling all frames with the maximal number of hits, the rate of fitting track candidates is 670 MHz for simulated data. The performance of the GPU tracking algorithm highly depends on the structure of the occupancy on the single planes. The best case is when each frame contains the same amount of hits per plane and the parameters are tuned to that size. This applies for pulsed beams as well, because nearly empty frames (less than 3 hits in total) are ignored. The actual situation however is worse, because the number of hits per plane in a frame is spread widely which introduces branch divergence to the GPU execution. This makes the GPU algorithm less efficient than possible according to the raw performance figures of the GPU model.
The sorting algorithm on the FPGA and the data transfer to the GPU using Direct Memory Access were tested at a second testbeam campaign at MAMI. Any major problems have been observed but a software issue resulted in random hits reconstructed as tracks which prevented the final approval of a whole working system. However, the comparison to the DESY data suggested the principal working of the GPU tracking. It would be the next step to test the complete setup at an upcoming testbeam campaign.
An additional confirmation of the whole system working would show the usability for such an online data processing for the Mu3e readout. Therefore, a triplet fit [47] for helical tracks will be implemented on GPUs for the online track reconstruction of the Mu3e experiment.

11 Outlook

The first implementation of the GPU tracking for the MuPix telescope enables some further development in that direction. It is also the basis for a similar implementation in the Mu3e readout system. Especially helpful for the latter is the proven data transfer between the FPGA and the GPU with data from real hits in the pixel sensors with a pre-sorting on the FPGA. For the Mu3e experiment this needs to be scaled up to many more sensors and a more complex tracking scheme which is the scope of an ongoing PhD thesis [39].
The online track reconstruction for the MuPix telescope can be further developed to provide a user friendly analysis in the course of a testbeam measurement. Two main applications would be an alignment tool and an efficiency measurement on the DUT. An alignment tool could be used to measure the offsets of the planes and correct for them mechanically or in software. An online efficiency measurement could enable the tuning of sensor parameters with the help of real particle hits and the sensor's behaviour to them. For the latter, the GPU track fit algorithm has to be further developed and optimized at some points. To achieve higher tracking efficiencies it needs to handle tracks crossing over frame boundaries and allow for more hits to be stored per frame. Then the results of the different frames have to be summed up on the GPU to compute the overall efficiency of the DUT.
A component that could gain some performance by optimizations is the sorting algorithm for the GPU implemented on the FPGA. At the moment it adds overhead due to the fixed size of the memory allocated per frame, which is especially problematic for sparsely filled frames. Therefore, the size of the memory has to be chosen such that it is large enough to hold the data of all frames but as small as possible to minimize the overhead. This requires a study of the hit occupancy per frame for different particle beam lines, or more specifically to different beam rates. Another approach would be to investigate the use of a dynamic allocation scheme for different frames.

Part IV

Appendix

A Lists

A.1 List of Figures

A.2 List of Tables

B Bibliography

[1] J. Beringer et al. Review of Particle Physics (RPP). *Phys.Rev.*, D86:010001, 2012. doi: 10.1103/PhysRevD.86.010001.

[2] Wikimedia. Standard Model of Elementary Particles, 2014. URL https://commons.wikimedia.org/w/index.php?curid=36335876. [Online; accessed 20-April-2016].

[3] F. Englert and R. Brout. Broken Symmetry and the Mass of Gauge Vector Mesons. *Physical Review Letters*, 13:321–323, August 1964. doi: 10.1103/PhysRevLett.13.321.

[4] P. W. Higgs. Broken Symmetries and the Masses of Gauge Bosons. *Physical Review Letters*, 13:508–509, October 1964. doi: 10.1103/PhysRevLett.13.508.

[5] G. S. Guralnik, C. R. Hagen, and T. W. Kibble. Global Conservation Laws and Massless Particles. *Physical Review Letters*, 13:585–587, November 1964. doi: 10.1103/PhysRevLett.13.585.

[6] G. Aad et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. 2012.

[7] S. Chatrchyan et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys.Lett.B*, 2012.

[8] R. H. Bernstein and P. S. Cooper. Charged Lepton Flavor Violation: An Experimenter's Guide. *Phys. Rept.*, 532:27–64, 2013. doi: 10.1016/j.physrep.2013.07.002.

[9] W. J. Marciano, T. Mori, and J. M. Roney. Charged Lepton Flavor Violation Experiments. *Ann.Rev.Nucl.Part.Sci.*, 58:315–341, 2008. doi: 10.1146/annurev.nucl.58.110707.171126.

[10] A. Baldini et al. Search for the Lepton Flavour Violating Decay $\mu \to e^+\gamma$ with the Full Dataset of the MEG Experiment. 2016.

[11] U. Bellgardt et al. Search for the Decay $\mu^+ \to e^+e^+e^-$. *Nucl.Phys.*, B299:1, 1988. doi: 10.1016/0550-3213(88)90462-2.

[12] W. H. Bertl et al. A Search for Muon to Electron Conversion in muonic Gold. *Eur.Phys.J.*, C47:337–346, 2006. doi: 10.1140/epjc/s2006-02582-x.

[13] K. A. Olive et al. Review of Particle Physics. *Chin. Phys.*, C38:090001, 2014. doi: 10.1088/1674-1137/38/9/090001.

[14] R. M. Djilkibaev and R. V. Konoplich. Rare Muon Decay $\mu^+ \to e^+e^-e^+\nu_e\bar{\nu_\mu}$. *Phys.Rev.*, D79:073004, 2009. doi: 10.1103/PhysRevD.79.073004.

[15] A. Blondel et al. Research Proposal for an Experiment to Search for the Decay $\mu \to eee$. *ArXiv e-prints*, January 2013.

[16] A. Herkert. Gaseous helium cooling of a thin silicon pixel detector for the mu3e experiment. Master thesis, Heidelberg University, 2015.

[17] I. Perić. A novel monolithic pixelated particle detector implemented in high-voltage CMOS technology. *Nucl.Instrum.Meth.*, A582:876, 2007. doi: 10.1016/j.nima.2007.07.115.

[18] H. Augustin et al. The MuPix System-on-Chip for the Mu3e Experiment. 2016.

[19] N. Berger. The Mu3e Readout Scheme. Private communication.

[20] J. Parkhurst, J. Darringer, and B. Grundmann. From single core to multi-core: Preparing for a new exponential. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '06, pages 67–72, New York, NY, USA, 2006. ACM. ISBN 1-59593-389-1. doi: 10.1145/1233501.1233516. URL http://doi.acm.org/10.1145/1233501.1233516.

[21] F. A. Förster. HV-MAPS Readout and Direct Memory Access for the Mu3e Experiment. Bachelor thesis, Heidelberg University, 2014.

[22] Wikimedia. Noise reduction using differential signaling, 2012. URL https://commons.wikimedia.org/wiki/File:DiffSignaling.png. [Online; accessed 03-May-2016].

[23] NVIDIA. CUDA Toolkit Documentation. 2015. URL http://docs.nvidia.com/cuda/#axzz48cpz3aFy. [Online; accessed 14-May-2016].

[24] Intel. Intel core i7-5820k processor. URL http://ark.intel.com/de/products/82932/Intel-Core-i7-5820K-Processor-15M-Cache-up-to-3_60-GHz. [Online; accessed 19-May-2016].

[25] T. Sandhu. Review: Nvidia GeForce GTX 980 (28nm Maxwell). URL http://hexus.net/tech/reviews/graphics/74849-nvidia-geforce-gtx-980-28nm-maxwell/. [Online; accessed 19-May-2016].

[26] N. Wilt. *The CUDA Handbook: A Comprehensive Guide to GPU Programming*. Addison-Wesley, 2013. ISBN 9780321809469. URL https://books.google.de/books?id=p3wEywAACAAJ.

[27] N. Gupta. What is "constant memory" in CUDA. URL http://cuda-programming.blogspot.de/2013/01/what-is-constant-memory-in-cuda.html. [Online; accessed 20-May-2016].

[28] Xinxin Mei and Xiaowen Chu. Dissecting GPU Memory Hierarchy through Microbenchmarking. *CoRR*, abs/1509.02308, 2015. URL http://arxiv.org/abs/1509.02308.

[29] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *Computer*, 41(7): 33–38, July 2008. ISSN 0018-9162. doi: 10.1109/MC.2008.209. URL http://dx.doi.org/10.1109/MC.2008.209.

[30] IEEE Computer Society. IEEE Standard for Binary Floating-Point Arithmetic, 1985. IEEE Std 754-1985.

[31] H. Wong, M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos. Demystifying GPU Microarchitecture through Microbenchmarking. In *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pages 235–246. IEEE, 2010.

[32] L. Huth. Development of a Tracking Telescope for Low Momentum Particles and High Rates consisting of HV-MAPS. Master thesis, Heidelberg University, 2014.

[33] J. Philipp. Efficienzanalyse von HV-MAPS anhand des MuPix-Teleskops. Bachelor thesis, Heidelberg University, 2015.

[34] P.A. Franaszek and A.X. Widmer. Byte oriented DC balanced (0,4) 8B/10B partitioned block transmission code, December 4 1984. URL http://www.google.com/patents/US4486739. US Patent 4,486,739.

[35] A. X. Widmer and P. A. Franaszek. A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code. *IBM Journal of Research and Development*, 27:440, 1983.

[36] A.-K. Perrevoort. *Sensitivity Studies to New Physics in the Mu3e Experiment and Development of firmware for the Mu3e pixel detector front-end*. PhD thesis. in preparation.

[37] A.-K. Perrevoort. Data Acquisition at the Front-End of the Mu3e Pixel Detector. DPG spring meeting talk, February 2016.

[38] L. Huth. Status and Results from the October Beam Test Campaigns. Talk, Mu3e collaboration meeting, November 2015.

[39] D. vom Bruch. *Online Track Reconstruction for the Mu3e Experiment on Graphics Processing Units*. PhD thesis. in preparation.

[40] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth.*, A389:81–86, 1997. doi: 10.1016/S0168-9002(97)00048-X.

[41] DESY. Schematic Layout of a Test Beam at DESY. URL https://particle-physics.desy.de/e252106/e252106/e252211. [Online; accessed 10-May-2016].

[42] "zeitgenosse". Racetrack-Microtron (shematic). URL http://www.relativ-kritisch.net/forum/viewtopic.php?t=1261. [Online; accessed 17-May-2016].

[43] A. Jankowiak. The Mainz Microtron MAMI: Past and future. *Eur. Phys. J.*, A28S1: 149–160, 2006. doi: 10.1140/epja/i2006-09-016-3.

[44] M. Unverzagt. eta and eta-prime Physics at MAMI. *Nucl. Phys. Proc. Suppl.*, 198: 174–181, 2010. doi: 10.1016/j.nuclphysbps.2009.12.034.

[45] MAMI. Layout MAMI-C. URL http://www.phmi.uni-mainz.de/Dateien/01_phys_kernphys_mamiC_einweihung_grundriss.pdf. [Online; accessed 17-May-2016].

[46] J. C. McGeorge et al. Upgrade of the Glasgow photon tagging spectrometer for Mainz MAMI-C. *Eur. Phys. J.*, A37:129–137, 2008. doi: 10.1140/epja/i2007-10606-0.

[47] A. Schöning. A Three-Dimensional Helix Fit with Multiple Scattering using Hit Triplets. 2014. Mu3e internal note.

Acknowledgements

I would like to thank everyone who supported me and helped to carry out this thesis.
First of all I would like to thank Prof. Dr. André Schöning and Prof. Dr. Niklaus Berger for giving me the opportunity to work on this very interesting topic at the Institute of Physics in Heidelberg and the Institute for Nuclear Physics in Mainz. Furthermore, I would like to thank Prof. Dr. Schöning and Prof. Dr. Ulrich Uwer for the survey of my thesis.
In general, I would like to thank the Mu3e group in Heidelberg and the group of Prof. Dr. Berger in Mainz for providing a really great working atmosphere and help whenever it was requested. It was a pleasure to join the testbeams or the DPG spring meeting with all of you.
Especially, I would like to thank:
Niklaus Berger for helpful discussions about GPU and FPGA programming, the Mu3e experiment and particle physics in general.
Dorothea vom Bruch for the help during the whole time of working out this thesis, for the ideas and discussions about GPU computation and for the productive teamwork on firmware, driver and GPU programming.
Lennart Huth for nice discussions and explanations concerning the telescope and anything else, for patient help with setting up the telescope at testbeams and for a motivating and optimistic attitude all the time.
Heiko Augustin for patient explanations of the MuPix sensor and being a great room mate during the testbeam and DPG week.
Sebastian Dittmeier for interesting discussions about FPGA programming and the MuPix readout and being a great room mate at PSI and DESY.
Marco Zimmermann for help with the ROOT framework, for creating the interest in learning Python, for discussions on the P2 detector and for providing me a place to sleep during the MAMI testbeam.
Uli Hartenstein for our nice conversations while going by train between Heidelberg and Mainz.
Additional thanks to Nik, Dorothea, Lennart, Heiko and Sebastian for proofreading of this thesis.
In the end I would like to thank my family and friends, especially my parents for supporting me during my whole study and covering my back all the time.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, 01.06.2016 .....