Department of
Physics and Astronomy

University of Heidelberg

Diploma thesis
in Physics

submitted by
Moritz Kretz
born in
Mannheim, Germany

May 2012

Studies Concerning the ATLAS IBL Calibration Architecture

This diploma thesis has been carried out by **Moritz Kretz**

at the

Institute of Computer Engineering

under the supervision of

Prof. Dr. Reinhard Männer and **Dr. Andreas Kugel**

Studies Concerning the ATLAS IBL Calibration Architecture

With the commissioning of the Insertable B-Layer (IBL) in 2013 at the ATLAS experiment 12 million additional pixels will be added to the current Pixel Detector. While the idea of employing pairs of VME based Read-Out Driver (ROD) and Back of Crate (BOC) cards in the read-out chain remains unchanged, modifications regarding the IBL calibration procedure were introduced to overcome current hardware limitations. The analysis of calibration histograms will no longer be performed on the RODs, but on an external computing farm that is connected to the RODs via Ethernet.

This thesis contributes to the new IBL calibration procedure and presents a concept for a scalable software and hardware architecture. An embedded system targeted to the ROD FPGAs is realized for sending data from the RODs to the fit farm servers and benchmarks are carried out with a Linux based networking stack, as well as a standalone software stack. Furthermore, the histogram fitting algorithm currently being employed on the Pixel Detector RODs is ported to a GPU architecture and optimized for parallel execution, increasing the performance of a previous implementation by a factor of 10. As an alternative, CPU based fitting methods are investigated for their practicability.

Untersuchungen zur ATLAS IBL Kalibrationsarchitektur

Mit der Installation des Insertable B-Layer (IBL) im Jahr 2013 am ATLAS-Experiment wird der bereits bestehende Pixeldetektor um 12 Millionen Pixel erweitert. Obwohl der Ansatz, Paare bestehend aus VME-basierten Read-Out Driver (ROD) und Back of Crate (BOC) Karten in der Auslesekette einzusetzen, weiterhin Bestand hat, wurde die IBL- Kalibrationsprozedur modifiziert, um momentan vorhandene Limitierungen seitens der Hardware zu beseitigen. Die Analyse von Kalibrationshistogrammen wird nicht mehr wie bisher auf den RODs durchgeführt werden sondern auf einem externen Cluster, der mit den RODs per Ethernet verbunden ist.

Diese Arbeit trägt zur Entwicklung des IBL-Kalibrationsverfahrens bei und stellt ein Konzept einer skalierbaren Software- und Hardwarearchitektur vor. Um Daten von den RODs an den Cluster zu senden, wird ein eingebettetes System für den Einsatz auf den ROD FPGAs umgesetzt und Benchmarks sowohl mit dem Linux-Netzwerk-Stack als auch einem optimierten, eigenständigen Software-Stack durchgeführt. Darüber hinaus wird der momentan auf den Pixeldetektor RODs verwendete Fitalgorithmus zum Einsatz auf einer GPU portiert und für parallele Ausführung optimiert. Im Vergleich zu einer vorherigen Implementierung kann die Geschwindigkeit der Fits um den Faktor 10 gesteigert werden. Als Alternative werden Software-basierte Fitmethoden zur Ausführung auf CPUs hinsichtlich ihrer Verwendbarkeit untersucht.

Contents

List of Figures

List of Tables

1 Introduction

"It's only work if somebody makes you do it." [1]

Over the past century particle physics experiments have come a long way from bubble chambers to the advent of colliders like LEP, eventually leading to the large colliders and experiments at the Tevatron and the LHC. The development, assembly, and operation of these large experiments have become an interdisciplinary effort involving scientists from different fields like physics, engineering, and computer science.

By observing the reaction products resulting from particle collisions one can draw conclusions regarding the fundamental interaction mechanisms and the structure of matter. Currently the standard model (SM) of particle physics best describes the observable phenomena related to the fundamental interactions (without gravity). The SM predicts and relies on the existence of the still to be discovered *Higgs boson*.

Besides the main goal of collider experiments to find the Higgs boson as the last missing particle of the standard model, one is also interested in probing a possible supersymmetric extension of the SM, resulting in heavy particles that could be produced by collisions at the LHC. Further research focusses on the creation of miniature black holes, the quark gluon plasma, more precise measurements of the properties of already known particles, and of course on unexpected (*new physics*) discoveries.

In order to achieve these goals not only a boost of the center of mass energy is necessary compared to previous colliding beam experiments, but one also needs to further increase the collision rates to efficiently probe very rare events.

ATLAS The ATLAS experiment is besides CMS one of the two general purpose detectors at the LHC at CERN and has been delivering physics data since 2009. The search for a Higgs boson in accordance to the Standard Model has been a benchmark process for the design of the detector and had a great influence on the detector architecture as it is in use today.

With the 2011 proton run ATLAS recorded an integrated luminosity of just over $5\,\mathrm{fb}^{-1}$, resulting in a total data taking efficiency of 93.5% for the year. Analysis of the 2011

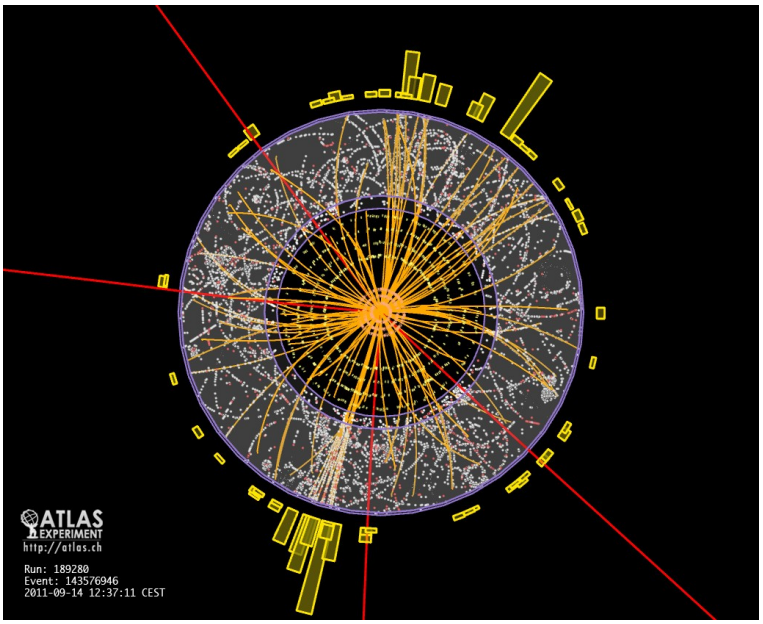


Figure 1.1: Higgs candidate event for the $H \to ZZ \to \mu\mu\mu\mu$ channel from ATLAS during the 2011 run. The innermost circles represent the three layers of the current pixel detector, the four red lines the muons.

dataset shows slight hints of the existence of a standard model Higgs boson with a mass of approximately 126 GeV [2], still, even combining the results of ATLAS and CMS did not give enough statistical certainty to claim a scientific discovery. Nonetheless it is expected that with the data to be taken during the 2012 run, ATLAS will be able to successfully prove (or rule out) the existence of a SM Higgs particle.

The pixel detector is part of the inner detector of ATLAS and plays a major role in delivering the required tracking performance. It is the innermost sub-detector, acquiring true 2-dimensional position measurements of traversing particles. Radiation hardness as well as a low material-budget were major design considerations in order to achieve good results concerning the reconstruction of particle track vertices, that are important for a good physics performance.

The current LHC run plan schedules a long shutdown of over a year beginning at the end of 2012, that gives needed time for repairs and installation of new equipment. A major project at ATLAS will be the commissioning of the Insertable B-Layer (IBL, [3]) during the shutdown period. The IBL adds an additional layer of approximately 12 million pixels to the current pixel detector and is with a radius of only 33 mm even closer to the beampipe and hence interaction point than the current innermost pixel layer (B-Layer). Upgrading the pixel detector not only guarantees a highly efficient tracking despite the continuously increasing irradiation and therefore failure of the original pixel sensors, but even further improves the precision of determining secondary vertices resulting from the

decay of heavy particles.

As the current pixel detector, the IBL heavily depends on the efficient read-out performed through a set of front-end electronics and the processing and formatting of the resulting data. Therefore the new front-end chip FE-I4 was developed that guarantees an efficient read-out despite a higher track density and also integrates the functionality of the separate Module Control Chip used with the FE-I3. For the read-out of the front-end electronics a new set of back of crate cards (BOC) and read-out driver cards (ROD) was developed, taking advantage of recent improvements in hardware technology by realizing the functionality of custom electronics on FPGA devices. These new cards are capable of handling an increased amount of data while remaining backwards-compatible with the previous versions.

The detector periodically needs to be calibrated due to radiation effects. The analysis of calibration data is a computationally demanding task that is performed on the ROD of the current pixel detector, but changes in the design of the IBL ROD lead to a new calibration architecture relying on external processing.

This work focusses on the changes of the IBL calibration procedure. The envisioned external computing farm needs to be integrated into the computing framework for a unified calibration process of all four pixel layers. In order to transfer the histogram data from the RODs to the computing farm, an appropriate network protocol must be chosen with regards to reliability and speed, and logical interfaces between the RODs and the fit farm computers need to be defined. Finally the fitting functionality that is currently performed on the ROD must be ported to the computing farm and possible alternatives should be evaluated regarding their performance.

Outline The following chapter gives a more detailed overview of the ATLAS experiment laying the focus on the pixel detector and the planned IBL upgrade as well as on the general computing framework. The next three chapters present the results relevant to the implementation of the calibration framework for the IBL: Chapter 3 shows the planned integration of the software components into the current pixel detector framework and identifies central points when scaling the system. The fourth chapter focusses on the histogram data transfer via Ethernet, while chapter 5 benchmarks different methods of performantly fitting the calibration histograms on standard CPUs as well as GPUs. Chapter 6 summarizes the obtained results and gives an outlook on open questions and tasks.

2 The Experiment

"I liked things better when
I didn't understand them." [1]

The ATLAS experiment is located at the LHC at CERN and has been in development for over 20 years. After the commissioning of the detector was finished, proton beams circulated in the LHC tunnel at the end of 2008 for the first time. Due to a faulty electrical connection a quench occurred in some LHC magnets nine days later, damaging magnets and resulting in a shutdown of over one year to perform the necessary repairs [4]. In November 2009 the LHC resumed its operation and ATLAS has been (with the exception of two planned maintenance periods) taking data since.

In this chapter relevant information about the LHC and the design of the ATLAS experiment in general is presented first, followed by a more detailed view of the pixel detector and the Insertable B-Layer (IBL) upgrade. We finish with a short discussion of the ATLAS trigger and data acquisition (TDAQ) concepts and software.

2.1 The LHC

The LHC was built in a 27 km tunnel near Geneva that had previously been used by the Large Electron-Positron collider (LEP) until its decommissioning in 2000. Unlike LEP, it accelerates *protons* in two separate vacuum beampipes using over 1500 superconducting magnets and currently is the most powerful particle accelerator in use. With a design luminosity of $\mathcal{L} = 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ and a center of mass energy of $\sqrt{s} = 14\,\mathrm{TeV}$ the LHC enables physicists to study a number of interesting open questions related to particle physics:

- Determine the mass of the Higgs boson as predicted by the standard model of particle physics: The Higgs boson is predicted by the SM, but its mass is a free parameter and needs to be experimentally determined. Previous experiments at the LEP or the Tevatron were only able to exclude a certain mass range for the particle, but did not prove its existence. The LHC allows for a more effective search above the previously excluded mass regions.

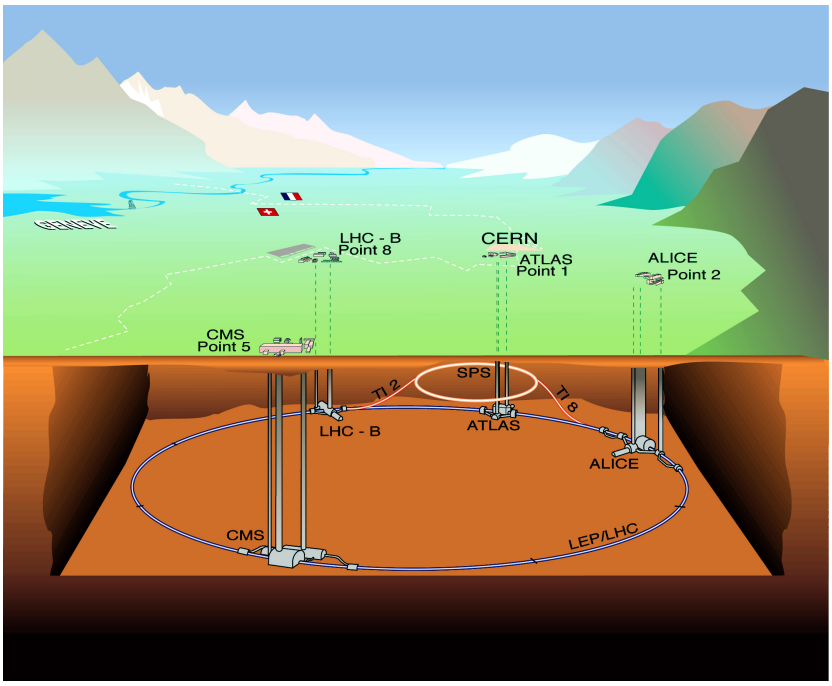


Figure 2.1: The LHC accelerator complex with the four detectors ALICE, ATLAS, CMS, and LHCb. Image: CERN, used with permission.

- In case the existence of a SM Higgs can be excluded, one is particularly interested in finding new particles as predicted by supersymmetric theories, like for example the minimal supersymmetric standard model (MSSM) that predicts (besides other superparticles) a family of Higgs bosons.

- What is the nature of the Quark-Gluon Plasma (QGP): In order to achieve the high energy densities needed to create QGP, the LHC is also able to accelerate lead ions up to an energy of 574 TeV/nucleus. For the analysis of Pb-Pb collisions a specialized detector (ALICE) has been built, but other LHC experiments will also take part.

- Processes that are violating Charge-Parity (CP) symmetry.

- One also wants to make more precise measurements of the properties of elementary particles, whose existence has already been verified, such as the W and Z gauge bosons and the top quark.

In order to achieve these goals the LHC provides high luminosity (and therefore high interaction rates that are necessary for statistical analysis of processes with a small cross-section) and also an unprecedented beam energy that allows the study of processes that were not accessible to previous collider experiments.

Accelerator Chain The LHC is fed with protons by a system of accelerators. Protons are first injected in a linear accelerator (Linac2) before being further accelerated by three synchrotons, namely the Proton Synchrotron Booster, the Proton Synchrotron (PS) and the Super Proton Synchrotron (SPS). After the last step the protons have an energy of 450 GeV and are injected into the LHC ring. Plans exist to optimize this acclerator chain in the future — after the LHC is running at its nominal specification — to further increase the achievable luminosity.

Figure 2.1 gives an overview of the LHC complex and the location of the four major experiments. ATLAS and CMS are the two general purpose detectors, while ALICE focusses on the analysis of Pb-Pb collisions. LHCb is a detector mainly designed for the study of b-physics covering a 2π solid angle.

As already mentioned in Chapter 1 the LHC provided just over $\int \mathcal{L}\,dt = 5\,\text{fb}^{-1}$ integrated luminosity in 2011 (with a peak luminosity of $\mathcal{L} = 3.65 \times 10^{33}\,\text{cm}^{-2}\,\text{s}^{-1}$) and it is expected to deliver an integrated luminosity of $\int \mathcal{L}\,dt = 15\,\text{fb}^{-1}$ at $\sqrt{s} = 8\,\text{TeV}$ in 2012. Despite the planned bunch spacing of 25 ns (corresponding to an interaction rate of 40 MHz) the LHC currently operates with a bunch spacing of 50 ns. After the 2012 run the LHC will enter a long shutdown phase of at least 18 months in order to prepare the machine for its design luminosity and a center of mass energy of $\sqrt{s} = 14\,\text{TeV}$. This timeframe will also be used by the experiments for maintenance and detector upgrades (see Section 2.4 for the plans for the ATLAS pixel detector upgrade).

2.2 ATLAS

ATLAS is besides CMS another multi-purpose detector located at the LHC and was designed to observe a very broad spectrum of expected physical processes, but also has the potential to discover so-called *new physics*. It is the largest detector at CERN with a length of 45 m, diameter of 25 m and is weighing 7000 t. Opposed to CMS it uses *two* superconducting magnets creating the fields necessary for measuring the momentum of particles — a solenoid magnet for the inner detector and a toroid for the muon chambers. In Figure 2.3 a picture of the whole detector is shown, illustrating the different sub-detector layers and the system of magnets as well as the immense size of the experiment.

2.2.1 Requirements

To meet the physics goals mentioned in the previous section, a set of requirements was developed for the ATLAS experiment [5, pp. 2-3]:

- Fast and radiation hard sensors and electronics due to high beam energy and luminosity.

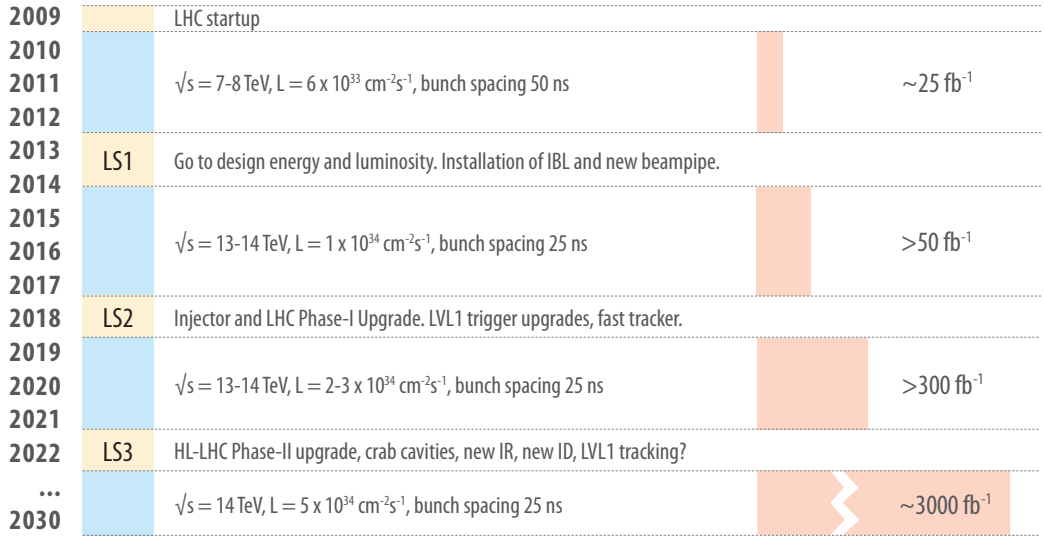


Figure 2.2: The planned upgrade phases for the LHC and possible changes for ATLAS according to [6]. The run parameters as well as the expected integrated luminosity are shown. More information can be found in [7, 8].

- An almost hermetic design with very good coverage of the azimuthal angle.
- Good electronic and hadronic calorimetry.
- High momentum resolution and reconstruction efficiency in the tracker, with the detector close to the primary interaction point for efficient b-tagging and vertex reconstruction.
- Good muon identification and momentum measurement.
- Sophisticated trigger system due to the large number of events.

With an increasing luminosity paired with the current modus operandi regarding the bunch spacing, the trigger and reconstruction systems already have to cope with a high pileup. The situation will become even more challenging in the future after several LHC upgrades. Figure 2.2 gives an overview of the planned shutdown phases as well as the associated upgrade projects.

2.2.2 Components

ATLAS follows an onion-like rotationally-symmetric layout, consisting of several sub-detectors. As CMS it is designed to be hermetic and covers almost the full 4π solid

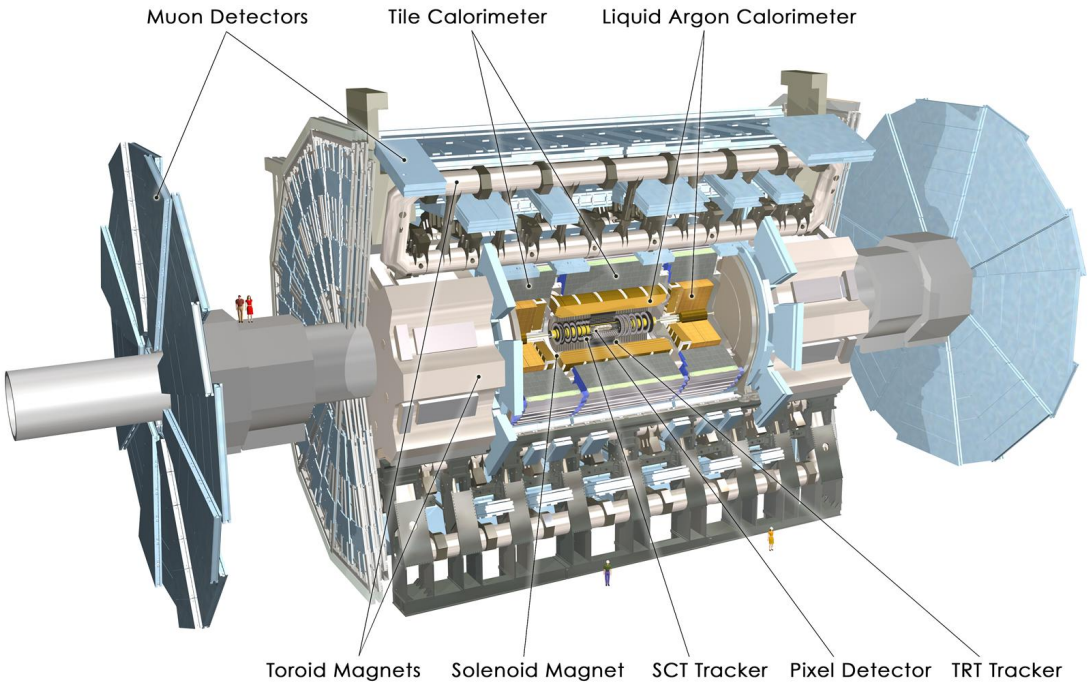


Figure 2.3: Overview of the main components of the ATLAS detector. Image: CERN, used with permission.

angle around the interaction point in order not to miss any particle emerging from an interaction[1].

Inner Detector The inner detector (ID) provides the precise tracking necessary to reconstruct primary and secondary track vertices and therefore allows for a good momentum resolution and b-tagging. It consists of the pixel detector, the silicon tracker (SCT), and the transition radiation tracker (TRT) and is surrounded by the solenoid magnet that creates a nearly uniform 2 T magnetic field bending the path of charged particles passing through the ID for momentum measurements.

The pixel detector is composed of 80 million pixels and is situated closest to the beampipe. Section 2.3 will cover this sub-detector in greater detail. The SCT is made up of four double-sided cylindrical layers of silicon strips, while in the end-cap regions it consists of nine disks aligned perpendicularly to the beam axis. Arranging the microstrips on one side of an SCT module at a slight angle towards the ones on the other side makes it possible to also determine the position of the traversing particle in the z-direction (parallel to the beampipe). With 6.2 million read-out channels it is coarser than the pixel detector, but provides tracking over a larger volume.

[1] Neutrinos are the only known particles that cannot be detected by ATLAS. Therfore 4π coverage is also important to indirectly identify neutrinos by calculating the "missing energy" of an event that can be associated with a neutrino.

The outermost part of the ID is the TRT, ranging from a radius of about 55 cm to 110 cm around the beampipe, but also covering the end-cap region. It consists of 298,000 straw tubes with a diameter of 4 mm filled with a gas mixture[2]. A particle generates on average 36 hits while traversing the TRT. The TRT is able to distinguish particles that are directly ionizing the gas in the tubes (hadrons) from particles that generate transition radiation photons when passing through radiator foils between the straws (electrons or positrons) in addition to its tracking capability.

Figure 2.4 shows the three sub-detectors of the ATLAS tracker annotated with their distances to the beampipe. A more detailed blueprint of the ID can be found in the ATLAS TDR [5, p. 54].

Calorimeters ATLAS employs two sampling calorimeters to measure the energy of a particle while also providing some spatial information: The *electromagnetic* calorimeter and the *hadronic* calorimeter. The elegromagnetic calorimeter is situated outside the ID and uses liquid argon for signal collection and lead as an absorber. The hadronic calorimeter also uses liquid argon in the end-caps, but scintillating plastic tiles in the barrel region. Tungsten, copper, and steel are used as absorber materials. Both calorimeters also provide data for generation of level-1 trigger signals.

Muon Detector The muon spectrometer is the largest sub-detector of ATLAS and consists of four different types of detectors to not only accurately identify and measure muon tracks, but also to provide trigger signals: Monitored Drift Tubes (MDTs) and Cathode Strip Chambers (CSCs) are used for the tracking while the fast Resistive Plate Chambers (RPCs) and Thin Gap Chambers (TGCs) provide trigger signals. To achieve the necessary positioning accuracy of 30 µm, a sophisticated alignment system is used. A toroid magnet in the barrel region as well as two end-cap toroids are used to create the magnetic field necessary for momentum measurement. In total about 1.1 million channels are read out.

Magnets There are two distinct superconducting magnet systems in use in ATLAS to generate the fields necessary for momentum measurements. The central solenoid creates a uniform field of 2 T in the inner detector to enable a good momentum resolution. Its layout was optimized with regard to the radiation length to achieve a good calorimeter performance. The non-uniform magnetic field for the muon chambers is created by a system of toroid magnets in the barrel (0.15 - 2.5 T) and end-cap (0.25 - 3.5 T) regions. Sensors are placed throughout the detector to monitor the magnetic field. Both systems need to be cooled by liquid helium and it takes one week for the solenoid and five weeks for the toroid to reach the desired temperature of 4.5 K.

[2] typically 70% Xe, 27% CO_2, 3% O_2

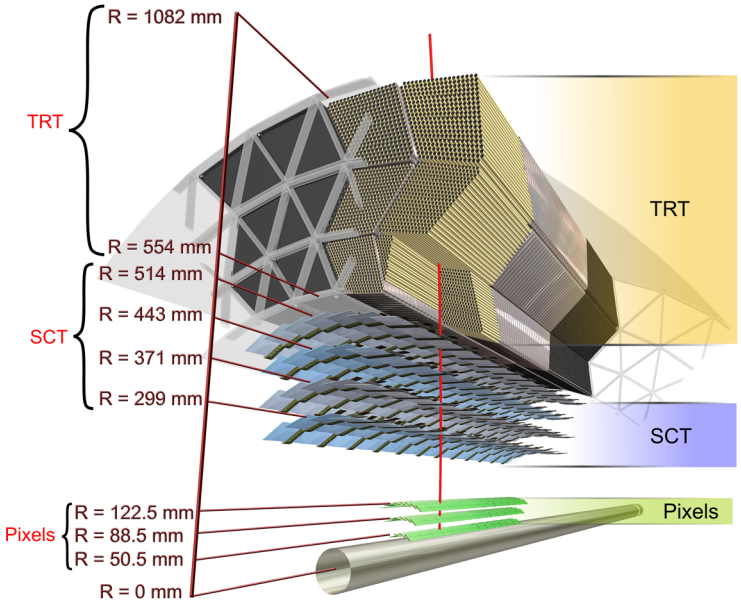


Figure 2.4: The inner detector of ATLAS with the pixel detector, the silicon tracker (SCT), and the transition radiation tracker (TRT). Image: CERN, used with permission.

2.3 Pixel Detector

As the detector closest to the interaction point the pixel detector has to fulfill a number of extreme requirements. Due to the expected flux of highly energetic particles it must be able to operate in harsh radiation conditions and still perform as expected even after years of irradiation. It was built with a low material budget in mind to minimize effects of multiple scattering and secondary interactions. The performance of the pixel detector is crucial for a good tracking and b-tagging efficiency.

2.3.1 Pixel Fundamentals

Pixel detectors are the obvious choice to fulfill the requirements of a tracking detector close to interaction point. In order to avoid a high occupancy a fine granularity of the detector elements is necessary. Yet, in contrast to the pixel sensors of for example digital cameras that sample incoming photons over a longer time period, the sensors used in particle physics also need to be fast (in the order of 10 ns) to cope with the high collision rates, radiation hard to guarantee a functioning system even after years in a high particle flux environment, and robust against single event upsets (SEUs) caused by radiation.

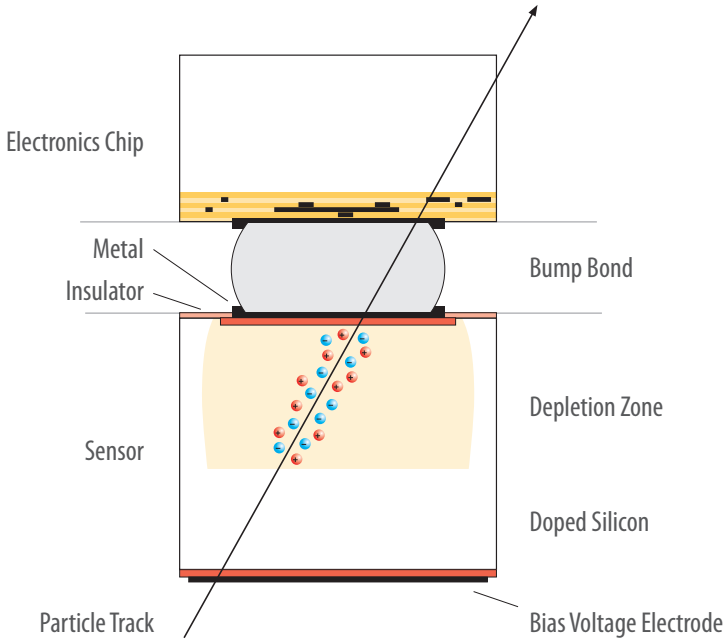


Figure 2.5: Schematic of a particle passing through a single hybrid pixel cell, generating free charges that are moving in the depletion region and produce a hit signal. Graphic adapted from [10].

This is achieved by using a hybrid design, where the sensors are produced separately from the front-end electronics, and later connected via bump bonds (for the ATLAS pixel detector two different bump bonding processes were used [9, pp. 41]). To get an impression how a hybrid pixel sensor is divided, Figure 2.5 shows a particle passing through a pixel sensor.

The physical principle involved is that ionizing radiation passing through the sensor generates free charges through scattering processes with the electrons of the sensor material. Doped silicon[3] is the most prominent example for a semiconducting sensor material (3.6 eV needed to produce an electron-hole pair) and the sensor itself can be depicted as a pn-diode that is operated reversely biased. In current particle physics experiments an n^+-type read-out electrode in n-type substrate is being used.

The mean energy loss of an ionizing particle in the sensor can be described by the Bethe-Bloch formula [11, pp. 286]. The notable result from this is that there exists a minimum of deposited energy for particles with certain energies, and therefore such a particle is referred to as a minimum ionizing particle (MIP). One can assume that all relativistic particles are MIPs and we can further convert the deposited energy of an MIP to an equivalent number of electron-hole pairs in the sensor — for the ATLAS pixel detector with a sensor width of 250 µm this is 20,000 e^- given a normal incidence [12].

[3] In the case of the ATLAS pixel detector it is also *oxygenated* to lessen the effects of irradiation.

Irradiation Effects

During its lifetime the pixel sensors will be exposed to a high integrated particle flux[4]. The incoming high-energy particles not only interact with the electrons of the silicon crystal, but are also able to irreversibly displace Si-atoms from the lattice creating vacancies and interstitials in the bulk material.

This irradiation has several effects on the pixel sensor according to [10, pp. 68]:

- Increase of leakage current due to generation-recombination centers leading to a higher thermal footprint requiring effective cooling to avoid thermal runaway[5]
- Charge trapping reducing signal height
- Type inversion (space charge sign inversion)

The creation of charged lattice defects leads to a change of the effective doping N_{eff} of the sensor material, up to the point where the original n-type material behaves like p-type material (type inversion) and the depletion voltage V_{dep} increases with further irradiation. As the depletion region grows in the opposite direction (from the pixel electrodes) after type inversion, it is possible to operate the detector partially depleted, although one has to cope with lower signal strengths. A damage model to predict the development of the depletion voltage taking into account the effects of (reverse) annealing was used in the design process of the ATLAS pixel detector, showing the usefulness of oxygenated silicon to lessen the effects of irradiation [13].

Front-End Electronics

The front-end is responsible for amplifying and shaping the signal received through the bump connection from the sensor. It has to apply a discriminator threshold used to determine if the signal height is above a configurable value and buffer the resulting digitized values until a trigger is received. Two possible modes of operation are a binary read-out, where only information on whether the signal of a pixel crossed the threshold (i. e. the pixel was hit) is buffered, and a read-out including the amplitude information of the signal (digitized value of the Time over Threshold (ToT)). Once a trigger signal arrives, the front-end needs to transfer the event data corresponding to the indicated bunch-crossing to the off-detector electronics via optical links for further processing. Several options exist for the implementation of the read-out: The data of many single front-end chips can be aggregated and formatted by an MCC (current ATLAS pixel detector), which also supplies the chips with timing and trigger information.

[4] For the IBL a NIEL dose tolerance of $5 \times 10^{15}\, n_{eq}/\text{cm}^2$ and a TID tolerance of 250 Mrad is required.
[5] Thermal runaway refers to a positive feedback system in which higher temperature leads to an increase in power dissipation, again producing more heat.

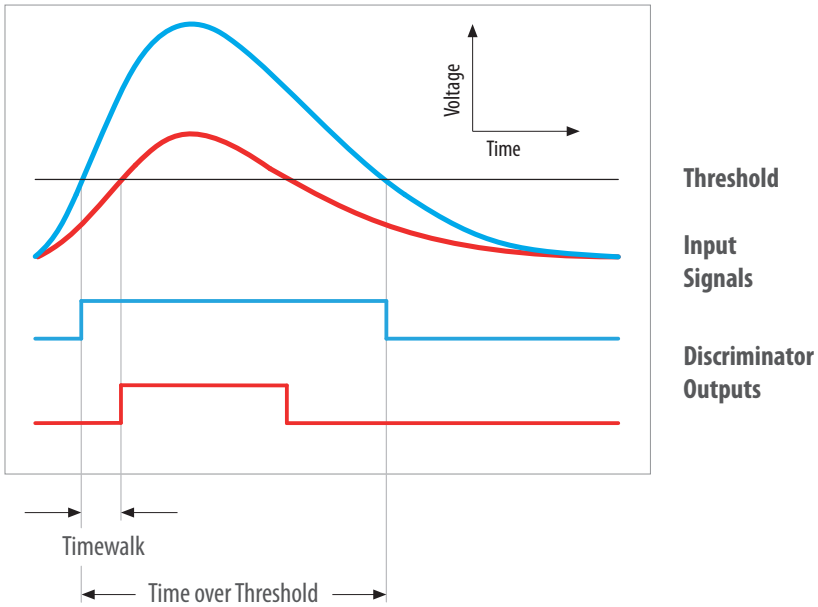


Figure 2.6: Two signals of different amplitude after the amplifier and discriminator stage, illustrating threshold value, the time over threshold, and timewalk.

To store configuration data of each pixel such as threshold values one needs digital registers that can cope with SEUs. For calibration and tuning purposes (see Section 2.3.4) a mechanism to inject a known charge into the pixel needs to be provided. Figure 2.6 illustrates the signals that are produced by the discriminator stage of the electronics.

Compared to silicon strip detectors the front-end design is more complex, as the electronics need to be connected to every single pixel sensor via bump bonds and therefore limit the minimum size of a pixel.

Noise

Electronic noise is an intrinsic unwanted effect in all circuits that needs to be accounted for and is a basic property of a pixel detector. In case of small signals it notably limits the achievable sensitivity of the detector, negatively impacting the tracking performance. There are several sources of noise in the analog part of the electronics (sensor, amplifier, shaper): Thermal noise, shot noise, and 1/f noise.

For a more detailed discussion of noise sources and quantitative analysis we refer to the literature ([14], [10, pp. 170], [15, pp. 33]). The noise varies from pixel to pixel caused by slight differences during the production processes as well as irradiation during operation.

Number of pixels	80.4 million
Operational pixels [17]	96.2%
Pixel size	$50 \times 400\,\mu\text{m}^2$
Sensor thickness	$250\,\mu\text{m}$
Intrinsic accuracy (barrel region)	$10\,\mu\text{m}$ $(r-\phi)$, $115\,\mu\text{m}$ (z)
Time resolution	$< 25\,\text{ns}$

Table 2.1: Parameters of the pixel detector

As the hit discriminator threshold is not uniformly distributed across the pixels, the resulting threshold dispersion has to be considered as an additional noise term, resulting in the total noise

$$\sigma = \sqrt{\sigma_{\text{el}}^2 + \sigma_{\text{thr}}^2}.$$

The design goals for ATLAS were $\sigma_{\text{el}} < 300\,\text{e}^-$ after irradiation and $\sigma_{\text{thr}} < 200\,\text{e}^-$ [16, p. 71].

2.3.2 ATLAS Pixel Detector

The ATLAS pixel detector consists of three barrel layers (b-layer, layer-1, layer-2, the b-layer being the innermost) and three disks in both forward directions, totaling 80.4 million pixels and thus reaching a higher granularity compared to the other two tracking detectors. The cylindrical layers span a distance $50.5\,\text{mm} < R < 122.5\,\text{mm}$ around the beam pipe and have a length of 81 cm, while the disks are situated perpendicular to the beam direction covering $88.8\,\text{mm} < R < 149.6\,\text{mm}$. The system has an active area of $1.7\,\text{m}^2$ and is operated at a temperature of $-7°\text{C}$ to achieve robustness against radiation damage.

Table 2.1 summarizes some design and operational parameters of the pixel detector. The non-operational part of the detector is mainly caused by disabled modules or front-end chips, a much smaller part is due to defective bump connections.

A Module Controller Chip (MCC) is used to connect 16 FE-I3 chips with the off-detector electronics. The electric signals are transformed to be transferred via optical links by patch panels located close to the MCCs detector.

2.3.3 Read-Out

The off-detector electronics are centered around the VME bus system, consisting of 132 pairs of Back of Crate (BOC) and Read-Out Driver (ROD) cards. They are mounted in VME crates that also house a single board computer (SBC) controlling the components in the crate and an interface card to the global ATLAS timing, trigger, and control. The

BOC card provides the optical interfaces for communication with the detector front-end. Arriving data is demultiplexed by the BOC card and forwarded to the corresponding ROD card that performs data processing. The resulting event fragments are then transferred back to the BOC and sent to the Read-Out System (ROS) that is part of the ATLAS DAQ chain via HOLA[6]. The corresponding hardware infrastructure can be seen in Chapter 3, while Section 2.5 will discuss the ATLAS TDAQ framework in more detail.

The master DSP (MDSP) on the ROD controls the calibration scans presented in the next section. The resulting histograms are directly analyzed on the RODs by slave DSPs (SDSPs) and the results are forwarded to the PixelDAQ framework via the SBC for storage in a calibration scan database.

2.3.4 Calibration

The pixel detector offers several scans in order to calibrate the optical modules used for communication with the off-detector electronics as well as the pixel modules themselves. This becomes necessary to guarantee a uniform behavior of the detector despite changing operating conditions. In the following we will focus on two commonly used scan types: the *threshold scan* and the *time over threshold scan*. Further details on available scans can be found at [18].

Threshold Scan

A threshold scan determines the threshold μ and noise σ_{el} of a pixel. This is achieved by injecting a defined charge Q multiple times a_0 into the pixel and counting the resulting hits. One then iterates over several charge steps (typically 100) by changing the voltage V_{cal} and creates a histogram[7]. In the ideal case a pixel would not display any noise effects and the hit probability would resemble the Heaviside step function

$$p_{\text{hit}}(Q) = H(Q - \mu) \tag{2.1}$$

with the threshold value μ and the injected charge Q. However, we need to take the intrinsic noise of the pixel into account and under the reasonable ([14, pp. 125]) assumption of a Gaussian noise distribution one can model the probability of a hit for a given charge injection by a convolution of the Heaviside step function with a normalized Gaussian function [15, p. 99]:

[6] High-Speed Optical Link for ATLAS, an implementation of a CERN data-link specification

[7] This procedure is not done for all pixels of a module in parallel. An additional loop is introduced to scan only a part of the module defined in a mask ("mask-stepping"). For the FE-I3 one mask step covers 1/32 of the module's pixels.

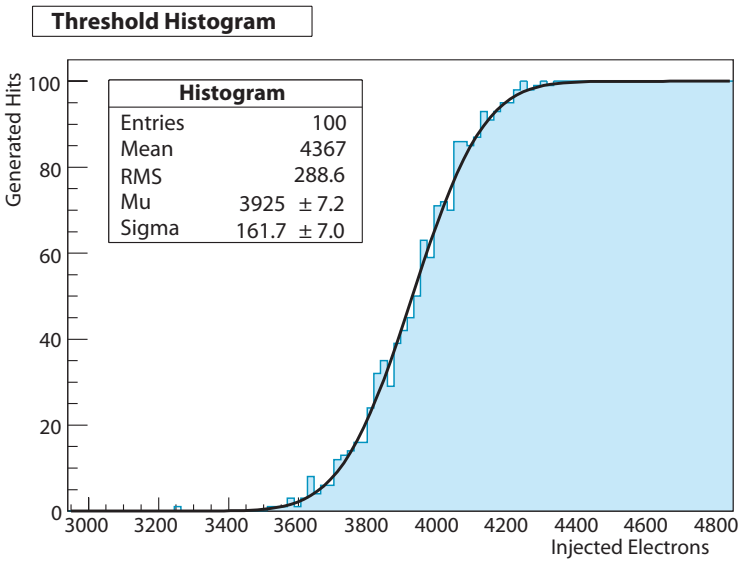


Figure 2.7: Histogram from a threshold scan and the corresponding S-curve fit with the fitted parameters for pixel threshold μ and noise σ.

$$p_{\text{hit}}(Q) = ()\, H\,(Q-\mu) * \frac{1}{\sqrt{2\pi}}\frac{1}{\sigma_{\text{el}}}\exp\left(-\frac{Q^2}{2\,\sigma_{\text{el}}^2}\right) \tag{2.2}$$

$$= \frac{1}{2}\,\text{Erfc}\left(\frac{\mu-Q}{\sqrt{2}\,\sigma_{\text{el}}}\right) \tag{2.3}$$

$$= \frac{1}{2}\left(1+\text{Erf}\left(\frac{Q-\mu}{\sqrt{2}\,\sigma_{\text{el}}}\right)\right) \tag{2.4}$$

Equation (2.4) corresponds to the cumulative distribution function (CDF) of a Gaussian distribution and is sometimes referred to as an S-curve type function because of its shape. The threshold is defined as the injected charge, where the hit probability corresponds to 50%. For a more detailed calculation see Appendix A. The error function $\text{Erf}(x)$ is defined as

$$\text{Erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,\text{d}t \tag{2.5}$$

and related to the complementary error function $\text{Erfc}(x)$ in the following way:

$$\text{Erfc}(x) = 1 - \text{Erf}(x)\,. \tag{2.6}$$

Figure 2.7 shows the resulting histograms of a scan with and without noise for 100 charge injections at 100 different values of V_{cal} and the resulting S-curve fit. The fitting of histogram data to the objective function is carried out by the SDSPs on the ROD.

As a last step the threshold dispersion for a complete module can be calculated using a histogram of the pixels' threshold values to decide if threshold *tuning* is necessary.

Threshold Tuning In order to keep the threshold dispersion σ_{thr} and thus the total noise level low one periodically needs to perform a threshold scan on the detector and possibly adjust the thresholds of each pixel using a process called threshold tuning. The front-end chip offers two methods to tune the threshold setting: changing the GDAC value that globally (per front-end chip) trims all pixel thresholds and/or adjusting each individual pixel's TDAC value.

Tuning the threshold is realized by different algorithms detailed in [18]. The GDAC value only needs tuning when the number of pixels requiring extreme TDAC values in order to reach the desired threshold is high. GDAC as well as TDAC have an approximately linear effect on the threshold only in the middle of their ranges, making the tuning procedure an iterative process. To speed up the tuning of the GDAC, only few values are scanned and then a linear interpolation is used to determine the best GDAC value.

There exist two approaches for tuning the TDAC: during initial tuning the TDAC value is set to the middle of its range and a charge corresponding to the desired threshold is injected multiple times — the next candidate TDAC value depends on whether this procedure generated more or less than 50% of the maximum number of hits. Initial and fine tuning differ in the number of steps and the step size and starting point.

In 2008 and 2009 the detector operated with a threshold $\mu = 4000\,\text{e}^-$ and a spread $\sigma_{\text{thr}} = 40\,\text{e}^-$ for the whole detector. The threshold value was lowered to $\mu = 3500\,\text{e}^-$ in 2011.

Time over Threshold Scan

The time over threshold scan is used to determine a calibration curve between the time over threshold and the charge being generated in the pixel. Charges are injected multiple times in the pixel and one iterates over V_{cal} in 6 steps. For each injection the ToT is recorded and subsequently for each iteration step of V_{cal} mean and sigma of the time over threshold are calculated. The resulting histograms are then fitted to polynomials of first and second degree.

Similar to the threshold tuning one can also perform a ToT tuning to have a uniform response across the detector. This is realized by tuning the global and local feedback currents controlled by the values of the IF and FDAC registers.

2.4 IBL Upgrade

Originally it was foreseen to replace the b-layer of the pixel detector, but extraction of the layer is not possible. Therefore it was decided to integrate an additional fourth pixel layer inside the current pixel detector. Installation of the IBL was intended for a long LHC shutdown in 2015/2016, but changes in the run plan lead to an earlier schedule starting at the end of 2012. Reasons for the installation of the IBL are manifold.

- Tracking robustness: compensate for the unavoidable failures in the current pixel layers and keep a good tracking performance.
- B-tagging performance: the higher luminosity expected in the future leads to more event pileup and therefore a higher occupancy in the b-layer and possibly a reduced b-tagging performance.
- Increased precision: the closeness to the interaction point improves vertexing and b-tagging performance.
- New beampipe: A new beampipe will be installed together with the IBL to fit the additional layer.
- Experience for ID replacement: The whole project is a good opportunity to further gain experience in new sensor technologies for the replacement of the whole ID in the far future.

Three possible types of sensor technologies are under investigation: planar, 3D, and diamond[8]. Currently it is planned to produce enough modules with planar sensors to build 100% of the IBL and an additional 25% with 3D sensors [19]. The 3D sensors will be installed at both ends of the layer (at large pseudorapidities $|\eta|$). In total there will be 12 million pixels in the IBL spread over 224 modules with 2 FE-I4 chips each.

2.4.1 Detector Front-End

A new version of the front-end chip FE-I3 had to be designed for the IBL called FE-I4, as the old chip did not meet the requirements regarding radiation hardness and hit rate capability. One sensor and one (for 3D sensors) or two (for planar sensors) front-end chips will form a pixel *module*.

The FE-I4 is produced in a 130 nm process and has a new internal architecture (local buffering as opposed to the column drain approach of the FE-I3) to support higher data-rates. It will feature an increased active area (74% $\rightarrow$ 90%) as well as smaller pixel

[8] The diamond sensors can be operated at room temperature at low threshold (target 800 e^-) and will be used for the diamond beam monitor (DBM), an addition to the beam conditions monitor (BCM) that is used for monitoring the beam condition and detecting instabilities in order to protect the detector, as well as for luminosity measurements. The DBM will be part of the IBL installation procedure.

Parameter	**FE-I3**	**FE-I4**
Pixel size	$50 \times 400\,\mu m^2$	$50 \times 250\,\mu m^2$
Process	$0.25\,\mu m$	$130\,nm$
Pixel array	$18 \times 160 = 2880$	$80 \times 336 = 26880$
Dimensions	$0.76 \times 1.08\,cm^2$	$2.02 \times 1.88\,cm^2$

Table 2.2: Comparison of FE-I3 and FE-I4 parameters

sizes and will futher integrate the functionality of the MCC. The ToT resolution of the FE-I4 of 4 bit is smaller compared to the 8 bit of the FE-I3. Smaller values are possible for the mask-stepping loop during scans (i. e. more pixels can be scanned in parallel). The changes will mostly be transparent to the calibration procedures we are discussing in this work and thus we refer to [20, 21, 22] for additional information on the FE-I4. Table 2.2 compares important parameters of the FE-I3 and FE-I4.

2.4.2 Readout and Calibration

The general design of the read-out chain remains the same with small changes [23, 24]. The ROD and BOC cards were redesigned to take advantage of technological advancements and to straighten out bottlenecks of the old cards [25, 26]. The new BOC can handle four times the data rate of the old one. The hardware of both cards was designed to be compatible with the old versions and only a different firmware version is needed to achieve full backward-compatibility.

One of the most drastic changes that also motivates this thesis is related to the calibration procedures of the detector. The new ROD will not be equipped with slave DSPs known from the old device, but will rather offload their functionality to an external computing farm. This becomes feasible as the ROD is no longer restricted to the slow communication via the VME bus: two Gigabit Ethernet links per ROD will be used to ship the gathered histogram data from calibration scans off the ROD (Chapter 4 focusses on this aspect). The demanding calculations can then be performed on commodity hardware (CPUs/GPUs) which will make development more flexible and convenient. Chapter 5 presents possible implementations to take advantage of this.

2.5 Trigger & Data Acquisition System

The ATLAS Trigger & Data Acquisition (TDAQ) system needs to efficiently handle and analyse the data provided by the 1600 read-out links coming from the detector. Figure 2.8 shows the major components and the dataflow of the system. It can be

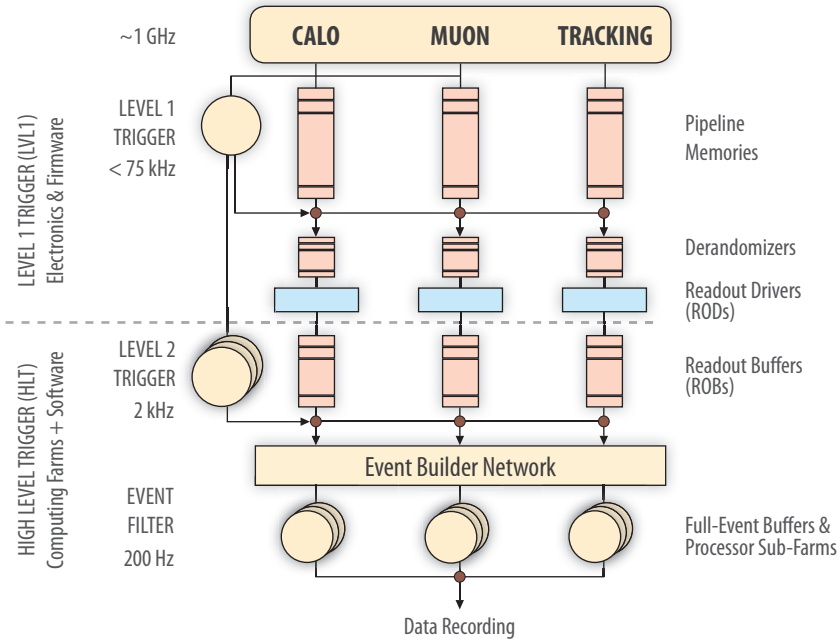


Figure 2.8: The ATLAS TDAQ system with its main components, the different trigger levels and corresponding rates [28, p. 16].

divided into four components: the dataflow system, the high level trigger (HLT), the online system, and the detector control system (DCS) [27, p. 6].

Trigger The trigger system consists of three levels to select only interesting events and reduce the event rate of 1 GHz at design specification down to 200-300 Hz for permanent storage and offline analysis. It is made up of the level-1 trigger and the high level trigger (HLT) that is further divided into the level-2 trigger and the event filter (EF).

The level-1 trigger is completely implemented in hardware to generate a trigger decision within 2.5 µs. In the current implementation only information from the muon detector and the calorimeters are being used for the level-1 trigger resulting in an output rate of 75 kHz. Event data accepted by the L1 are sent from the RODs to the Read-Out Systems (ROSs) that consist of multiple Read-Out Buffers (ROBs — in total a farm of about 150 computers equipped with ROBIN cards [29]), where they are further buffered and made available to the HLT.

The HLT is realized in software for easier maintainability and scaling purposes — also computing nodes of the HLT can be dynamically used by either level-2 or EF processes. The level-2 trigger employs faster trigger algorithms with lower precision than the EF, reducing the event rate to 2 kHz. It operates only on a subset of the available event data, so called Regions of Interest (RoIs), that were previously defined by the level-1 trigger. RoIs for an event are combined and then assigned to a compute node for analysis by

the level-2 algorithms that can then request event data from the ROBs. Finally, the EF operates on fully built events and selected events are then stored permanently.

Online Software The online software system is the central component connecting the elements of HLT, DAQ and detector control. It provides services for information exchange, handles the start and shutdown of processes and enables the user to monitor the operation of the detector. The Process Manager (PM) is used to create and monitor processes in the highly distributed computing landscape ($\mathcal{O}$(1,000) nodes with $\mathcal{O}$(10,000) processes) [30].

The common object request broker architecture (CORBA)[9] is used for distributed communication as the basis of the online software implementation [31]. A software wrapper (IPC) has been introduced for easier development of DAQ applications. *Partitions* can be used to run self-contained instances of the DAQ for a sub-detector that are able to independently take data.

Software Development Software components for ATLAS are continuously developed[10] to fix bugs and add new features. tdaq-04-00-01 is the production release of the software for the year 2012, while tdaq-common-01-18-04 and dqm-common-00-18-03 are the releases containing support packages commonly being used for ATLAS software development.

The development for the pixel detector subsystem is done in the PixelDAQ source tree, providing libraries and applications for the detector operation such as *CalibrationConsole*. The related development for the IBL software takes place in the IBLDAQ branch, which will eventually be merged with the main PixelDAQ tree.

Additional information on the design and specifications of the TDAQ system can be found in the TDR [27]. For details on the offline computing and analysis infrastructure and concepts that are not discussed here, we refer to [32].

[9] `http://www.corba.org`
[10] `http://atlas-tdaq-sw.web.cern.ch/atlas-tdaq-sw/`

3 IBL Calibration Architecture

"What do they think I am,
an engineer?" [1]

It was decided that for the IBL a new approach for the calibration procedure would be taken. The computationally demanding step of analyzing the histograms, that are gathered during calibration scans, is not any longer performed on the RODs by DSPs, but will be offloaded to an external computing farm. Consequently, the SDSPs have been removed from the new ROD design and additional Gigabit Ethernet network links were added to the card for high-speed data transfer to the computing farm as the VME bus was only able to deliver 7 MByte/s [33].

The introduction of these new components in the system calls for a change in the calibration system architecture. Furthermore, interfaces on different logical levels between IBL RODs and the computing farm need to be defined and configuration items must be elaborated.

In this chapter the envisioned architecture for the calibration system of the IBL is presented. First an overview of the command and data flow is given, before the integration into the ATLAS TDAQ system is explained. Later scalability and implementation details are discussed.

3.1 Components

The major off-detector hardware components introduced with the IBL relevant for calibration are:

- **Two Spartan-6 XC6SLX150 FPGAs** per ROD: During calibration scans these FPGAs will be used to accumulate hits (and additionally ToT and ToT^2 for ToT scans).
- **Gigabit Ethernet PHY** (DP83865DVH) connected to each Spartan-6: Histogram data is transferred from the ROD cards to the corresponding server(s) of the computing farm via these interfaces.

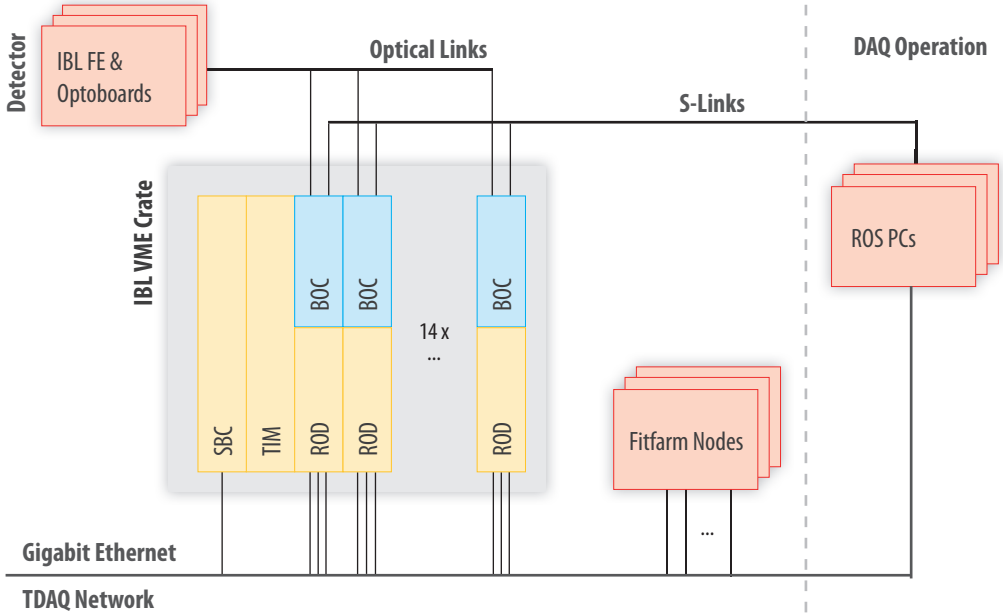


Figure 3.1: Hardware architecture of the IBL off-detector electronics and read-out components.

- **Gigabit Ethernet Network**: The network connecting the above network interfaces with the computing farm. We will not discuss this any further and assume that it is able to handle the theoretical maximum bandwidth of 28 GBit/s (14 ROD cards with the two relevant[1] Gigabit interfaces each) and provides the needed services (DHCP, network booting, etc.).

- **Computing farm** within the ATLAS TDAQ network: Provides the necessary processing power for the analysis of scan data. The hardware details of the farm still need to be specified. For now we assume n physical computers with m Gigabit Ethernet interfaces each (possibly also equipped with o graphics cards used for GPU computing).

An overview of the hardware components and their arrangement is given in Figure 3.1. The IBL VME crate houses the 14 ROD/BOC pairs. The BOC is connected to the detector front-ends as well as the read-out buffers (ROB) of the read-out system (ROS) via S-Links. The three Gigabit Ethernet interfaces of the ROD and the single board computer (SBC) are connected to the ATLAS network. Timing information, trigger and control signals are distributed to the crate via the TIM card. The fit farm machines are also accessible from the Ethernet network and need not necessarily be physically close to the IBL crate.

[1]The IBL ROD features a third Gigabit Ethernet interface accessible from the Virtex-5 FPGA

Configuration

When initializing the system the following configuration items need to be completed:

- ROD network interfaces: Each interface needs a MAC address as well as an IP address. Currently it is planned to configure the Spartan-6 FPGAs via the Virtex-5.
- Computing farm network interfaces: IP addresses need to be assigned to these interfaces. They could either be statically assigned or for greater flexibility be dynamically leased via DHCP.

The network addresses should be defined after consulting with the ATLAS network administrators and be stored in a configuration database for later access.

Before performing a scan the components of the system need to be correctly configured. We focus on the items relevant from the histogramming step onward:

1. Network destination address (i. e. IP address and port[2]) for the histogram data coming from the FPGA. This information could for example be stored in a register file accessible by the Spartan-6 FPGAs by the Virtex-5.
2. The FitManager (introduced in Section 3.3) should spawn an appropriate number of processes that listen for data from the ROD FPGAs on the previously defined network address.

The successful execution of the configuration steps needs to be confirmed to the correct PixActionsServer on the SBC before the actual scan can be executed. The scan is then steered in a coarse level by the PixActionsServer processes on the SBC — complex tuning processes are not visible on the ROD level, but only on the SBC. The completion of the scan is indicated via IS and the results are published to the Histogram Server.

3.2 Dataflow

On a logical level the histograms are being populated on the ROD FPGAs and then handed over to a fitting application for processing. The chain of events performed by the system for a threshold scan is laid out in Figure 3.2.

32 FE-I4 chips will be connected to one BOC/ROD pair and transfer 8B10B encoded data via 160 MBit/s optical links, resulting in a maximum net input data rate of 4096 MBit/s per ROD. Data transfer rates and volumes for a threshold scan and ToT scan are summarized in Table 3.1 according to [34]. The input rate for the threshold scan is averaged under the assumption that 50% of the charge steps are under the threshold.

[2] irrespective of whether UDP or TCP is going to be used, as will further be elaborated on in Chapter 4

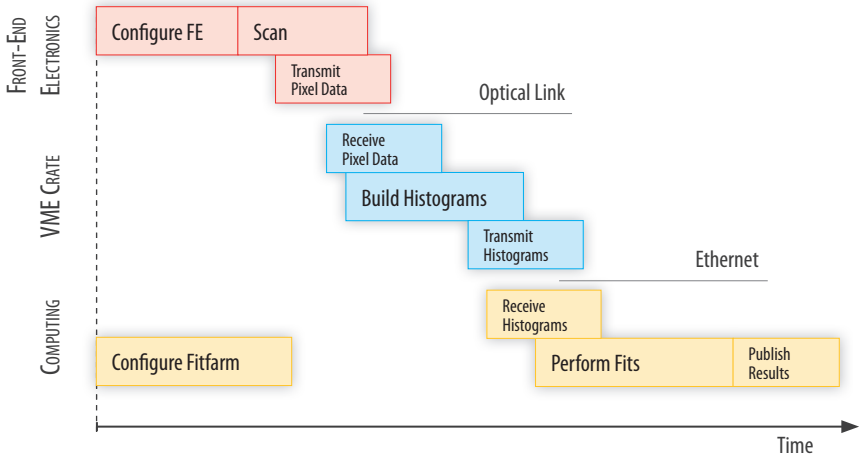


Figure 3.2: The steps of a calibration scan that are being performed by the different components of the system.

	Threshold scan	**ToT scan**
Netto input bandwidth	1024 MBit/s	
Input volume	16.1 MB	32.3 MB
Output volume	215 kB	1.08 MB
Reduction factor	75	30
Output rate	13.7 MBit/s	34.1 MBit/s

Table 3.1: Data rates and volumes produced by 8 FE-I4 (4 modules, 215040 pixels) going into and coming out of the ROD during threshold and ToT scans for one charge step. Values (except reduction factor) per Spartan-6 FPGA are twice as large, for the whole ROD four times as large.

Example data sizes: A threshold scan with $n = 12 \times 10^6$ pixels and $s = 100$ charge steps (with 100 injections per step the histogram value can be stored in one byte) results in $d = n \times s$ Bytes $= 1.2$ GB of histogram data. Assuming a wirespeed throughput rate (and neglecting protocol overhead) of 1 GBit/s and an equal distribution of the data over all 14 RODs (28 Spartan-6 FPGAs respectively) this yields a lower limit of 0.4 s for the transfer time.

When scaling the system we have to consider the possible bottlenecks:

1. Throughput rate for sending the histogram data limited by the ROD side implementation (discussed in Chapter 4).
2. Throughput rate limited by fit farm network interfaces. The number of physical Gigabit Ethernet interfaces as well as the processing power needed to handle the incoming data will most likely be the limiting factors.
3. Processing speed of the histograms on the computing farm. This highly depends on the chosen method for performing the fits and available hardware. The performance of different approaches is compared in Chapter 5.
4. Latency introduced by configuring the fit farm and ROD to handle the data for a certain type of scan.

It is reasonable to associate one FitServer process with the output generated by a ROD network interface, leading to 28 processes that can be distributed across the fit farm machines and their network interfaces. The granularity is fine enough to allow for flexible scaling of the fit farm computers' number of network interfaces and computing capabilities to reach the desired performance. As a benchmark one should take the performance of the current pixel detector during scans [34, p. 1] to not slow down the unified calibration of all four pixel layers after the IBL integration.

The bottlenecks mentioned in the second and third point could thus easily be removed by introducing additional networking hardware (i. e. more interfaces per computing node) or processing power (more/faster servers). Of course the processing speed of the histograms would also directly benefit from the introduction of more efficient algorithms. Points one and four on the other hand cannot be simply improved by deploying more (powerful) hardware, but rather need careful optimization, if they are deemed critical for the overall system performance.

3.3 Fitting Application

The application needs to be embedded in the current PixelDAQ build chain for consistent development and deployment. We will further discuss the implications of this in Chapter 5 when talking about compiling for a GPU environment.

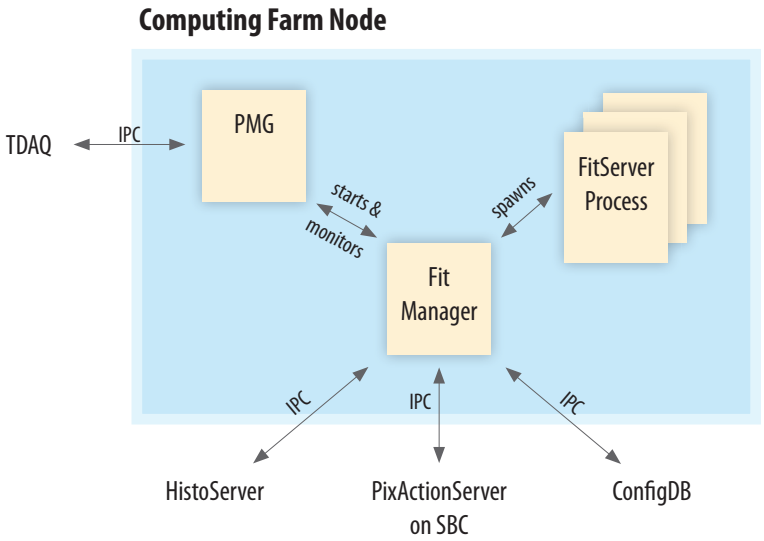


Figure 3.3: The processes on a node of the computing farm.

Figure 3.3 shows the processes being run on a computing node. The process manager (PMG) starts, stops and monitors the FitManager processes. When a scan is initialized the scan configuration is then supplied to the FitManager process via IPC which then in turn spawns the correctly configured FitServer processes that wait for ROD interfaces to sign in and send data. The FitServer processes analyze the incoming histograms and publish the results via IS.

As the histograms generated by the RODs will in most cases not cover a complete front-end but only a part of it (due to mask-stepping), the exact information on which pixels the histogram data belongs to needs to be supplied to the FitServer processes. This could either be realized by adding the information to the network data stream generated by the RODs or be communicated via IS.

In principle the details of how the fits are being performed on the computing farm should be transparent to the FitServer application. This should also apply to a possible computation not on the CPU but on a GPU as being discussed in more detail in Chapter 4 and allows for a flexible improvement and adaption of the fitting process to new hardware. One can achieve this by defining a suitable API for the fitting routine.

3.4 Communication Protocol Format

A central task will be the design of a clearly defined application layer protocol for the communication between RODs and FitServer processes. Main areas to be covered in the protocol specification are:

- Sign-In/Sign-Out procedure for a ROD connecting to a FitServer: Would the ROD need to identify itself or do we assume that the allocation of RODs to FitServers is well known by either FitServer or FitManager?
- Header/Trailer information (if necessary) and format/structure of histogram data
- Error handling

Depending on how the implementation of the transport layer protocol (discussed in the next chapter) will be realized, the protocol might have to deal with undesired data loss or out of order delivery of packets.

4 Moving Histogram Data

"You know, Hobbes, some days even
my lucky rocketship underpants don't help." [1]

To overcome the bandwidth limitations of the VME bus employed by the current ROD, the new IBL ROD features three Gigabit Ethernet interfaces, one of them belonging to a Xilinx Virtex-5 (XC5VFXT70T) FPGA acting as the ROD controller, while the remaining two can be accessed from Xilinx Spartan-6 (XCS6LX150) FPGAs used for histogramming [35]. These network links will be used to transfer data from the ROD to a server farm that further processes the collected histograms.

IP packets need to be generated by the Spartan-6 FPGA before they can be passed to the Ethernet MAC and sent out over the network link. As the throughput is relevant to the calibration performance of the system (including the current pixel system), we need to assess the attainable data rates on the network links. According to Table 3.1 output data rates are 6.9 MB/s for a threshold scan and 17.1 MB/s for a ToT scan per ROD (27.4 MBit/s and 68.2 MBit/s per FPGA respectively). These are average bandwidth requirements and data transfers will likely happen in bursts as the scans are performed in several steps, but this could be well hidden by sending the data while the next histogramming step is being performed.

In the following section the 4-layer TCP/IP reference model and the two transport layer protocol options TCP and UDP will be introduced. Then we will present the embedded system that is used on the FPGA for protocol comparison and performance evaluation. Finally the performance measurements under a Linux and standalone software environment are shown and compared for two different FPGA platforms (Spartan-6 and Virtex-5).

4.1 Network Protocols

In order to reduce the amount of design complexity, network systems use a hierarchy of different protocols that rely upon each other by using well defined services, that each protocol has to provide. These protocols define the behavior of a certain part of the

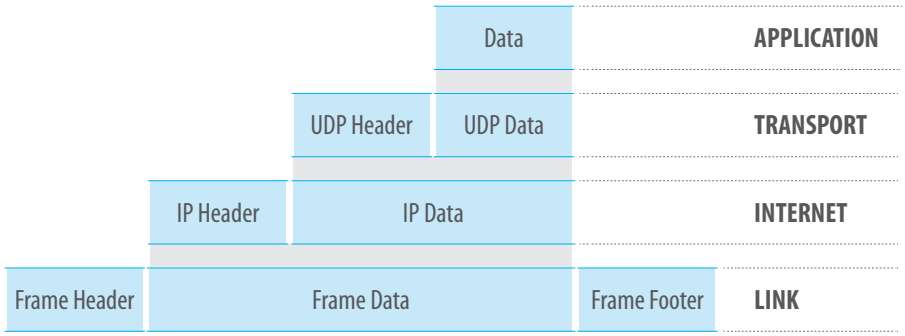


Figure 4.1: The four layers of the TCP/IP reference model.

network system, for example the electrical specifications on the lowest layer (e. g. Gigabit Ethernet on a copper medium 1000BASE-T) or on a process to process communication level (e. g. HTTP).

For this work it is important to understand the services provided by the protocols being used, as well as the advantages and downsides of choosing a certain protocol for the final implementation.

4.1.1 Protocol Layers

There exist different models to describe the protocol hierarchy and abstraction used in today's network communication landscape. In these models one layer relies on the services of the layer below and offers services to the layer on top of it. Protocols acting on a certain layer are not necessarily aware of the surrounding protocols, so that ideally a protocol on one layer can be replaced by another one without impacting the functionality of the complete *protocol stack*.

The well-known OSI model [36] uses seven layers to characterize the hierarchy, beginning with the physical layer at the bottom and ending with the application layer protocol at the top.

We will be using a simpler model defining only four network layers[1], yet well-suited to describe the networking architecture of our embedded system: the TCP/IP reference model [37]. Figure 4.1 shows its four layers and illustrates the concept of encapsulating data payloads.

The services offered by the four layers are as follows:

1. **Link Layer**: Describes how one communicates with hosts on the directly attached network. As the RFC is not very concise on this layer, 5-layer models often add an additional hardware (or physical) layer below the link layer, mimicking the OSI

[1] Some books use a slightly different model definition adding a fifth layer.

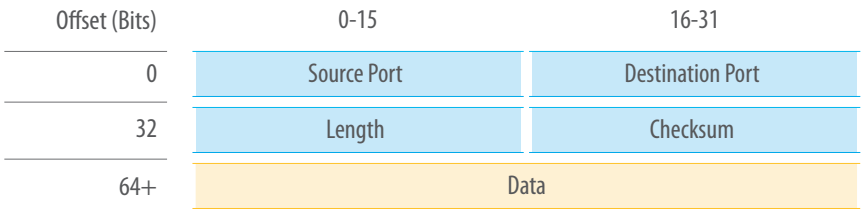


Figure 4.2: Structure of the UDP header.

model.[2] For our purposes, in this layer the low level communication via Gigabit Ethernet over copper (1000BASE-T) is implemented on the ROD interface side, including the generation of Ethernet frames.

2. **Internet Layer**: It is the central layer in the architecture and provides packet delivery across multiple (physically distinct) networks via routing. In the TCP/IP model the Internet Protocol (IP, either version 4 or 6) together with Internet Message Control Protocol (ICMP) is used on this layer. IP is designed as a best-effort protocol that does not guarantee data delivery.
3. **Transport Layer**: End-to-end communication for applications is provided by this layer. Services that might be implemented at this layer include flow and congestion control, stream-orientation, reliability and in-order delivery. Examples for protocols on this layer are UDP and TCP.
4. **Application Layer**: It contains the high-level protocols designed for application to application communication — the ROD to FitServer process communication in our case.

More details on this model as well as an alternative approach with five layers can be found in [38, pp. 67].

Several protocols exist on the transport layer, the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) being the most prominent ones. While UDP offers a simple and fast method of *message* delivery as opposed to TCP's *stream-based* approach, it is a stateless protocol with several downsides compared to TCP.

4.1.2 User Datagram Protocol (UDP)

UDP was defined in RFC768 [39] and is widely used today for example in the domain name system (DNS) or for Voice over IP (VoIP) applications. It is a connectionless protocol designed to send messages (datagrams) with a maximum size of 65,507 bytes rapidly across the network. It is unreliable in the sense that it does not guarantee

[2] The physical layer then defines the electrical and physical attributes of the network (signal strengths, connector design, etc.), while the link layer is concerned with sending data frames of a designed length over the network (Ethernet frames in our case).

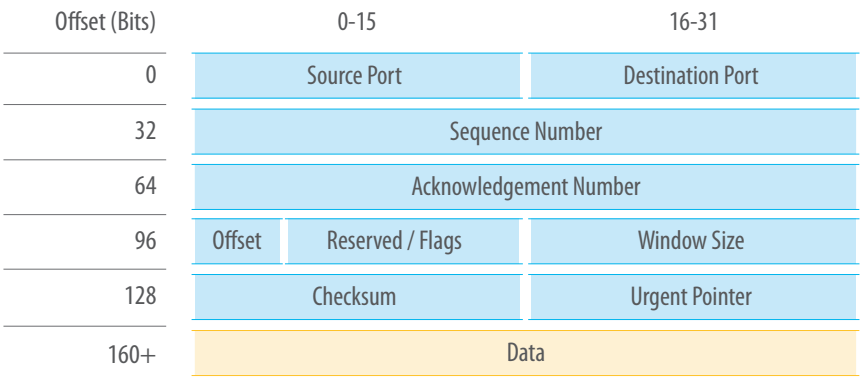


Figure 4.3: Structure of the TCP header.

datagram delivery or delivery in the right order, nor does it in case of packet loss offer a retransmission mechanism. UDP is not able to avoid congestion of the network link by itself and might also flood the receiver as it lacks flow control mechanisms.

UDP uses ports to associate datagrams with a corresponding application process. In total 65,535 ports are identifiable and hence a complete network address consists of the IP address and port number. For error detection purposes an optional checksum can be generated.

As one can see in Figure 4.2 the format of the UDP header is quite simple, only containing source and destination port, as well as the length of the following data block and the optional checksum. Because of that it is easy to implement a peripheral generating UDP datagrams directly in the FPGA hardware. Required features that are missing in UDP such as reliable data transfer would have to be implemented on top of UDP in a custom application layer protocol, that, if again implemented in hardware, would add additional complexity, however.

4.1.3 Transmission Control Protocol (TCP)

Opposed to UDP, TCP is a complex, stateful protocol providing many of the features missing in UDP. It is defined in RFC793 [40] and more functionality was added later on. In contrast to UDP, data is transferred as a byte *stream* and not as datagrams. It also offers a reliable bi-directional connection between applications. TCP is used by many application level protocols that expect reliable data transfer, such as HTTP, SSH, or SMTP.

Like UDP, the network address is made up of an IP address and a port number. A connection between client and server is established via a so-called three-way handshake, requiring the transfer of three packets between the hosts before any data can be sent. Similarly, there is a controlled termination of a connection.

TCP encapsulates the byte-stream into units called *segments* that are then handed over to the protocol (IP) of the underlying layer. In the ideal case the segment can be directly encapsulated again into an IP packet, which in turn gets framed by a header and footer of the link layer protocol and is then sent out over the physical network interface. Because the link layer protocol only accepts data up to a specific size, called the maximum transmission unit (MTU), the IP packet might be fragmented by any router on the way to its destination. Fragmentation is undesired (especially when processing power is limited like in the embedded setup we will introduce later), and therefore a maximum segment size (MSS) can be specified in TCP to avoid fragmentation. Typical values for an Ethernet based network are an MTU of 1500 bytes, resulting in an MSS value of 1460 bytes (assuming a 20 byte header for TCP and IP each).[3]

Other main features of TCP include the sliding window algorithm used for flow control and a set of algorithms that were later specified in RFC793 [40] for congestion control: slow start, congestion avoidance, fast retransmit, and fast recovery. We will not discuss these algorithms and the details on how TCP realizes a reliable connection and therefore refer to the corresponding RFCs and the literature [38, pp. 457].

The price for the high functionality of TCP is paid by introducing additional complexity for the implementation, more protocol overhead, and latency, for example during the three-way handshake. The TCP header depicted in Figure 4.3 reflects the complexity of the protocol.

Because of its complexity it cannot be easily implemented directly in hardware and only a software TCP/IP stack becomes feasible. The performance of this stack however highly depends on the hardware being employed in our embedded environment.

Table 4.1 compares TCP and UDP. For the purpose of transferring data between RODs and the fitting farm nodes, TCP is the protocol of choice, as it already has desired features like reliable data transmission and flow control built in, that would otherwise need to specifically be implemented in the application layer protocol. The disadvantage of running a software TCP/IP stack that possibly results in reduced performance needs to be considered in the context of the requirements for the IBL fitting architecture regarding the ROD output rates.

4.2 Embedded System

The generation of network packets occurs in both of the RODs Spartan-6 FPGAs. As we will be evaluating a software-based approach, we need to make use of a CPU running the

[3] The MSS can be "negotiated" during the TCP handshake by both communication partners. It may also be later adjusted by using an algorithm called path MTU discovery in order to account for smaller MTU values that might occur on router network links between both hosts. This procedure is mandatory for IPv6 based communication, as IPv6 routers will not fragment, but rather directly drop packets that are too large.

	UDP	TCP
Header size (byte)	8	20 - 60
Connection oriented	✗	✓
Reliable transport	✗	✓
In-order delivery	✗	✓
Checksum for data	optional	✓
Congestion control	✗	✓
Flow control	✗	✓

Table 4.1: Feature comparison of UDP and TCP.

networking stack. The Spartan-6 FPGAs do not feature a CPU (unlike some variants of the Virtex or Zynq platform that integrate power PC or ARM cores) and because of that a soft-core CPU needs to be synthesized for use on the FPGA fabric.

Several open source CPU designs exist that will synthesize for the Spartan-6 (OpenRISC, PicoBlaze), but we will use the MicroBlaze core that is supplied and supported by Xilinx. There are two main reasons to use the MicroBlaze: On one hand there is a readily available development platform and documentation, together with a variety of peripherals and drivers that are of great use for putting together a complete embedded system. On the other hand the Linux vanilla kernel has supported the MicroBlaze architecture out of the box for over two years starting with version 2.6.30.[4]

MicroBlaze The MicroBlaze is a 32 bit RISC-architecture[5] soft-CPU developed by Xilinx for use on their FPGA devices. It can be heavily customized to the needs of the target application by defining its properties such as instruction and data cache sizes, use of a memory management unit, use of a floating point unit etc. The clock frequency can be adjusted so that the final design meets the timing constraints. The reference guide [41] gives a detailed overview of the architecture, configuration options, and bus systems of the MicroBlaze.

To complete the usefulness of the embedded system built around the MicroBlaze, IP cores like an interrupt controller, memory controller, timer, ethernet core etc. can be added to the design within the Xilinx Embedded Development Kit (EDK). These cores are then connected to the CPU either via the Processor Local Bus (PLB) or the Advanced eXtensible Bus (AXI) in more recent versions of the Xilinx software. In Figure 4.4 one can see the MicroBlaze processor connected to several peripherals via the PLB and to memory via the LMB (Local Memory Bus) and XCL (Xilinx Cache Link).

[4] OpenRISC support was added to the mainline Linux kernel version 3.1 in October 2011.
[5] Reduced Instruction Set Computer

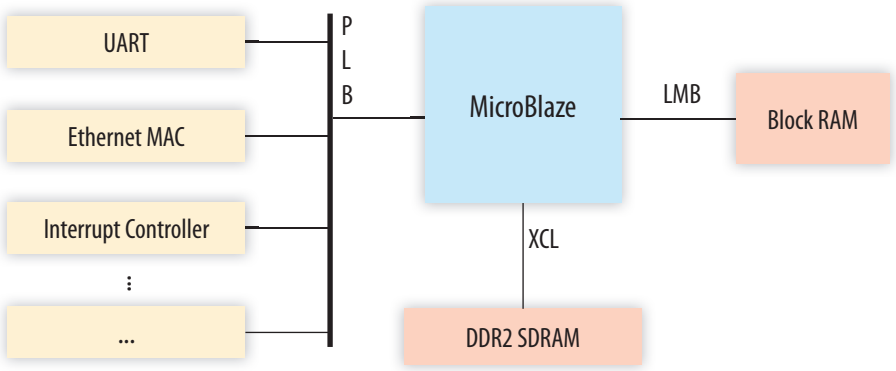


Figure 4.4: An example of an embedded system centered around the MicroBlaze CPU with the PLB connecting the peripheral components, while the Xilinx Cache Link (XCL) and the Local Memory Bus connect off-chip and on-chip memory.

The typical workflow for generating an embedded system consists of creating and modifying a hardware design in the Xilinx Platform Studio (XPS), first. After the design has been successfully "compiled" (or went through the process of synthesis, translate, map, place and route, and bitstream generation to be more precise), a *device tree* can be generated, which describes the hardware environment and is needed for compilation of the Linux kernel. One can also export the hardware design specifications to the Xilinx SDK (a modified Eclipse[6] environment) for application development with or without an underlying operating system (Xilkernel).

As the process of creating a bit-file containing the information to configure the targeted FPGA is rather time-consuming, we employed a simple batch process using the Xilinx command line tools for sweeping over interesting parameters like cache-sizes or clock frequency and creating the corresponding bit-files overnight.

4.3 Performance

The first round of measurements was performed on a system based on a Virtex-5 due to lack of an available Spartan-6 platform. Despite differences in the architecture of the FPGAs we expected the results to be at least in the same order of magnitude as for a Spartan-6 system and thus give an indication of the suitability of the examined approaches. Later we were able to make measurements on a Spartan-6 system and compare the results of both platforms.

Although the measurements made with the Virtex-5 are not directly relevant to the final implementation on the Spartan-6 FPGAs on the ROD, it might still be rewarding

[6] `http://www.eclipse.org`

to compare the performance differences between the two FPGA platforms.

4.3.1 Virtex-5 Setup

Our experimental setup consists of an ML506 development board[7], featuring amongst other components

- Virtex-5 (XC5VSX50T) FPGA
- Marvell M88E1111 Ethernet PHY
- Micron 256 MB DDR2 SDRAM

As this differs from the hardware being used on the IBL ROD, we will discuss implications on the implementation and expected performance in greater detail later on in section 4.3.2. The development board and the receiving workstation (Linux 2.6.35 on an Intel Core 2 Duo T7700 with Intel 82566MM Ethernet controller) are both connected via a switch (Netgear GS108) as no advantage is expected by using a direct connection.

Because the Spartan-6 (and Virtex-5 on the ML506 board) does not offer an integrated PPC (as opposed to the ROD controller), we need to generate an embedded system by making use of the MicroBlaze soft-core CPU and further IP cores supplied by Xilinx's EDK. A basic system targetting the Virtex-5 consists of the following components:

- MicroBlaze CPU running at 125 MHz
- PLBV46 bus
- Hard Ethernet MAC
- Interrupt controller
- Timer
- Multi-port memory controller

The MicroBlaze offers different configuration options allowing for heavy customization to its planned application. We are mostly interested in options regarding the caches of the CPU, due to the expected impact on the network performance, as well as the memory management unit, which is needed in order to run a full-blown operating system like Linux on the MicroBlaze. Other settings (multiplier, barrel shifter, etc.) were left untouched. The Ethernet MAC was configured with TX/RX FIFO sizes of 4096 B for testing of the Linux system and 16384 B for the standalone test — the (partial) checksum offload feature was enabled.

The tests were run on two very distinct software platforms: A current Linux system and a standalone software system combined with an open-source TCP/IP stack. There are some numbers on network performance from different sources and for varying setups

[7] `http://www.xilinx.com/products/boards/ml506/datasheets.html`

Configuration	Throughput (MBit/s) TCP TX	TCP RX	UDP TX
8 kB caches	4.37	2.52	7.34
16 kB caches	4.46	2.77	7.45
Advanced cache	4.75	2.93	8.64
2048 B FIFOs	4.33	2.61	7.39

Table 4.2: Results of the throughput measurements of the Virtex-5 platform with Linux for TCP and UDP. Estimated error < 0.1 MBit/s.

[42, 43, 44, 45]. You therefore have to be careful when comparing these numbers or jumping to conclusions regarding your own setup. Furthermore, the used networking hardware such as switches and network interface cards is not identical to the hardware used in the final implementation, yet one would not expect a striking influence on the achievable performance by these differences.

From above sources one can assume two things: First, the MicroBlaze will not be able to deliver wirespeed on a GbE link for any higher-level protocol like TCP or UDP; according to [42] generating Ethernet frames is possible at wirespeed, though. Second, we expect a huge difference for the Linux system compared to the standalone system: Measurements on a commercial embedded Linux version (Petalinux) indicate around 11 Mbit/s TCP speed [44], while for a standalone system rates between 104 (LwIP stack) and 531 Mbit/s (Treck stack with jumbo frames) can be expected [45].

Linux System

For the first test a Linux 3.0 vanilla kernel was compiled for the MicroBlaze architecture and complemented by the BusyBox[8] environment. The network throughput of the system was measured with iperf[9] version 2.0.5, which was used on the embedded system (required cross compilation as the GNU toolchain was not available on the embedded setup) as well as the desktop computer. Table 4.2 summarizes the measured rates for the different cache configurations of the MicroBlaze.

Instruction and data cache sizes were varied between 8 and 16 kB for the first two configurations. The "advanced cache" configuration used two 16 kB caches, a cache line length of 8 words (instead of 4) and an instruction cache stream with 4 victims.[10] For the last measurement the TX and RX FIFO depths of the Ethernet MAC were reduced to 2 kB with the second configuration as a starting point.

[8] `http://www.busybox.net`
[9] `http://iperf.sourceforge.net`
[10] More information on the caching options of the MicroBlaze can be found in the reference manual [41, pp. 70].

We notice a slight dependency on the cache sizes of the MicroBlaze and the FIFO depth of the Ethernet MAC core. Increasing the cache line length and activating prefetching for the instruction cache improved performance more noticeably. In general UDP outperformed TCP by about 70%.

The measured rates are rather disappointing, but not necessarily unexpected, considering the overhead in the form of context switches, additional system daemons and so forth introduced by the Linux system. Optimizing the kernel as well as the measurement application by supplying the -O3 compiler flag to GCC did not have a noticeable effect. As not even the UDP rates came close to 10 MBit/s, further attempts to increase the speed were abandoned.

Standalone System

The second test was performed in the "standalone" software mode, meaning no operating system was running on the MicroBlaze; rather a minimalistic set of libraries supplying basic software and hardware functionality (drivers, interrupt and timer support, output to serial console, etc.) called board support package or BSP was combined with the open-source TCP/IP stack LwIP (Lightweight IP, [46]). Thus no advanced operating system features like multithreading or memory protection are available in standalone mode. The MicroBlaze was configured with no MMU, but the rest of the underlying embedded system was identical to the one of the previous test.

LwIP offers a RAW and a socket API, with the socket API being very similar to the common Berkeley/POSIX socket API. For the measurements we chose the RAW API nonetheless, as it provides better performance. As initial tests showed promising results with an order of magnitude increase of throughput compared to the Linux system, it was decided to port the most recent LwIP version (1.4.0) for easy access from the Xilinx SDK.[11] This resulted in modifications of data structures for supporting the Xilinx TEMAC drivers and minor changes to the LwIP base code and configuration.

For measuring the throughput, iperf was still employed on the desktop computer, but on the MicroBlaze a modified version of the software supplied with [43] was being executed and 70 MB of data stored in the DDR2 RAM were repeatedly sent to the iperf server application. Several parameters of the LwIP stack can be modified and the following non-standard values were used:

- `tcp_rx/tx_checksum_offload`
- `mem_use_pools`
- `memp_n_pbuf = 32`
- `tcp_wnd = 8192`
- `tcp_snd_buf = 24000`

[11] Xilinx also upgraded the LwIP stack shipped with their software environment (ISE 13.4) from version 1.3.2 to 1.4.0 as of January 2012.

Cache Sizes (kB)		Throughput (MBit/s)
I-Cache	D-Cache	TCP TX
32	32	178
16	16	135
32	16	168
64	16	193

Table 4.3: Results of the throughput measurements for the standalone system based on the LwIP TCP/IP stack.

First the checksum calculation is offloaded to the hardware. The second and third option will enable a pool based memory management instead of an approach making use of heap memory. The fourth and fifth option increase the TCP window size as well as the send buffer size.

Table 4.3 shows the results of the TCP TX measurements with the LwIP stack. The cache configuration was, besides the varying cache sizes, identical to the "advanced cache" setup of the previous measurements under Linux.

As expected the throughput is considerably higher compared to the Linux system. One also notices the strong dependency on instruction and data cache sizes, while varying the instruction cache has a more drastic effect. These rates all meet the throughput requirements mentioned in the introduction of this chapter by a safety factor of 2.

To get a rough idea of the responsiveness of the system, the ping latency to the embedded system was measured, yielding the following results, that clearly show a drop for the system under load:

- 145 ms (idle)
- 885 ms (transferring data)

There is potential to somewhat increase the throughput by optimizing LwIP as well as the Ethernet drivers supplied by Xilinx.

Using a profiler like gprof one can get an overview of the most time consuming functions of the library and adjust the code to fit more closely to the requirements imposed by the ROD data transfer procedures. This could include removing code fragments that are not needed for our setup (like for example calls to the `ip_route` function that determine on which network interface a packet should be sent out — in our setup only one interface exists).

Another noteworthy option resulting in a drastic performance increase is the use of Jumbo Ethernet frames for the ROD-FitServer communication. Ordinary Ethernet frames have a payload limit of 1500 Bytes, while Jumbo Ethernet frames allow an MTU

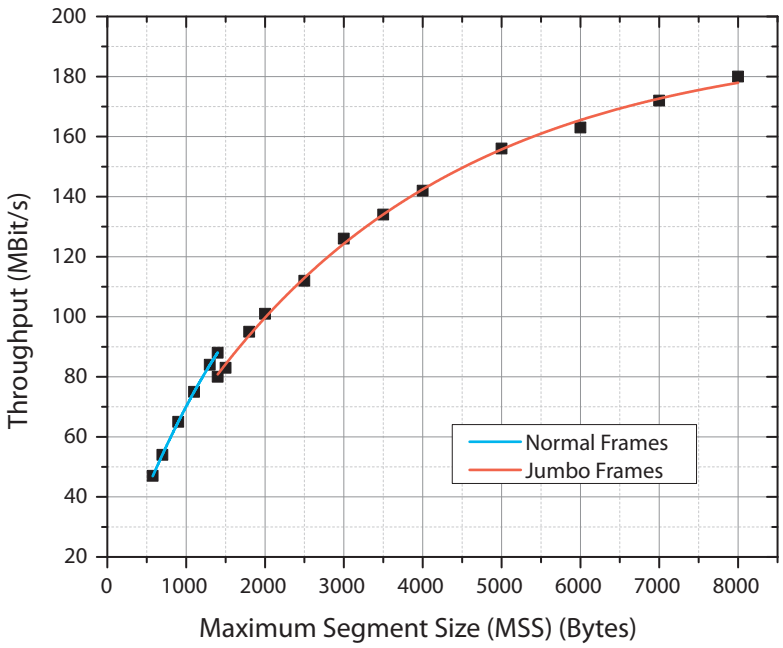


Figure 4.5: Throughput with "standard" Ethernet frames and Jumbo Ethernet frames for different MSS values using the standard LwIP configuration parameters. A linear and an exponential model were fitted to the data.

of up to 9000 bytes. This leads to a reduction of protocol overhead by a factor of six, but more importantly less processing power is needed by hosts and network equipment. Preliminary measurements (Figure 4.5) on the Virtex-5 board showed potential for a 100% throughput increase — still, this might not be feasible due to restrictions concerning ATLAS networking/computing rules, as all hosts on the same subnetwork must support Jumbo frames in addition to the networking hardware.

4.3.2 Spartan-6 Setup

As mentioned before, there are several factors that need to be considered when realizing the discussed setup on a Spartan-6.

First of all, newer embedded systems created by EDK will no longer use the Processor Local Bus (PLB), but rather the Advanced eXtensible Interface (AXI, [47]). Connected to that, the MicroBlaze will be implemented as a little-endian CPU to achieve compatibility with the new bus system. This also affects packet handling by the TCP/IP stack, as the stack now explicitly needs to convert to network byte order (big-endian), which should result in a performance hit.

Furthermore the Ethernet MAC [48, 49] on the Spartan-6 will not be readily available on the FPGA like the Virtex-5 Hard TEMAC, but needs to be generated. This will

consume additional FPGA resources, but on a brighter note enable the use of full TCP/IP checksum offload as opposed to the partial checksum offload feature (TCP checksum only) offered by the Virtex-5 hard TEMAC.

Another factor potentially affecting performance might be the use of a different memory (interface), which needs further analysis.

Despite these differences, one does not expect a striking discrepancy of the system performance between the Virtex-5 and Spartan-6 implementation. Nonetheless it makes sense to build a test setup centered around a Spartan-6 to investigate possible problems and also get an impression of the resources being consumed on the FPGA.

For further testing an ATLYS demonstration board produced by Digilent[12] was used offering the following major components:

- Spartan-6 (XC6SLX45) FPGA
- Marvell M88E1111 Ethernet PHY
- Micron 128 MB DDR2 SDRAM

Two different configurations were used for our measurements: one system was based on the PLB while the other one implemented the AXI bus already discussed earlier. Table 4.4 summarizes the main differences between these two configurations.

The embedded systems created for the measurements on the ATLYS board were similar to the ones used on the ML506 board before – the BSB wizard supplied by Digilent was used to create the AXI and PLB systems and unnecessary IP cores (AC97, USB, etc.) were removed from the designs. One difference is the smaller RX and TX FIFO size of 4 kB compared to the 16 kB FIFOs on the ML506, as designs with 16 kB FIFOs would not achieve timing closure. We also varied the clock rates of the MicroBlaze CPU — for the previous measurements on the ML506 board the clock rate was fixed at 125 MHz.

The results of the throughput measurements for the different implementations can be found in Table 4.5. We can clearly see the impact that a larger instruction cache also has on the AXI implementation by comparing design 1 to design 2: the throughput increases

[12] `http://www.digilentinc.com/atlys/`

	AXI	**PLB**
TE-MAC	AXI_ETHERNET	LL_TEMAC
Checksum offload	full	partial
Endianness	little	big
Memory controller	AXI_S6_DDRX	MPMC

Table 4.4: Differences of AXI and PLB implementation parameters.

Design	Bus	Clock (MHz)	I-Cache (kB)	I-Cache Streams	TCP TX (MBit/s)
1	AXI	100	16	0	100
2	AXI	100	32	0	130
3	AXI	100	32	1, 4 victims	137
4	PLB	83	32	0	128
5	PLB	83	32	1, 4 victims	137
6	PLB	100	32	1, 4 victims	160

Table 4.5: Throughput of AXI and PLB implementations. Options for all designs: D-Cache 16 kB, cache line length of 8 words, branch target cache enabled.

by 30%. Enabling the instruction cache stream feature again results in a performance improvement, although not as drastic. We notice that a PLB system running at 83 MHz seems to be comparable to its AXI counterpart running at 100 MHz (designs 2 vs. 4, and 3 vs. 5 respectively).

Comparing the throughput rates of design 6 (160 MBit/s) to the rates of the corresponding implementation on the ML506 (168 MBit/s) and compensating for the higher clockrate of 125 MHz, the performance of the Spartan-6 PLB implementation is 20% better. One possible cause could be the different memory interfaces being used on the two boards.

Figure 4.6 shows the throughput for varying the MicroBlaze clock rate of design 3 from Table 4.5. The available clock rates were constrained by the capabilities of the clock generator. The dependency can be assumed to be almost linear in the region of interest between 50 MHz and 150 MHz, offering a fast way to gain performance by simply increasing the clock speed as much as possible (i.e. the design must still meet timing constraints).

Device Utilization

As the resources on the FPGA such as block RAM (BRAM), registers, and look up tables (LUTs) are limited, it is fruitful to get an impression on how much fabric will be consumed by the embedded system. The following configuration options of the Xilinx tools[13]

- `map -pr b -ol high -global_opt speed` and `par -ol high -xe n`

[13] more details on the command line tools can be found in [50]

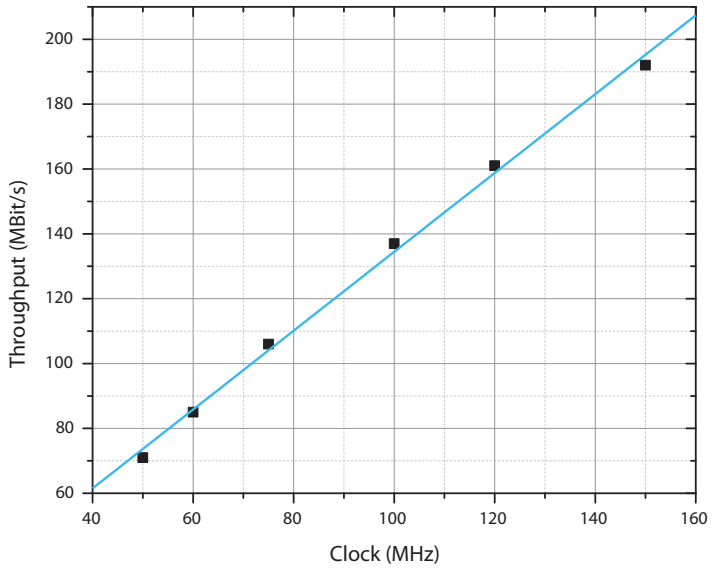


Figure 4.6: Throughput vs. MicroBlaze clockrate of design #3 from Table 4.5.

led to these numbers for the implementation of design 3 running at 150 MHz:

- Slice registers: 11950 (21%)
- Slice LUTs: 14288 (52%)
- RAM16BWERs: 48 (41%).

The large portion of BRAMs being occupied is mostly caused by the MicroBlaze cache (32 kB I-Cache and 16 kB D-Cache) accounting for 28 blocks of BRAM. We also note that the relative numbers are for the Spartan-6 LX45, while the ROD will use a Spartan-6 LX150 with roughly 2-3 times the amount of available resources.

4.4 Discussion

We measured the throughput performance on a Virtex-5 and Spartan-6 based platform, using a software TCP/IP stack on the MicroBlaze soft-core CPU. Running a full-blown Linux operating system on the MicroBlaze resulted in disappointing rates of well below 10 MBit/s even for the most optimized embedded design we could find (TCP TX: 4.75 MBit/s, TCP RX: 2.93 MBit/s, UDP TX: 8.64 MBit/s).

By making use of the LwIP TCP/IP stack, much higher throughput rates were achievable, complying with the requirements on network transfer rates introduced in Chapter 3 with a sufficient safety margin. As measurements were initially carried out on a Virtex-5 platform, we discussed the implications of a migration to a Spartan-6 based system. We finally measured the throughput on the ATLYS board featuring a Spartan-6 and confirmed that similar rates as on the ML506 board were realizable. Looking at the rates

that were realized in [43] for a slightly different Spartan-6 system (SP605 board with DDR3 RAM), the used setup yielded better TCP TX performance for the AXI based system (130 MBit/s vs. 102 MBit/s) as well as for the PLB based designs (128 MBit/s vs. 81 MBit/s).

Preliminary measurements on the second iteration of the redesigned IBL ROD card confirmed TCP throughput rates of 160 MBit/s, proving the feasibility of the approach on the ROD.

The measurements clearly indicate that the network performance heavily depends on the cache sizes of the MicroBlaze and is almost directly proportional to its clock frequency. With Jumbo Ethernet frames rates well above 200 MBit/s were reached, but more detailed measurements and optimization were abandoned due to concerns regarding the practicability of the approach in the ATLAS networking environment.

There exist some starting points if an additional performance increase is required: The LwIP stack's codebase as well as some of its configuration parameters could be optimized. Additional work might be put into the design of the embedded system to achieve timing closure for even higher MicroBlaze clock frequencies, which would directly translate into a better throughput rate.

This chapter focussed only on optimizing the performance of the ROD network interfaces side of communication, but not the receiving side of the computing farm machines that will have to deal with a lot higher throughput rates. In case a bandwidth of 200 MBit/s per ROD FPGA is realized in the final setup, the fit farm would have to deal with a total incoming data rate of 5.6 GBit/s (28 Spartan-6 network interfaces on the RODs), which might need further consideration as soon as the explicit hardware architecture of the computing farm has been decided upon.

For the purpose of stress testing the stability and reliability of the network connections one could envision introducing artificial packet loss and additional latency in the communication channel to analyze the effects on data transfer. The Linux tools `tc` and `iptables` could provide the necessary functionality for such a test.

5 Fitting of Calibration Histograms

"This one's tricky. You have to use imaginary numbers, like eleventeen." [1]

The previous chapter dealt with transferring the histogram data that was generated on the RODs to the fit farm computers, where further processing will take place. As the fit farm replaces the SDSPs that were used on the previous ROD, their functionality needs to be migrated. We will focus on the histograms resulting from a threshold scan or the threshold tuning procedure. An S-Curve type function needs to be fitted to the histogram data in order to determine threshold and noise of a pixel, as well as the threshold dispersion of a complete front-end chip.

In the next section an overview of the general problem of threshold histogram fitting is given. As we want to accelerate the fitting process with the help of GPUs, the following section will introduce the general architecture of GPUs and the approach for programming them, also highlighting some potential pitfalls. The next part of this chapter deals with the porting of the DSP fitting algorithm and its adaption to take advantage of the GPU architecture. Performance results for the fitting of threshold histograms will be given for different implementations of the GPU version, further comparing them with additional fitting approaches running on a standard CPU.

5.1 Fitting Theory

By fitting observed data to a model function $f(x; \boldsymbol{\beta})$ we want to find the n parameters contained in the vector $\boldsymbol{\beta}$ that best describe the N measured values y_i. In the case of the histograms resulting from threshold scans or the threshold tuning procedure an S-Curve type function

$$f(Q; \mu, \sigma_{\text{el}}) = \frac{1}{2} a_0 \left(1 + \text{Erf}\left(\frac{Q - \mu}{\sqrt{2}\,\sigma_{\text{el}}}\right)\right) \tag{5.1}$$

as already derived in Section 2.3.4 needs to be fitted to the histogram data. The factor a_0 denotes the total number of injections for one charge step.

There are two main approaches to function fitting: the method of least squares and the method of maximum likelihood [51, pp. 85], [11, pp. 351].

Least Squares Using the least-squares method one wants to find a $\boldsymbol{\beta}$ that minimizes the squared sum of residuals

$$\chi^2 = \sum_{i=1}^{N} w_i \, r_i^2 \tag{5.2}$$

with the residual

$$r_i = y_i - f\left(x_i; \boldsymbol{\beta}\right) \tag{5.3}$$

being defined as the difference between observed values y_i and the model function f evaluated at that point x_i. Each data point can optionally be weighted by a factor w_i.

One can further define the reduced χ^2 as a goodness of fit parameter

$$\chi^2_{\text{red}} = \frac{\chi^2}{\nu} \tag{5.4}$$

with the degrees of freedom $\nu = N - n$ of the fit being the difference of the number of available data points and the free parameters of the model function (the size of the vector $\boldsymbol{\beta}$). The reduced χ^2 therefore accounts for the complexity of the model as well as for the size of the available dataset used for fitting.

Values of $\chi^2_{\text{red}} \approx 1$ indicate that the model fits the data reasonably well, while values smaller than 1 show that the model might be overfitting the data. On the other hand, values greater than 1 are an indicator of the model not capturing all of the structure in the data.

The downside of least squares fitting is that it requires Gaussian errors for all data points, an assumption that does not strongly hold in the case of a threshold scan histogram. Bin values there follow a binomial distribution, that can only be well-approximated by a Gaussian distribution under certain conditions[1].

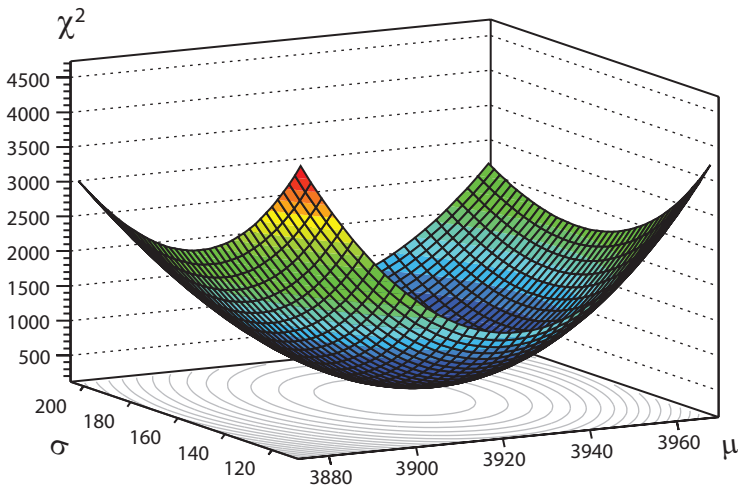


Figure 5.1: The χ^2 hypersurface of a simulated pixel threshold scan histogram as a function of the two parameters σ and μ.

Maximum Likelihood The method of maximum likelihood is more flexible than the least-squares method and is the method of choice for low statistics data samples and non-Gaussian probability distributions. The principle idea is to calculate the probability of finding the measured data in a certain model and then to maximize this value by varying the model parameters. Given a set of measurements y_i of the independent variable x_i and its corresponding probability density (or mass) functions (PDFs)

$$P_i(\boldsymbol{\beta}) = P(x_i;\, \boldsymbol{\beta}) \tag{5.5}$$

we define the likelihood function

$$\mathcal{L}(\boldsymbol{\beta}) = \prod_{i=1}^{N} P_i(\boldsymbol{\beta}) \tag{5.6}$$

as the product of the individual PDFs. For convenience (and safer computation because of rounding issues when multiplying many small numbers) one uses the log-likelihood

$$M = \ln \mathcal{L} = \sum_{i=1}^{N} \ln P_i(\boldsymbol{\beta})\,. \tag{5.7}$$

In the case of a Gaussian PDF the maximum likelihood approach is equivalent to the least-squares method:

$$\ln \mathcal{L} \propto -\chi^2\,. \tag{5.8}$$

[1] according to the de Moivre-Laplace theorem

5.1.1 Minimization Algorithms

Regardless of which method is chosen to determine the parameter set that best describes the data, one needs to either minimize (χ^2 approach) oder maximize (log-likelihood approach) a function that is non-linear in its parameters, like the hypersurface depicted in Figure 5.1. Several numerical algorithms exist to find a *local* extremum that can be used to solve this problem.

Common to all of these algorithms is that they require an initial guess of the parameters as a starting point for their iterative search of the extremum. The algorithm might be robust or susceptible to parameter guesses that result in a starting point far away from the extremum. The minimization algorithm either terminates after a defined number of maximum iterations were performed, or if the convergence criterion is met.

One of the simplest methods is the *grid search* algorithm, that varies one parameter of the objective function until it crosses a "ravine" (this vivid description of course only works in the case of objective functions with two free parameters) and approximates the function in this parameter direction by a second degree polynomial to determine the parameter value corresponding to the minimum. Then, the procedure is repeated with the next parameter and so forth until the fit converges.

More elaborate algorithms make use of the gradient information in order to converge more rapidly towards a minimum, but hence do require values for the first derivative of the function. The *Levenberg-Marquardt* algorithm uses gradient information for the initial phase of the minimization, before smoothly switching over to an analytical approximation of the function's surface for faster convergence behavior [52]. For details and an extensive mathematical description of common minimization algorithms we refer to the literature, for example [53], [54, pp. 148], or [55, pp. 686].

5.2 GPU Computing

Starting in 2005, there has been a trend in changing the architecture of CPUs, from only having a single processing unit to nowadays employing up to ten so-called processing *cores* like the Intel Xeon E7. This step had become necessary, as increasing the CPU clock speed as a means of gaining performance was no longer feasible. In the future, designs containing even hundreds of cores can be expected. Opposed to the CPU market, current GPUs already offer over thousand cores in one chip due to their special application in computer gaming and have therefore become attractive as a computing platform.

At the beginning of general purpose computing on GPUs (GP-GPU), GPUs had to be programmed by "abusing" graphics programming APIs like Direct3D or OpenGL. Nowadays there exist two major frameworks that enable the straightforward development of applications for GPUs — CUDA (Compute Unified Device Architecture) and OpenCL

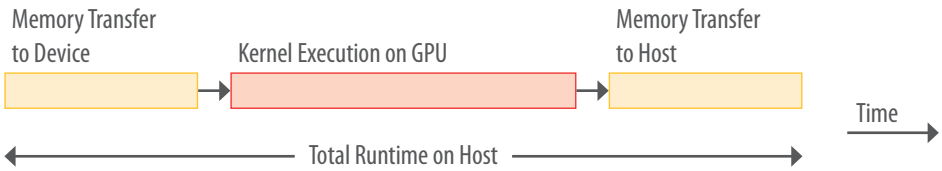


Figure 5.2: Total runtime of a program on a GPU with overhead due to memory transfer to and from the main memory of the graphics card. More recently, multiple kernels can be executed in parallel, also overlapping with memory transfers.

(Open Computing Language). There are also two main companies involved in the GPU market, NVidia and ATI/AMD. While OpenCL can be used on GPUs by both firms, the use of CUDA is restricted to NVidia's products.

Over the past years the flexibility to program GPUs has risen dramatically, so that nowadays they are even complementing CPUs in supercomputing clusters [56], also due to the fact that GPUs can be more energy-efficient when considering the number of operations per second performed per watt of dissipated power (FLOPS/watt).

While programming for highly parallel architectures is still in its infancy, there is a strong interest in exploiting the computing power of current GPUs by exposing parallelism in applications to benefit from a performance boost on the GPU. This might enable calculations that previously had to be performed on clusters to now run on a single GPU in a comparable time frame.

In order to effectively exploit current and future manycore architectures (be it GPU or to a lesser degree CPU), the major task is to find ways to parallelize compute intensive algorithms.

5.2.1 Fundamentals

The main advantages of a GPU are its high memory bandwidth of well over 100 GB/s for high-end models and the enormous theoretical operation throughput ranging above 1000 GFLOPS. Compared to state of the art CPUs containing four to ten cores, GPUs have hundreds of cores built in. The challenge lies in effectively using the vast number of cores by writing or changing algorithms that fit the architecture.

Memory transfers to and from the graphics card's main memory are expensive operations that need to be accounted for when measuring the performance of an application. Because of that, only more complex computations involving several operations on a dataset can be considered candidates for an implementation on GPUs. Figure 5.2 visualizes this issue.

Due to the fact that the original field of application of GPUs did not require precise results, only floating point arithmetics with single precision were supported for a long time. Today's architectures also support double precision operations, although accompanied by a significant decrease of performance[2].

Before implementing an algorithm for a GPU, one has to consider the potential benefit first: Is the GPU architecture a fitting match for the structure of the algorithm? Can the needed computations be effectively carried out on the GPU and are there limitations regarding the available memory?

A major point to bear in mind is how much of the code can be parallelized to take advantage of the many cores of the GPU. *Amdahl's law* describes the fact that the achievable speedup factor $S(n)$ by using many computing cores n is limited by the percentage of code p that can be parallelized [57, p 195]:

$$S(p,n) = \left((1-p) + \frac{p}{n}\right)^{-1} \tag{5.9}$$

with the maximum speedup

$$\lim_{n\to\infty} S(p,n) = \frac{1}{1-p}\ . \tag{5.10}$$

Figure 5.3 illustrates Amdahl's law. For programs that allow only for a limited degree of parallelization, a CPU might be the better target architecture, offering a rich instruction set and being more optimized for sequential operation.

5.2.2 Compute Unified Device Architecture (CUDA)

With the release of the GeForce 8800 GTX GPU based on the G80 architecture in 2006, NVidia unified the device architecture by shifting the functionality of the GPU's separate vertex and pixel pipelines, used for graphics calculations, to programmable processing units. This did not only allow the dynamic partitioning of the functionality depending on the graphics application requirements, but also made it possible to program the GPU using the C language with some restrictions.

To make NVidia GPUs programmable with widely used programming languages and not rely on descriptions on the level of shaders, textures etc. the CUDA hardware and software architecture was developed by NVidia. Programs written in many everyday languages (C, C++, Fortran, etc.) can be executed on the GPU with some minor modifications — still optimization of already existing code fragments is necessary to take

[2] Interestingly enough, this decrease of performance is to some degree not due to the design, but rather to an artificial limitation in GPU models intended for the "consumer" market like the NVidia GeForce GTX480. NVidia's "professional" Tesla series does not have this restriction.

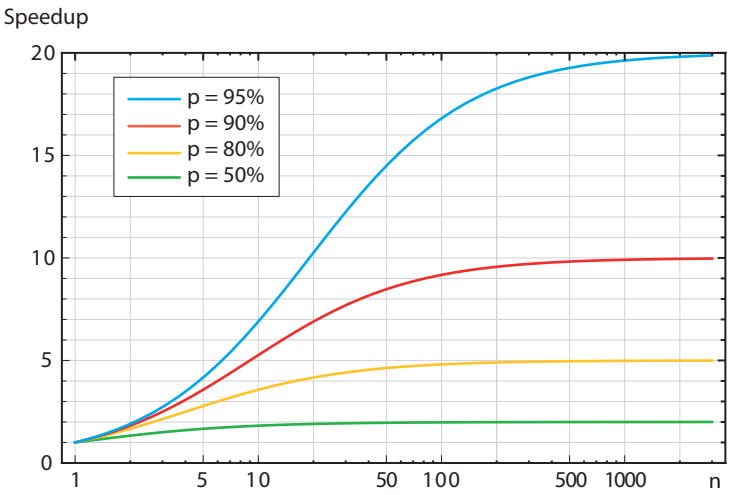


Figure 5.3: Achievable speedup through parallelization according to Amdahl's law for different values of percentage p of parallelizable code and available computing cores n.

advantage of the parallel architecture. NVidia provides documentation for programming CUDA devices with C, including best practice programming examples [58, 59, 60].

The availability of CUDA for Microsoft Windows, MacOS X, and Linux and its relatively smooth learning curve have led to its widespread use. The current version of the CUDA software framework is 4.2 and it includes drivers, (mathematics) libraries, and compilation, debugging and optimization tools — later measurements were carried out using version 4.0, though.

While there exist plenty of GPU variants, based on different architectures, the *compute capability* specifies the hardware's capabilities by providing some kind of versioning scheme for an easier overview. The compute capability not only specifies available features like double precision floating point arithmetic or atomic functions, but also defines hardware properties like certain memory sizes for caches and shared memory. For compute capability 1.x, for example, CUDA did not support C function pointers and recursion. A list of GPUs and their compute capability can be found in [58, pp. 115].

A function that runs on the CUDA device is called a *kernel*. CUDA employs a hierarchy of program execution units, which are from the bottom upward:

- **Thread**: A thread is the smallest unit of execution, executing an instance of a CUDA kernel. It has access to private registers.

- **Block**: Threads are grouped in blocks and can be identified by an index. Threads within the same block are concurrently executed and can make use of shared memory to access and exchange data for cooperation.

- **Grid**: Thread blocks themselves are grouped together in an array called the grid and are denoted by a block index. Communication between threads of different blocks is possible via the global memory.

This hierarchy allows for transparent scaling of programs to different GPU architectures.

There are constraints on the maximum allowed sizes for blocks and grids, depending on the CUDA compute capability. For the Fermi architecture (see next section) a thread block may contain up to 1024 threads, while a grid can be comprised of up to $65{,}535^2$ thread blocks. The smallest unit for execution on a GPU is called a warp and is made up of 32 threads. Threads from the same thread block can access and exchange data via shared memory, while a single thread can use its own private memory as well as registers.

5.2.3 GPU Architecture

As the later focus will lie on an implementation using CUDA, the description of GPU architecture will use a current NVidia GPU based on the Fermi architecture for illustration purposes. Still, a lot of the information is also valid for AMD GPUs.

The GPU is divided into Streaming Multiprocessors (SMs), each containing 32 CUDA cores. Depending on the chip yield (the GPU model), up to 16 SMs are available, totaling 512 CUDA cores. Each core consists of an ALU and an FPU, supporting single and double precision floating point operations. An SM is complemented by four special function units, that execute functions like `sin`, `log`, or `Erf`.

The hierarchy of execution units introduced in the previous section maps to the hierarchy of the GPU in the following way:

- Up to 16 kernels can be launched for concurrent execution on the GPU.
- Thread blocks are executed on an SM and do not migrate to another SM within their lifetime.
- Threads are scheduled in warps of 32 threads for execution on the CUDA cores of an SM. Up to 48 warps can be active at the same time.

Threads within a warp execute the same instructions on different data, realizing data-level parallelism. Consequently, it is unfavorable to have *divergent* pieces of code within those threads (for example due to `if/then` conditional statements), as these diverging code branches can no longer be executed in parallel and a part of the warp needs to be stalled, until a common code path is restored.

Memory is arranged in different levels, starting from registers located in the SMs (32,768 registers) with very fast access times, up to the global graphics memory, that is located off-chip. The available memory bandwidth is immense compared to a CPU,

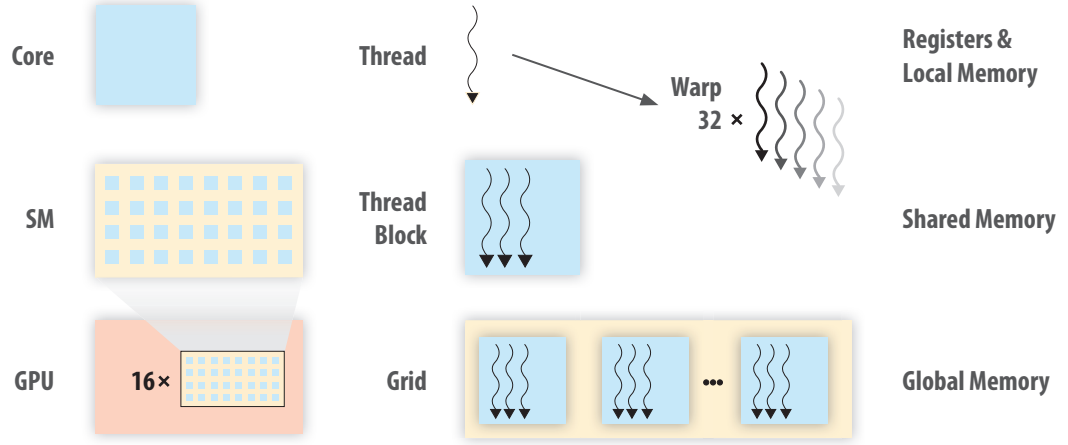


Figure 5.4: CUDA architecture units including processing units, program execution units, and memory hierarchy.

reaching rates of about 180 GB/s by employing six 64 Bit memory interfaces. An overview of the corresponding logical CUDA units, associated memories, and processing hardware is given by Figure 5.4.

Additionally, there is a caching hierarchy with level 1 and level 2 caches. The level 1 cache is realized within the same memory hardware as the shared memory (its size is configurable by reducing the amount of available shared memory: either 16/48 kB cache/shared memory or the opposite), while the level 2 cache (768 kB) is placed centrally on the chip. The introduction of this caching hierarchy is a major concession to make mainstream (super-)computing on GPUs even more attractive, by supporting algorithms that are not able to efficiently hide the huge memory access latency.

In contrast to CPUs running an operating system, context switches are very lightweight and cheap (instantaneous, as registers and shared memory do not need to be saved and restored, but are persistent). Thus it becomes feasible to hide the latency generated by arithmetic instructions or memory accesses by scheduling another warp, yet there must be enough warps available to the scheduler.

The two most recent GPU architectures by NVidia are called Fermi and Kepler (introduced in 2009 and 2012). An overview of the architectures and available features can be found in [61, 62].

5.2.4 CUDA C Programming Interface

In the following we will demonstrate the basics of programming C for CUDA, as the DSP code has already been written in that language.

In order to compile C-like code that targets the GPU, the compile driver `nvcc` is used to separate code to be executed on the host, from code written for execution on the GPU and to further steer the whole process. For compilation of host code, the GCC is called, although there exist restrictions regarding the version (CUDA 4.0 only supports GCC versions < 4.5, CUDA 4.2 supports GCC 4.6) that need to be considered when integrating the GPU application code into an already existing project environment like IBLDAQ.

The part of the code that is targeted for execution on a GPU is first compiled into PTX code (Parallel Thread eXecution), an intermediate language. The PTX code is either directly translated by the assembler `ptxas` into a CUDA binary targeted for a specific architecture, or it may be compiled *just-in-time* during the runtime of the final executable, to take advantage of compiler improvements or even run on architectures that were not yet available at compile time. CPU and GPU object files (or the PTX code) are finally complemented by the CUDA runtime libraries during the linking step.

For recent compute capabilities, most of the features of C are supported, including recursive function calls and function pointers. Functions need to be tagged by keywords like `__global__`, `__host__`, or `__device__` to give `nvcc` the necessary information to split up the source file. Within a kernel function, the `__shared__` keyword is used to place a variable in shared memory. There also exist structures that hold information on the block and thread identifier of each thread (`threadIdx` and `blockIdx`).

An important point to consider is synchronization of threads within a block, that are not scheduled in the same warp, to correctly communicate results via shared memory. A synchronization barrier can be introduced in the code with the `__syncthreads` function, that waits until all threads within a block reach it (and they must reach it, otherwise the behavior of the kernel is undefined).

Listing 5.1 shows a minimalistic code snippet that performs the necessary steps for a CUDA calculation: Allocation and copying of memory, execution of the CUDA kernel, copying of the results and releasing the allocated device memory. The call of the kernel follows a special syntax, with the size of the grid and thread blocks supplied within the `<<< >>>` construct, that gets replaced by the appropriate calls to CUDA runtime library functions during the compilation process.

5.3 Histogram Fitting

In this section we present how the fitting of threshold scan histograms is currently implemented on the ROD SDSPs. The starting points for an implementation of a parallelized GPU version are identified and discussed, before finally the test setup for the benchmarking is presented.

```
1  /* define block and grid size */
2  dim3 blockSize(PROBSIZE);
3  dim3 gridSize(1);
4
5  /* pointers to host and device memory */
6  float *c_mem;
7  float *h_mem;
8
9  /* allocate host memory and device memory */
10 h_a = (float *) malloc(MEMSIZE);
11 cudaReturn = cudaMalloc((void **)&c_mem, MEMSIZE);
12
13 /* copy data to device memory */
14 cudaReturn = cudaMemcpy(c_mem, h_mem, MEMSIZE, cudaMemcpyHostToDevice);
15
16 /* execute kernel on GPU */
17 kernelFunction<<<gridSize, blockSize>>>(c_mem);
18
19 /* copy results to host memory */
20 cudaReturn = cudaMemcpy(h_mem, c_mem, MEMSIZE, cudaMemcpyDeviceToHost);
21
22 /* release device memory */
23 cudaReturn = cudaFree(c_mem);
24 c_mem = NULL;
25
26 /* further processing */
```

Listing 5.1: Code snippet of a CUDA kernel call and related helper functions with omitted error handling.

5.3.1 Current Algorithm

The fitting algorithm used on the SDSPs consists of three major steps:

1. Finding valid bins and guessing initial parameters
2. Maximizing the log-likelihood / minimizing the χ^2 value (both approaches are supported)
3. Calculating the reduced χ^2 value

In step 1 the initial values for the threshold scan fitting problem are guessed in the following way:

1. Find the first histogram bin containing at least one entry.
2. Find the last histogram bin containing less than a_0 entries.
3. Define the range of valid bins by these two boundaries, also including the two adjacent bins.
4. Find the bins where the count crosses the 16%, 50%, and 84% mark and calculate the values x_{16}, x_{50}, and x_{84} using the adjacent bins.

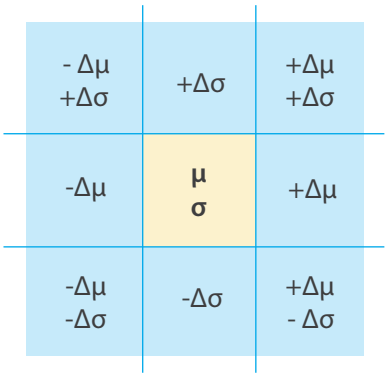


Figure 5.5: 3×3 Search mask containing (μ, σ) value pairs.

5. Estimate the mean $\mu = x_{50}$ and standard deviation $\sigma = \frac{1}{2}(x_{84} - x_{16})$.

In the case of only two valid bins the threshold value can merely be guessed to lie in between both bins and the fitting procedure is aborted.

The basic algorithm of the second step is laid out in pseudocode in Algorithm 1 below. Details of the implementation specific to the DSP, like explicit loop unrolling, are omitted.

The principle idea is to have a search mask of (μ, σ) value pairs with the current optimal value pair at its center as depicted in Figure 5.5. One then calculates the log-likelihood (or χ^2) values related to the parameter pair entries and looks for a new maximum (minimum). The corresponding (μ, σ) tuple is then used as a new center of the mask to seed the values for the next iteration step. In case no new extremum is found, the step sizes $\Delta\mu$ and $\Delta\sigma$ used to generate new (μ, σ) tuples are reduced.

The algorithm terminates, either if the convergence criterion is met, or the maximum number of allowed iterations is reached.

5.3.2 Implementation for GPU

There are three major parts of the fitting process that can be parallelized to make efficient use of a GPU.

- **Parallelism in each pixel**: The histogram belonging to each single pixel can be analyzed independently from the other histograms. Due to the large number of available pixels, this is an obvious starting point for parallelization, not even affecting the fitting algorithm.

Algorithm 1 The fitting algorithm currently employed in the ATLAS pixel detector. Values for $s_\mu,\ s_\sigma,\ \epsilon_\mu,\ \epsilon_\sigma,\ \Delta_{\text{div}},$ and $iterations_{max}$ are predefined.

$\mu \leftarrow \mu_{\text{guess}},\ \sigma \leftarrow \sigma_{\text{guess}}$
$\Delta\mu \leftarrow \mu\, s_\mu,\ \Delta\sigma \leftarrow \sigma\, s_\sigma$
$converged \leftarrow$ **false**
$iterations \leftarrow 0$
while $converged =$ **false and** $iterations < iterations_{max}$ **do**
  $iterations \leftarrow iterations + 1$
  calculate $(\sigma \pm \Delta\sigma,\ \mu \pm \Delta\mu)$ tuples around current (μ, σ) tuple
  calculate log-likelihood/χ^2 for those tuples and fill the search mask
  if new maximum/minimum in search mask exists **then**
    set mask center value to corresponding (μ, σ) tuple
  else
    $\Delta\mu \leftarrow \Delta\mu\, \Delta_{\text{div}}$
    $\Delta\sigma \leftarrow \Delta\sigma\, \Delta_{\text{div}}$
  end if
  if $\Delta\sigma < \epsilon_\sigma\, \sigma$ **and** $\Delta\mu < \epsilon_\mu\, \mu$ **then**
    $converged \leftarrow$ **true**
  end if
end while
return converged

- **Search mask**: Calculating the log-likelihood or χ^2 values belonging to (μ, σ) pairs of the search mask can be done in parallel. The calculation of each search mask cell's value must be finished before the algorithm can decide on the next iteration step. One could also think about resizing the standard 3×3 mask to, for example, a 5×5 mask.

- **Log-likelihood or χ^2 values**: The final value is basically a sum of the results from calculations made independently for each valid histogram bin. These calculations, involving calls of time consuming `Erf` and `log` functions, can also be parallelized and the individual results be added together at the end.

For the above points we have to consider some constraints: Parallelizing over the pixels is a key point, as it is the only way to make use of all of the GPU's SMs, but does not occupy all of the available CUDA cores. On the other hand, it is not efficient to create a high number of threads for the log-likelihood calculation, as the number of valid bins, that are available for parallel computation per pixel histogram, varies, and hence threads might be idle and not perform useful calculations.

In Table 5.1 the theoretical occupancy of warps from a thread block is shown, depending on the number of threads used per search mask cell. We already notice a minimum of unused threads for the case with 7 threads per search mask cell, resulting in 63 threads per thread block and only one unused thread. Later we will measure the

Threads per Cell	3×3 Search Mask			5×5 Search Mask		
	Threads	Unused	Percentage	Threads	Unused	Percentage
1	9	23	72%	25	7	22%
2	18	14	44%	50	14	22%
3	27	5	16%	75	21	22%
4	36	28	44%	100	28	2%
5	45	19	30%	125	3	6%
6	54	10	16%	150	10	9%
7	63	1	2%	175	17	11%
8	72	24	25%	200	24	12%
9	81	15	16%	225	31	2%
10	90	6	6%	250	6	5%

Table 5.1: Saturation of CUDA warps with "productive" threads: Depending on the number of threads that calculate the log-likelihood/χ^2 value of a search mask cell, the warp(s) (each containing 32 threads) might be used more or less efficiently. The percentage of unused threads is shown for a 3×3 and a 5×5 search mask.

performance depending on the number of threads, also taking into account the warp size of 32 threads.

Figure 5.6 shows a histogram of the number of valid bins for a simulated threshold scan (see next section for details on data generation). This is particularly interesting in order to estimate an appropriate number of threads per search mask cell for the implementation. With a mean value of 23 valid bins in connection with the wide distribution, it would obviously be quite inefficient to have a large number of threads available for working on each bin, but not to have enough bins. The choice of thread block size is therefore a tradeoff between a possibly non-optimal utilization of GPU resources (due to lack of available threads) and inefficiencies because of threads not being able to perform calculations (due to lack of data points). The optimum block size will later be determined experimentally.

Our implementation creates a grid of thread blocks that correspond to the pixels of the scan. Depending on the search mask size and the number of threads per search mask cell, the number of threads per block is determined. At the beginning of a kernel execution, the histogram data is copied to shared memory for fast access. The search mask values and results are also stored in shared memory for cooperation purposes.

After the fitting process on the GPU is finished, the resulting (μ, σ) tuples, the χ^2 value, the number of iterations, and the number of valid bins for each pixel histogram are transferred back to the host computers memory.

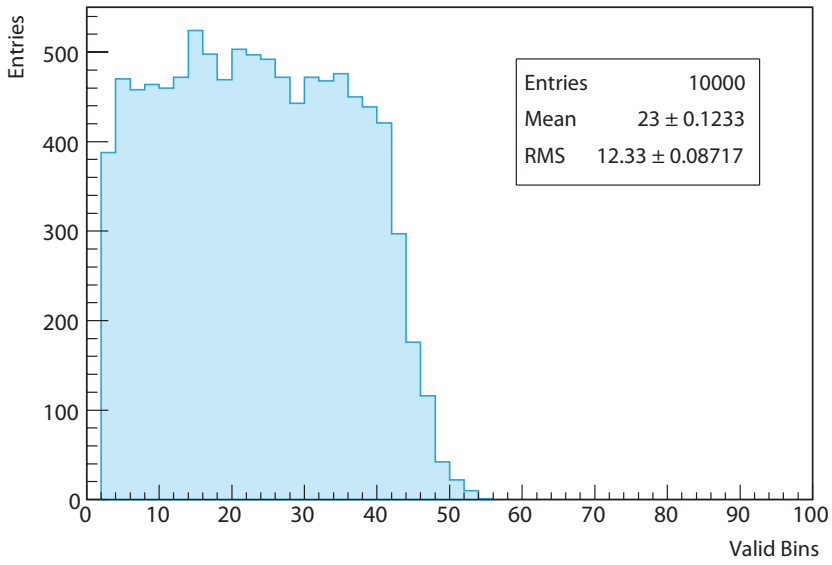


Figure 5.6: Histogram of valid bins that are determined by the prepare step of the fitting procedure.

5.3.3 Measurement Setup

The main goal of this benchmarking is to compare the efficiency of the CUDA approach to the fit problem with the previously employed SDSP hardware approach. Therefore we use the ported version of the DSP code from [34] on the CPU and the original performance numbers for the DSPs as a comparison.

We will also measure the performance of two open-source frameworks: ROOT [63], which is mainly a data analysis and statistics framework used by many particle physics experiments, and LMFIT [64], an open-source library implementing the Levenberg-Marquardt minimization algorithm for least-squares problems.

For benchmarking and evaluation purposes the program `fiteval` has been developed to supply a common interface to the different fitting routines and to harmonize data input and output containing versioning information, enabling later analysis of the results on a different host. The data files contain information on relevant compiler and run time options (single or double precision, code optimization, fit method, runtime, etc.).

Some parameters for measurements can be supplied during the runtime of `fiteval`, while others (GPU optimization options and number of threads, for example) were hardcoded using C preprocessor macros and can only be changed during compile-time by supplying the corresponding parameters during the make process.

Simulated histogram data of a threshold scan was generated by the tool used for the measurements in [34]. The standard dataset contained histograms of 10,000 pixels with a mean threshold of 3950 e^-, threshold dispersion of 40 e^-, and per pixel noise of 58 e^-.

	System 1	System 2	System 3
CPU (Intel)	Core2 Duo T7700 [65]	Core2 Quad Q6600 [66]	Core i7 920 [67]
Clock Frequency	2.4 GHz	2.4 GHz	2.66 GHz
Available Cores	2	4	4
Cache Size	4 MB	8 MB	8 MB
RAM	8 GB	4 GB	8 GB
GPU (NVidia)	Quadro FX 570M [68] G84M	GeForce GTX280 [69] GT200	GeForce GTX480 [70] GF100
Compute Capability	1.1	1.3	2.0
Available SMs	4	30	15
CUDA Cores	32	240	480
Core Frequency	475 MHz	602 MHz	700 MHz

Table 5.2: The three computer systems used for performance measurements of the different software approaches to the fitting problem.

Three different computer setups were used to compare the performance of the approaches, summarized in Table 5.2. On System 1 a desktop environment was running in the background, while System 2 and System 3 were server machines with no significant load other than that generated by the benchmarking. One can generally expect that performance would increase for CPU as well as GPU calculations moving from System 1 to System 3.

5.4 Results

In the following we present the performance results of the fitting variants. We start with a short assessment of the algorithm used to guess the initial parameter values of μ and σ, followed by an evaluation of the ported DSP code running on CPUs. As an alternative to the DSP algorithm, the fitting performance of ROOT and LMFIT is looked into. Finally, the implementation with CUDA is evaluated on three different GPUs with varying implementation details.

Measurements for the CPU based algorithms only made use of a single CPU core. As can be seen in [34], an approach utilizing the OpenMP API for parallel programming and parallelizing over the pixels should scale well with the number of available CPU cores, not only for the DSP code, but also for the ROOT and LMFIT variants. Therefore the performance of the CPU measurements can be scaled by the number of cores to do justice to the used CPUs.

Time intervals were measured with the help of the `clock_gettime` function. Short tests with the Intel C++ Compiler (ICC, [71]) suite on System 1 showed no significant performance gain over the open-source GNU Compiler Collection (GCC, [72]) and

Precision	Fit Time/Pixel (µs)			$\overline{\chi^2_{\text{red}}}$
	System 1	System 2	System 3	
Single	425 ± 20	428 ± 5	335 ± 5	0.5223 ± 0.0052
Double	430 ± 20	437 ± 5	342 ± 5	0.5219 ± 0.0052
Lookup Table	17.1 ± 0.5	15.3 ± 0.1	13.5 ± 0.1	0.5214 ± 0.0051

Table 5.3: Performance of the ported DSP algorithm on CPU for single and double precision, as well as with lookup tables (single precision).

therefore the source code was compiled with GCC 4.4.x for all test runs, with the `-O2` optimization option enabled.

5.4.1 Initial Parameter Guesses

It is rewarding to compare the errors of the fitted parameter values to the values of the initial guesses achieved by the simple algorithm described in Section 5.3.1. The errors of the fitted values of threshold μ and noise σ compared to the original values are shown in the first row of Figure 5.7 for the standard data set of 10,000 simulated pixel histograms. Figure 5.8a shows the distribution of the resulting mean χ^2_{red} values of the problem.

We note the values of $\text{RMS}_\mu = 47.0$ and $\text{RMS}_\sigma = 9.01$ as well as the mean of the distribution of reduced χ^2 values $\overline{\chi^2_{\text{red}}} = 0.94$, that will later be used to compare the performance of the different fitting approaches.

5.4.2 Ported DSP Code on CPU

The DSP code has already been ported for compilation on a CPU for the measurements made in [34]. The code was recycled and adapted for evaluation with the `fiteval` program. There are two main modes of execution for the DSP code: one employs the use of a lookup table (LUT) for fast access to the results of often needed mathematical functions like Erf or log. The second variant uses functions of the C standard library for these calculations.

Because of the small size of the LUT tables (7,000 and 14,000 entries, totaling 84 kB), they can easily fit in today's large CPU caches and should drastically improve the performance compared to the explicit calculation of values. As a downside, the approach is inflexible and might not be as exact.

Errors of the fit as well as the distribution of $\overline{\chi^2_{\text{red}}}$ values are shown in Figure 5.7 and Figure 5.8b. Looking back at the results from the previous section, one can now compare the errors of the fitted (μ, σ) parameters of the initial guessing and the fit.

Framework	Fit Time/Pixel (µs)	$\overline{\chi^2_{\text{red}}}$
ROOT	122 ± 5	0.4217 ± 0.0037
LMFIT	59 ± 2	0.4176 ± 0.0036

Table 5.4: Performance of the ROOT and LMFIT frameworks on System 1.

First, one notices in Figure 5.7 that the error of the threshold value μ does not improve remarkably by employing the fitting routine. Still, the error of the noise value σ improves by about a factor of 3 when looking at the RMS of the distribution. This is an interesting fact, considering that for determining and tuning the thresholds, one does not need the single pixel noise values.

The performance numbers are summarized in Table 5.3 for all three systems (reminder: only *one* thread was used). First of all, one notes that there is no significant difference in the $\overline{\chi^2_{\text{red}}}$ values. The runtime for the double precision implementation is slightly higher than for single precision. As already indicated, the variant using LUTs is considerably faster by over an order of magnitude, because of the efficient use of the large CPU caches. System 3 is the fastest of all compared setups. One further notices that for System 1 the estimated errors are larger compared to the other two systems — this was due to the greater fluctuation in timing results, possibly caused by system load that was generated by background processes.

5.4.3 ROOT & LMFIT

To be able to compare the performance of the "naive" fitting approach used on the DSPs with more elaborate algorithms, two frameworks were used to perform a standard χ^2 minimization of the histogram fitting problem. The original dataset of 10,000 pixels remained unchanged and also the algorithm for guessing initial parameter values was identical.

For ROOT the histogram data from the `fiteval` data structure was used to populate a one-dimensional ROOT histogram object of the `TH1F` class, and then the `Fit` member function was called to perform the fit. As a standard minimizer Minuit [73] is used. Processing is sped up by not creating a new histogram object for every pixel, but instead resetting it via `Reset`, effectively recycling it.

The LMFIT library implements the Levenberg-Marquardt minimization algorithm for generic least-squares problems and supplies several low level functions to perform the fitting.

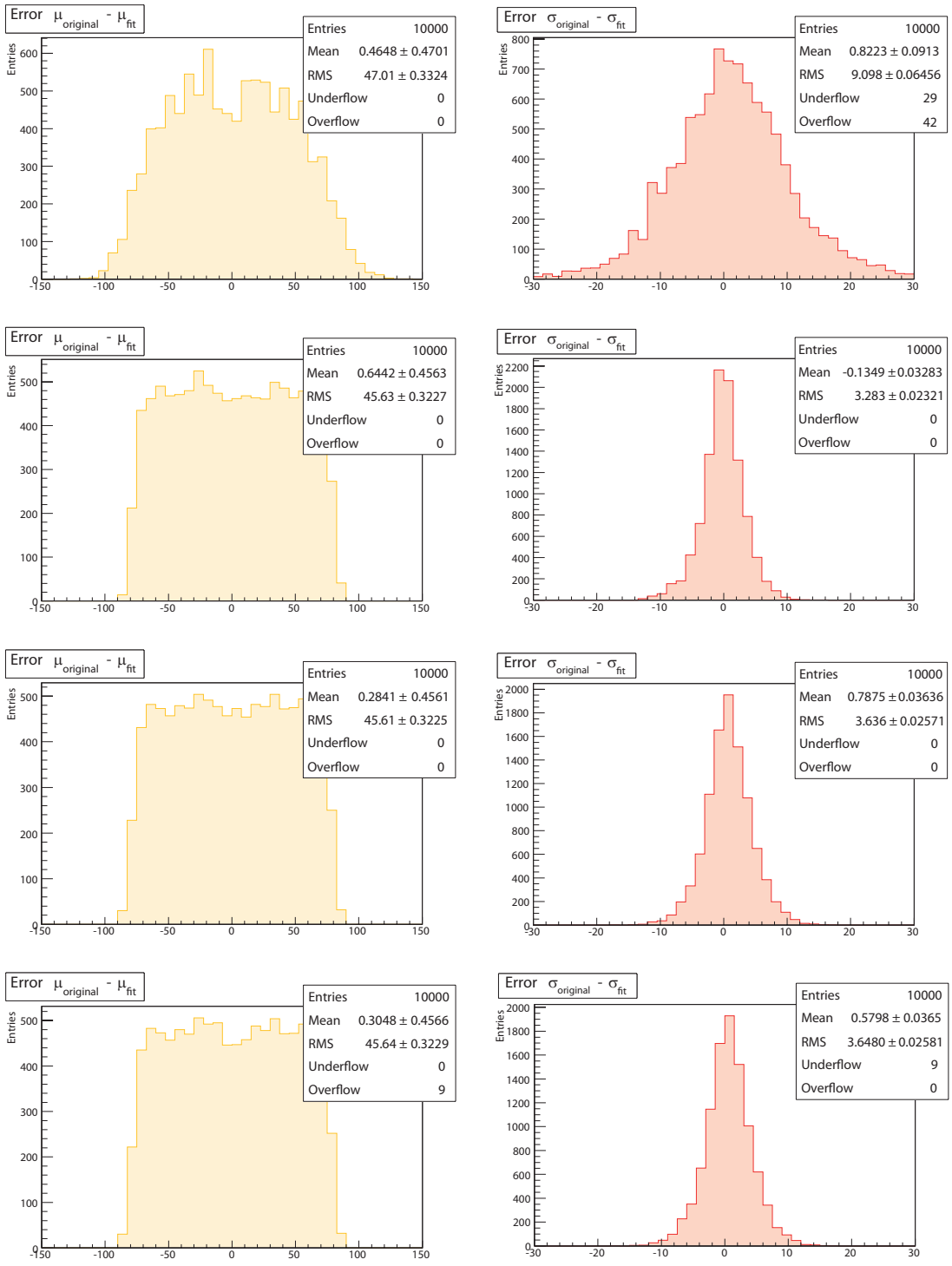
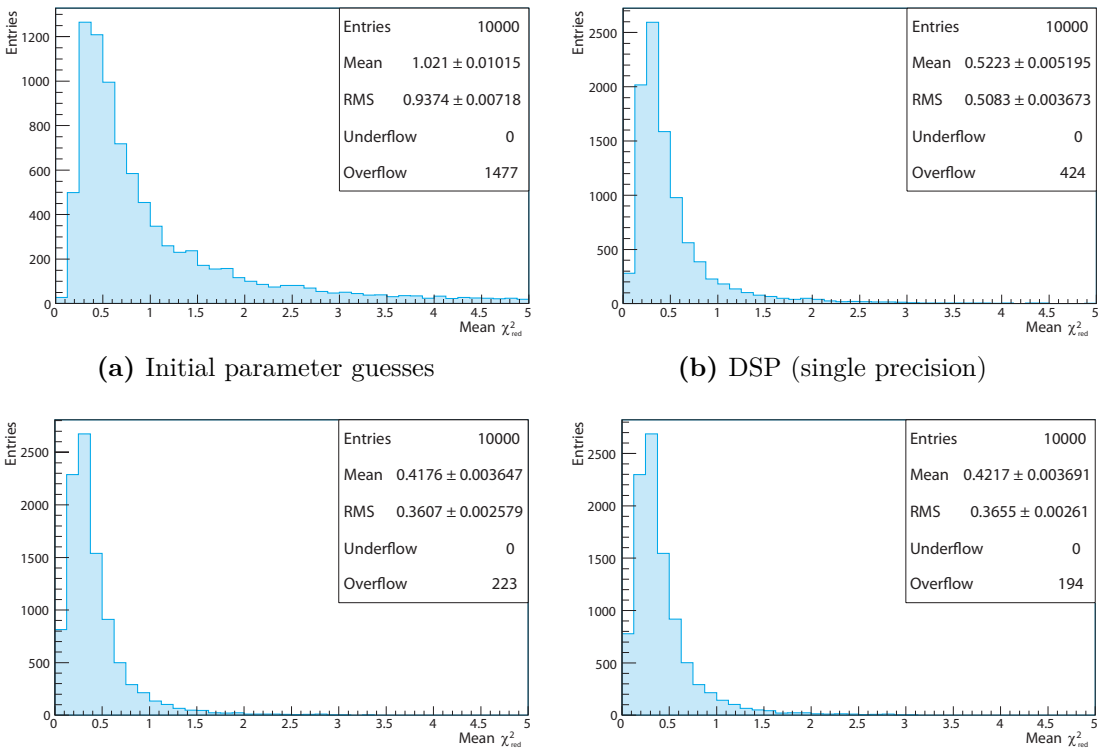


Figure 5.7: Errors of μ and σ for the following algorithms/frameworks (top to bottom): initial parameter guesses, DSP (single precision), LMFIT, ROOT.



(a) Initial parameter guesses

(b) DSP (single precision)

(c) LMFIT

(d) ROOT

Figure 5.8: The distributions of mean χ^2_{red} values for different fitting approaches.

Measurements were only carried out on System 1 and results are shown in Table 5.4, as well as Figure 5.7, Figure 5.8c, and Figure 5.8d. For both packages the standard values to control the fitting process were used. No significant difference in numerical performance can be made out when looking at the values of $\overline{\chi^2_{\text{red}}}$, indicating that the frameworks perform on the same level. Still, LMFIT is twice as fast as ROOT, which is likely caused by the overhead introduced by ROOT with the creation of histogram objects and filling them with data points.

5.4.4 GPU Implementation of DSP Algorithm

The first round of measurements was used to determine a good working point regarding the number of threads per search mask cell (or thread block, respectively). As System 2 and System 3 support the use of double precision floating point calculations, the impact of using a different precision was also investigated — here, we expect a much more drastic impact on the performance in comparison to the CPU.

One must also mention that, although most likely not noticeable when looking at the mean χ^2_{red} values, there might be differences in the accuracy and standard conformance for floating point operations on CPU and GPU, depending on GPU architecture. This issue is negligible in our case and hence we refer to this article [74] for additional information on floating point operations on CPU and GPU.

Another optimizing parameter for a performance increase is the `-use_fast_math` switch for `nvcc`, that speeds up the execution of math functions, with the disadvantage of a lower accuracy [59, pp. 56]. Measurements performed with this setting are marked with an "FM".

3×3 Search Mask

In order to determine an efficient value for the number of threads per search mask cell, we measured the time of the fitting process, while varying the number of threads between 1 and 32. The results of this parameter sweep on System 3 can be seen in Figure 5.9. As expected, the variants employing single precision arithmetic are considerably faster by a factor between about 3-5 than the ones using double precision. The FM option has a noticeable effect on performance regarding single precision calculations, yet the impact on double precision processing is barely observable.

Taking a closer look at the results of the single precision with FM dataset in Figure 5.10a in combination with the percentage of usable threads, we learn two things: For the 3×3 search mask, the optimum number of threads per mask cell is 7. Employing more threads will not further improve performance. Second, one can see the impact that unused threads have on performance, by comparing the progression of the fit times to the percentage of unused threads.

	Precision	Fit Time/Pixel (µs) System 1	System 2	System 3	$\overline{\chi^2_{\text{red}}}$
F	Single	60 ± 1	7.01 ± 0.01	2.34 ± 0.01	0.5122 ± 0.0050
M	Double	-	-	8.85 ± 0.01	0.5122 ± 0.0050
	Single	-	7.13 ± 0.01	3.04 ± 0.01	0.5122 ± 0.0050
	Double	-	34.55 ± 0.1	8.68 ± 0.01	0.5122 ± 0.0050

Table 5.5: Performance of the ported DSP algorithm on GPU for single and double precision, as well as with the FM optimization enabled.

Analysis of the compilation of the fitting program shows that 34 SM registers are used per thread, as well as 1168 bytes of shared memory per thread block. Choosing the approach with 7 threads per mask cell results in 63 threads per block that are divided into two warps. As the number of active thread blocks per SM is limited to 8, the SM can execute a maximum of 16 warps — with an upper limit of 48 active warps per SM, this results in an occupancy of 33%, a number that is confirmed by the CUDA profiler. Therefore the available registers or the amount of shared memory is not the limiting factor for occupancy in the case of 7 threads, but rather the number of thread blocks, that can be active on each SM. A higher occupancy does not generally lead to a higher performance, though, as can clearly be seen in Figure 5.10a for the case of 14 threads per search mask cell.

5×5 Search Mask

By changing the search mask size to 5×5 one can easily "generate" more calculation tasks for execution on the GPU, possibly resulting in a faster fit time. From an algorithmic point of view, this approach calculates more unnecessary intermediate results and tries to use brute force to increase performance, by occupying the GPU more. Yet, Figure 5.10b suggests, that it is still 20% slower (3.31 µs/pixel for the fastest implementation) than the approach employing a 3×3 search mask.

System 2

As we want to be able to quantify the improvements to the GPU implementation to the one from [34], we take a closer look at the measurements, that were performed on System 2. This setup contains a GeForce GTX280, which is based upon the GT200 architecture and not on the Fermi architecture. The corresponding parameter sweep over the number of threads per search mask cell can be seen in Figure 5.11.

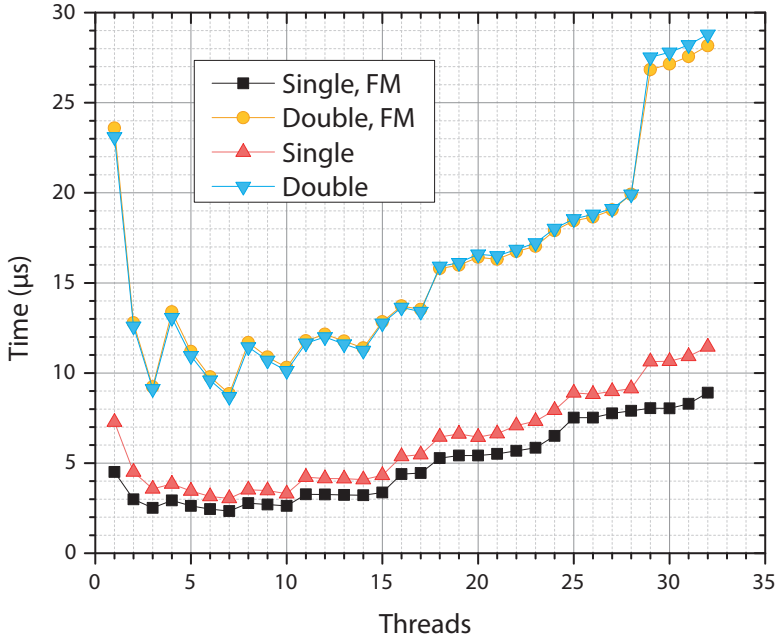


Figure 5.9: Fit times on System 3 for single and double precision fits using a 3×3 search mask depending on the number of threads per search mask cell.

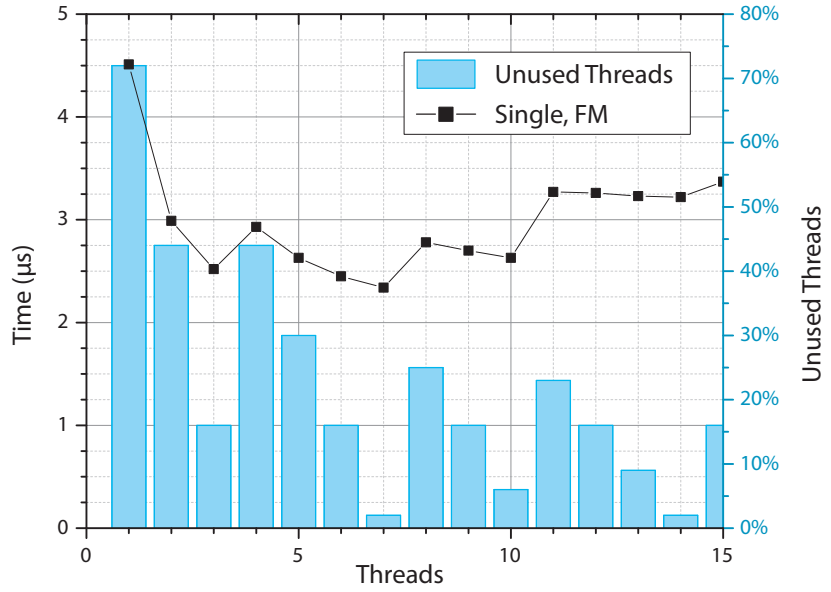
The single precision variant again outperforms the one that is using double precision arithmetics by about a factor of 5. With a peak performance of 7.01µs/pixel, the GeForce GTX480 is three times faster for our problem than the GeForce GTX280.

If not explicitly mentioned otherwise, all subsequent results were produced on System 3 with a 3×3 search mask, 7 threads per mask cell, single precision, and the `fast_math` setting enabled. Performance numbers for all systems are listed in Table 5.5: As the architecture of the GPU of System 1 did not support double precision, only a single precision measurement was made. System 3 outperforms System 2 by a factor 3, while System 2 is still almost 9 times faster than System 1 for the single precision. We cannot see significant differences in the numerical quality of the fits, indicating that the single precision approach with enabled fastmath mode is the method of choice.

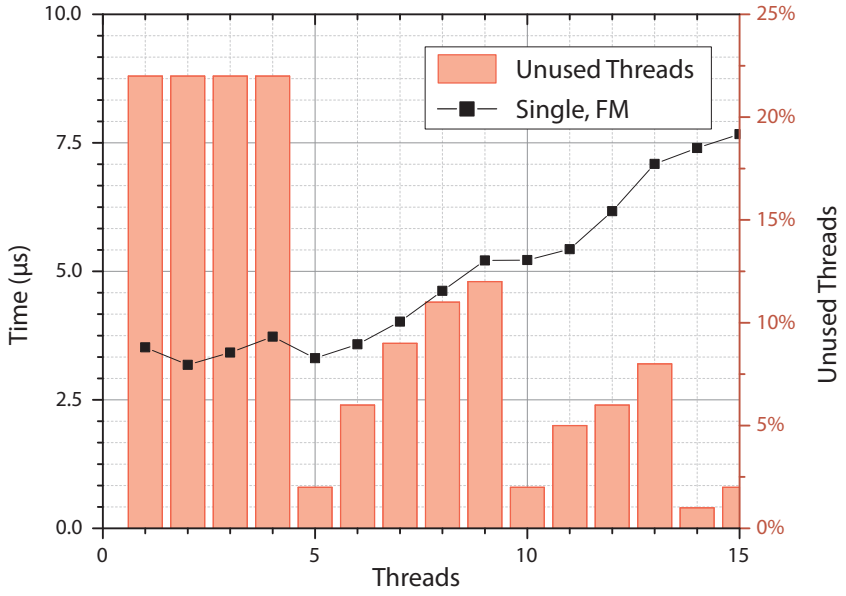
χ^2 Fitting Approach

As mentioned at the beginning of this chapter, there are two ways of fitting a function to data: the log-likelihood and the χ^2 approach. The original DSP code included both ways of performing the fit, although currently the log-likelihood method is used.

Minimizing the χ^2 value of the histogram fitting problem yields a fit time of 1.84 µs/pixel and a mean $\chi^2_{\text{red}} = 0.4663 \pm 0.0043$. This method is not only faster by about 20%, but also more accurate in determining the threshold and noise values.



(a) Thread utilization for a 3×3 search mask together with fit times.



(b) Thread utilization for a 5×5 search mask together with fit times.

Figure 5.10: Thread utilization for 3×3 and 5×5 search mask and corresponding fit times.

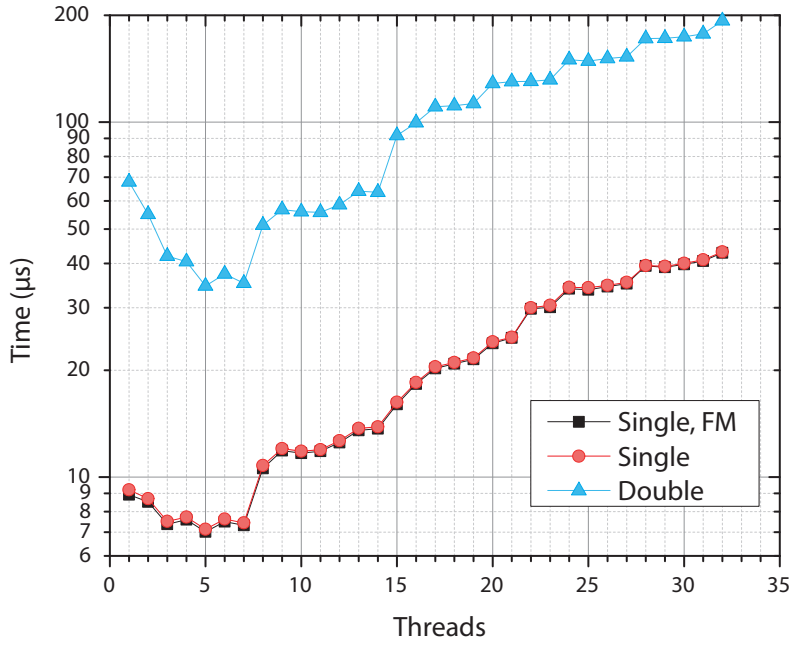


Figure 5.11: Fit times on System 2 for single and double precision fits using a 3×3 search mask depending on the number of threads per search mask cell. The y-axis is scaled logarithmically.

Numerical Considerations

The fitting algorithm iteratively tries to find the best (μ, σ) value pair and terminates if either the convergence criterion is met or the number of maximum allowed iterations $iterations_{max}$ is reached. By reducing this value, one can improve the fit time with the downside of degrading the fit quality. Figure 5.12 shows the distribution of iterations for the fits to terminate with the GPU implementation.

In order to get a feel for how changing the maximum number of iterations affects the results we processed the dataset with different values for $iterations_{max}$ — the resulting numbers for fit time and mean χ^2_{red} can be seen in Figure 5.13. The quality of the fit improves with higher values, while fit time performance decreases. Tuning the standard value of 200 should therefore only be done after some more careful consideration, in order not to degrade fit quality.

Size of the Problem

Histogram data is copied in larger chunks (order of 10,000 pixels) to the main memory of the graphics card to reduce overhead produced by the transfer operation. Currently the values for the histogram entries and interval locations are copied for each pixel,

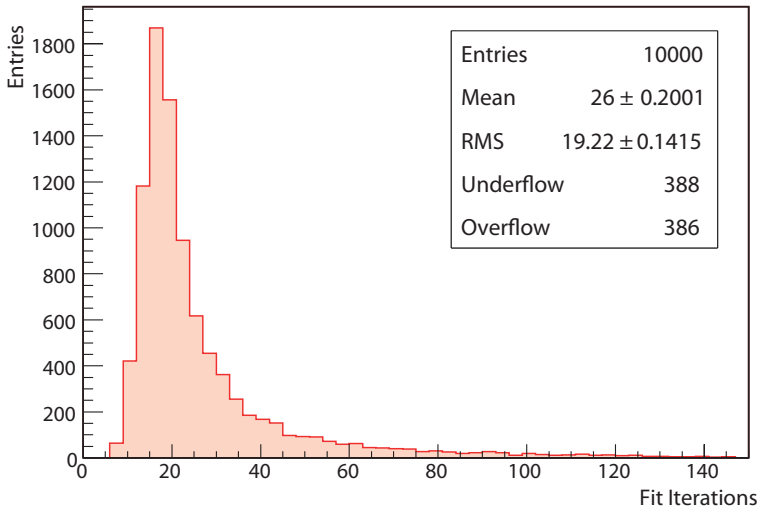


Figure 5.12: Distribution of fit iterations that were needed for the fit to converge. The underflow entries are pixel histograms that were not handled by the normal fitting procedure (not enough valid bins).

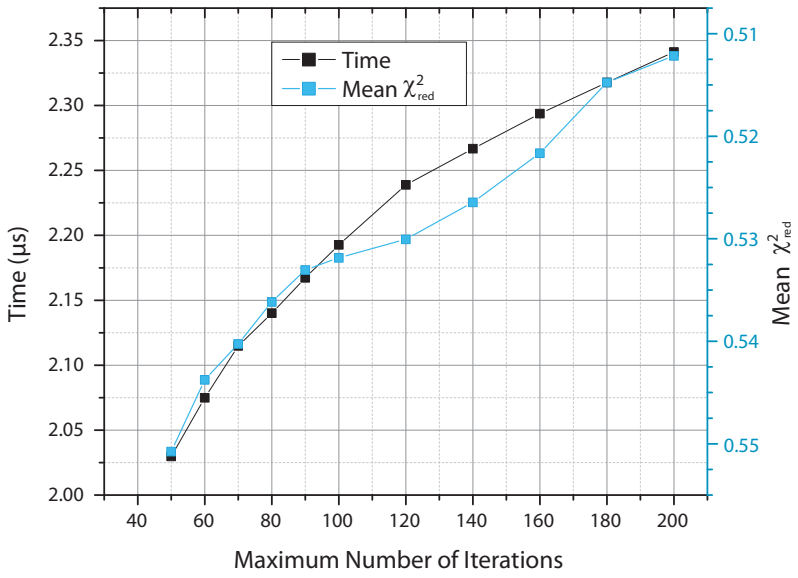


Figure 5.13: Fit times and mean values of χ^2_{red} depending on the maximum number of allowed iterations, before the algorithm terminates. Second y-axis is scaled reversely.

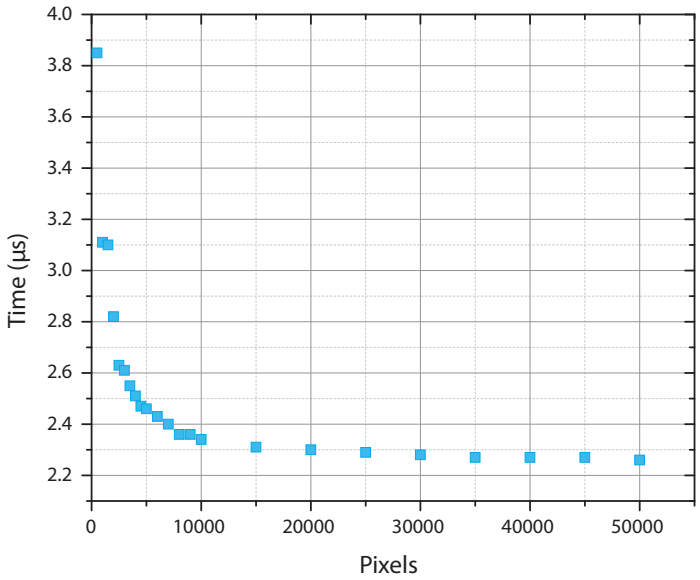


Figure 5.14: The effective fit time per pixel depending on the size of the problem set for a CUDA kernel invocation and memory transfers to/from the device.

resulting in $100 \times 4\,\text{Bytes/float} \times 2\,\text{floats} = 800\,\text{Bytes}$ for a histogram with 100 bins and single floating point precision. 1 GB of GDDR memory can hold the histogram data of about 1 million pixels — memory is therefore not a constraining issue for the implementation.

In order to estimate an efficient size of the problem set to minimize overhead imposed by memory transfers and call of the CUDA kernel, the performance depending on the problem set size was measured. Results for this parameter sweep can be seen in Figure 5.14 for problem sizes ranging from 500 to 50,000 pixels.

We see that for problem sizes that are greater than 10,000 pixels the performance increase becomes negligible. Problem chunks of 10,000 pixels or greater also seem to be a reasonable size for an implementation for the IBL calibration framework, leaving enough room for parallel histogram data transfer and processing.

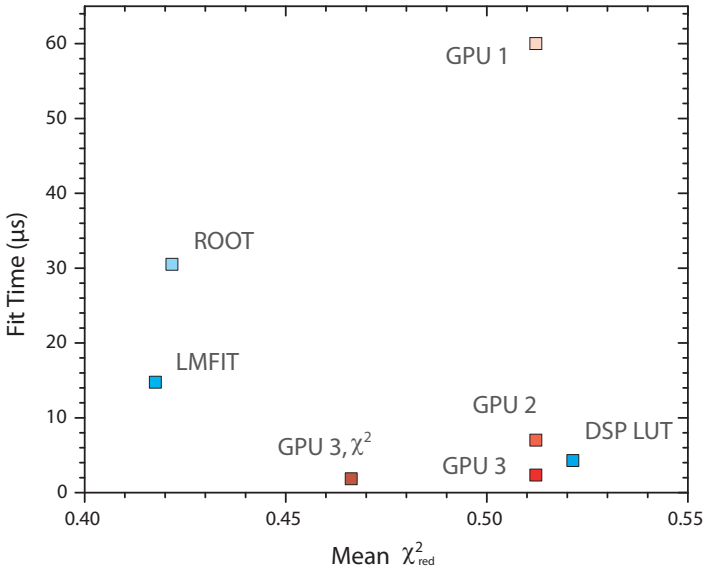


Figure 5.15: Overview of the major performance results for different fitting approaches.

5.5 Discussion

In order to summarize the performance results, fit times and the corresponding mean χ^2_{red} were plotted in Figure 5.15 for the major measurements. Fit times of the CPU approaches were divided by 4 to account for the assumed performance increase resulting from multiple threads on an architecture with four cores.

The two software frameworks ROOT and LMFIT deliver the best fit results, although being almost an order of magnitude slower than the fastest GPU variants. The DSP algorithm on CPU used in conjunction with lookup tables performs similarly as the GPUs on System 2 and 3. The χ^2 fitting approach on System 3 yielded the fastest fit times per pixel.

The new GPU implementation of the DSP fitting algorithm runs 10 times as fast as the old implementation (70 µs/pixel vs. 7.01 µs/pixel) on the GTX280. Using a GPU of the recent Fermi architecture even resulted in another speedup, reaching fit times of 2.34 µs/pixel.

In the context of IBL, this leads to a fit time of 28.1 s for all 12 million pixels on a *single* GeForce GTX480 GPU. Scaling the system to decrease fit times can be achieved by simply adding more graphics cards and splitting up the fit problems. A good problem size is 10,000 pixels or greater, to minimize overhead. On the current ROD the four SDSPs can perform a fit with a rate of 40 µ/pixel, yielding a fit time of 24.4 s per ROD (under the assumption of an even distribution of pixel detector modules to the RODs).

We found that the choice of floating point accuracy as well as the fastmath compiler option did not have a significant impact on the quality of the fit for the GPU algorithm and therefore used single precision floating point arithmetics together with the FM option for benchmarking. With the advent of the Kepler architecture in 2012, up to 1536 cores are available per GPU and it will be interesting to determine how this increase affects the fitting performance.

Again, one has to make a tradeoff decision for the IBL calibration architecture: Either use a software fitting framework like ROOT or LMFIT with a slower runtime, but better fitting results and greater flexibility, or run the DSP algorithm on GPU(s), achieving the best fit times. Optionally, one could further investigate running more complex fitting algorithms on GPU.

6 Conclusion & Outlook

"Careful. We don't want to learn from this." [1]

This thesis has contributed to the IBL calibration architecture in three different areas: A scalable software and hardware architecture for processing calibration histograms was presented and newly introduced hardware and software interfaces were pointed out. In the second part, the performance of embedded software-based TCP/IP networking was reviewed for its use on the IBL ROD. Last, several approaches of fitting threshold scan histograms were evaluated for their deployment on the processing farm.

We showed that an embedded system centered around the MicroBlaze soft-CPU on a Spartan-6 FPGA in conjunction with the LwIP standalone TCP/IP stack is able to deliver the necessary throughput rates for the IBL histogram data transfer with a large safety margin. Compared to the reference implementation by Xilinx a performance improvement between 30% and 40% was observable by tuning the parameters of the networking stack. Several starting points for further optimization of the throughput with the LwIP stack were presented. As TCP is able to perform at these rates, the design of the application layer protocol will prove to be simpler, as reliability features will not have to be considered there. An embedded Linux system running on the MicroBlaze could not meet the performance requirements by producing throughput rates well below 10 MBit/s.

By optimizing the DSP fit algorithm for execution on a current GPU architecture, we were able to prove the suitability of a GPU based approach to the presented fitting problem. With respect to a previous implementation of the algorithm for GPUs, a ten-fold performance increase of the fit times has been made. In addition to the original algorithm from the ROD SDSPs, the two frameworks ROOT and LMFIT were benchmarked for their fit performance. It might be rewarding to investigate the numerical behavior of the current DSP fitting algorithm with regards to its convergence behavior and step size adjustment to even further improve the achievable fit times.

Once a choice regarding the network protocol for transferring data from RODs to the fit farm is made, one can continue to define and implement the capabilities of the application layer protocol. The computing farm can then explicitly be scaled according to

the performance requirements and the implementation of the FitManager and FitServer applications and their integration into IBLDAQ can be started.

Generally, the histogramming procedure of the IBL involves many work areas (ROD FPGA and MDSP firmware development, IBLDAQ libraries and software components like CalibrationConsole, the ATLAS TDAQ computer network, hardware and software of the computing farm) and changes to the architecture and procedures might affect many parts of the system. Therefore a careful definition of stable interfaces and tasks in combination with a continuous information exchange between the involved groups and exhaustive documentation is (as so often) key to a successful implementation.

A Derivation S-Curve

We define the step function for a given threshold μ

$$f(Q) = H\left(Q - \mu\right) \tag{A.1}$$

and a Gaussian noise distribution

$$g(Q) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma_{\text{el}}} \exp\left(-\frac{Q^2}{2\,\sigma_{\text{el}}^2}\right) \tag{A.2}$$

with standard deviation σ_{el}. The probability of obtaining a hit p_{hit} for a given charge Q is then given by the convolution of those two functions:

$$\begin{aligned}
p_{\text{hit}}(Q) &= f(Q) * g(Q) && \text{(A.3)}\\
&= \int_{-\infty}^{\infty} f(Q-u)\; g(u)\; \mathrm{d}u && \text{(A.4)}\\
&= \int_{-\infty}^{\infty} H\left(Q-u-\mu\right)\; \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma_{\text{el}}}\; \exp\left(-\frac{u^2}{2\,\sigma_{\text{el}}^2}\right) \mathrm{d}u && \text{(A.5)}\\
&= \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma_{\text{el}}} \int_{-\infty}^{Q-\mu} \exp\left(-\frac{u^2}{2\,\sigma_{\text{el}}^2}\right) \mathrm{d}u && \text{(A.6)}\\
&= \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma_{\text{el}}} \left[\frac{\sqrt{\pi}\,\sigma_{\text{el}}}{\sqrt{2}}\; \text{Erf}\left(\frac{u}{\sqrt{2}\,\sigma_{\text{el}}}\right)\right]_{-\infty}^{Q-\mu} && \text{(A.7)}\\
&= \frac{1}{2}\left(1 + \text{Erf}\left(\frac{Q-\mu}{\sqrt{2}\,\sigma_{\text{el}}}\right)\right) && \text{(A.8)}
\end{aligned}$$

To obtain Eq. (A.7) it was used that

$$\int \exp\left(-cx^2\right) \mathrm{d}x = \sqrt{\frac{\pi}{4c}}\, \text{Erf}\left(\sqrt{c}x\right). \tag{A.9}$$

List of Acronyms

ADC	Analog-to-Digital Converter
ALICE	A Large Ion Collider Experiment
ASIC	Application Specific Integrated Circuit
BCM	Beam Conditions Monitor
BOC	Back of Crate Card
CERN	European Organization for Nuclear Research
CMS	Compact Muon Solenoid
CORBA	Common Object Request Broker Architecture
CSC	Cathode Strip Chamber
CTP	Central Trigger Processor
CUDA	Compute Unified Device Architecture
DAC	Digital-to-Analog Converter
DAQ	Data Acquisition
DBM	Diamond Beam Monitor
DCS	Detector Control System
DLP	Data-Level Parallelism
DSP	Digital Signal Processor
EF	Event Filter
ENC	Equivalent Noise Charge
FLOP	Floating-Point Operation
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GP-GPU	General Purpose Computing on GPUs
GPU	Graphics Processing Unit

HLT High-Level Trigger

ID Inner Detector

ILP Instruction-Level Parallelism

IP Internet Protocol

IPC Inter-Process Communication

ISO International Organization for Standardization

LEP Large Electron-Positron Collider

LHC Large Hadron Collider

LHCb Large Hadron Collider beauty

LwIP Lightweight IP

LUT Lookup Table

MCC Module Controller Chip

MDSP Master DSP

MDT Monitored Drift Tube

MIP Minimum Ionizing Particle

MSS Maximum Segment Size

MTU Maximum Transmission Unit

MSSM Minimal Supersymmetric Standard Model

NIEL Non Ionizing Energy Loss

OpenCL Open Computing Language

OpenMP Open Multi-Processing

PM Process Manager

PTX Parallel Thread Execution

OSI Open Systems Interconnection

QGP Quark-Gluon Plasma

RISC Reduced Instruction Set Computer

ROB Read-Out Buffer

ROBIN Read-Out Buffer Input

ROD Read-Out Driver Card

RoI	Region of Interest
ROS	Read-Out System
RPC	Resistive Plate Chamber
SBC	Single Board Computer
SCT	Silicon Tracker
SDSP	Slave DSP
SEU	Single Event Upset
sLHC	super-LHC
SM	Standard Model (of Particle Physics)
TCP	Transmission Control Protocol
TDAQ	Trigger and Data Acquisition
TDR	Technical Design Report
TGC	Thin Gap Chamber
TID	Total Ionizing Dose
ToT	Time over Threshold
TRT	Transition Radiation Tracker
UDP	User Datagram Protocol
VME	Versa Module Eurocard

Bibliography

[1] B. Watterson. *The Complete Calvin And Hobbes*. Andrews McMeel Pub., 2005. ISBN 9780740748479.

[2] G. Aad et al. *Combined Search for the Standard Model Higgs Boson Using up to* $4.9fb^{-1}$ *of pp Collision Data at* $\sqrt{s} = 7$ *TeV with the ATLAS Detector at the LHC*. *Physics Letters B*, 710(1):49 – 66, 2012. ISSN 0370-2693. doi:10.1016/j.physletb.2012.02.044.

[3] M. Capeans, G. Darbo, K. Einsweiller, et al. *ATLAS Insertable B-Layer Technical Design Report*. Technical Report CERN-LHCC-2010-013. ATLAS-TDR-019, CERN, Geneva, September 2010.

[4] *Summary of the Analysis of the 19 September 2008 Incident at the LHC*. Technical report, CERN, Geneva, October 2008.

[5] G. Aad, S. Bentvelsen, G. J. Bobbink, et al. *The ATLAS Experiment at the CERN Large Hadron Collider*. *Journal of Instrumentation*, 3:S08003. 437 p, 2008. Also published by CERN Geneva in 2010.

[6] F. G. Oakham. *ATLAS Upgrade for the HL-LHC*. January 2012.

[7] *Letter of Intent for the Phase-I Upgrade of the ATLAS Experiment*. Technical Report CERN-LHCC-2011-012. LHCC-I-020, CERN, Geneva, November 2011.

[8] C. Gemme. *ATLAS Upgrades Programme*. April 2012.

[9] G. Aad, M. Ackers, F. A. Alberti, et al. *ATLAS Pixel Detector Electronics and Sensors*. *Journal of Instrumentation*, 3:P07007, 2008.

[10] L. Rossi, P. Fischer, T. Rohe, and N. Wermes. *Pixel Detectors: From Fundamentals to Applications (Particle Acceleration and Detection)*. Springer, 2006. ISBN 9783540283324.

[11] K. Nakamura and P. D. Group. *Review of Particle Physics*. *Journal of Physics G: Nuclear and Particle Physics*, 37(7A):075021, 2010.

[12] $\frac{dE}{dx}$ *Measurement in the ATLAS Pixel Detector and its Use for Particle Identification*. Technical Report ATLAS-CONF-2011-016, CERN, Geneva, March 2011.

[13] G. Lindstroem et al. *Developments for Radiation Hard Silicon Detectors by Defect Engineering: Results by the CERN RD48 (ROSE) Collaboration*. 2000.

[14] J. Weingarten. *System Test and Noise Performance Studies at the ATLAS Pixel Detector*. Ph.D. thesis, Bonn University, 2007.

[15] J. Grosse-Knetter. *Vertex Measurement at a Hadron Collider - The ATLAS Pixel Detector*, March 2008. Bonn University.

[16] N. Wermes and G. Hallewel. *ATLAS Pixel Detector*. Technical Design Report ATLAS. CERN, Geneva, 1998.

[17] T. Ince. *Operational Experience with the ATLAS Pixel Detector*. Technical Report ATL-INDET-PROC-2012-001, CERN, Geneva, January 2012.

[18] *Pixel Detector Calibration Manual*. `https://atlasop.cern.ch/twiki/bin/view/Main/PixelDetectorCalibrationManual`.

[19] C. Gallrapp. *Overview of the ATLAS Insertable B-Layer (IBL) Project*. February 2012.

[20] M. Barbero. *The FE-I4 Pixel Readout Chip and the IBL Module. Proceedings of Science*, (ATL-UPGRADE-PROC-2012-001), January 2012.

[21] M. Barbero, D. Arutinov, T. Hemperek, et al. *FE-I4 ATLAS Pixel Chip Design. PoS*, VERTEX2009:027. 10 p, November 2010.

[22] M. Garcia-Sciveres, D. Arutinov, M. Barbero, et al. *The FE-I4 Pixel Readout Integrated Circuit*. Technical Report ATL-UPGRADE-PROC-2010-001, CERN, Geneva, January 2010.

[23] G. Bruni, M. Bruschi, I. D'Antone, et al. *ATLAS IBL: Integration of new HW/SW Readout Features for the Additional Layer of Pixels*. (ATL-INDET-PROC-2010-032), October 2010.

[24] J. Dopke, D. Falchieri, T. Flick, et al. *The IBL Readout System*. (ATL-INDET-PROC-2010-031), October 2010.

[25] D. Falchieri, G. Bruni, M. Bruschi, et al. *Proposal for a Readout Driver Card for the ATLAS Insertable B-Layer*. (ATL-INDET-PROC-2010-039), November 2010.

[26] N. Schroer, G. Bruni, M. Bruschi, et al. *The ATLAS IBL BOC Prototype*. (ATL-INDET-PROC-2011-029), October 2011.

[27] P. Jenni, M. Nessi, M. Nordberg, and K. Smith. *ATLAS High-Level Trigger, Data-Acquisition and Controls: Technical Design Report*. Technical Design Report ATLAS. CERN, Geneva, 2003.

[28] *ATLAS Trigger Performance: Status Report*. Technical Report CERN-LHCC-98-015, CERN, Geneva, June 1998.

[29] R. Cranfield, G. Crone, D. Francis, et al. *The ATLAS ROBIN*. *Journal of Instrumentation*, 3:T01002, 2008.

[30] G. Avolio, M. Dobson, G. L. Miotto, and M. Wiesmann. *The Process Manager in the ATLAS DAQ System.*

[31] S. Kolos, D. Burckhart-Chromek, J. Flammer, et al. *Experience with CORBA Communication Middleware in the ATLAS DAQ.*

[32] *ATLAS Computing: Technical Design Report.* Technical report, Geneva, 2005.

[33] J. Dopke, D. Falchieri, T. Flick, et al. *Study of FPGA and GPU Based Pixel Calibration for ATLAS IBL.* May 2010.

[34] IBL WG4. *Notes on Calibration of the ATLAS IBL Detector Using the New VME Readout Architecture*, July 2011. Unpublished.

[35] A. Gabrielli, G. Bruni, M. Bruschi, et al. *ATLAS IBL: Integration of new HW/SW Readout Features for the Additional Layer of Pixels.* September 2010.

[36] I. O. F. Standardization. *ISO/IEC 7498-1:1994 Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model. International Standard ISOIEC 74981*, p. 59, 1996.

[37] R. Braden. *Requirements for Internet Hosts - Communication Layers.* RFC 1122 (Standard), September 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298.

[38] A. S. Tanenbaum and D. J. Wetherall. *Computer Networks (5th Edition).* Prentice Hall, 2010. ISBN 0132126958.

[39] J. Postel. *User Datagram Protocol.* RFC 768 (Standard), August 1980.

[40] J. Postel. *Transmission Control Protocol.* RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[41] Xilinx. *MicroBlaze Processor Reference Guide*, 2008. UG081.

[42] E. Hallett. *Reference System: XPS LL Tri-Mode Ethernet MAC Embedded Systems for MicroBlaze and PowerPC Processors.* Xilinx, September 2008. XAPP1041.

[43] Xilinx. *LightWeight IP (lwIP) Application Examples*, April 2011. XAPP1026.

[44] M. Simek. *Ethernet Performance with Netperf.* `http://www.monstr.eu/wiki/doku.php?id=kernel:testing:net`, February 2010.

[45] *FPGANotes Wiki - TCP/IP Stacks.* `http://wiki.fpganotes.com/doku.php/edk:library:tcpip`, May 2011.

[46] `http://savannah.nongnu.org/projects/lwip/`.

[47] Xilinx. *AXI Reference Guide*, March 2011. UG761.

[48] Xilinx. *LogiCORE IP XPS LL TEMAC (v2.03a)*, December 2010. DS537.

[49] Xilinx. *LogiCORE IP AXI Ethernet (v2.01a)*, March 2011. DS759.

[50] Xilinx. *Command Line Tools User Guide*, March 2011. UG628.

[51] L. Lyons. *Statistics for Nuclear and Particle Physicists*. Cambridge University Press, 1989. ISBN 0521379342.

[52] D. W. Marquardt. *An Algorithm for Least-Squares Estimation of Nonlinear Parameters. Journal of the Society for Industrial and Applied Mathematics*, 11:431–441, 1963.

[53] K. Madsen, H. Nielsen, and O. Tingleff. *Methods for Non-Linear Least Squares Problems*. 2004.

[54] P. Bevington and D. K. Robinson. *Data Reduction and Error Analysis for the Physical Sciences*. McGraw-Hill Science/Engineering/Math, 2002. ISBN 0072472278.

[55] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery0. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002. ISBN 0521750334.

[56] *Top 500 Supercomputing Sites*. `http://www.top500.org`.

[57] D. B. Kirk and W. W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*. Morgan Kaufmann, 2010. ISBN 0123814723.

[58] NVidia. *NVIDIA CUDA C Programming Guide*, May 2011.

[59] NVidia. *CUDA C Best Practices Guide*, May 2011.

[60] NVidia. *CUDA API Reference Manual*, February 2011.

[61] NVidia. *Whitepaper - NVIDIA's Next Generation CUDA Compute Architecture: Fermi*, 2009.

[62] NVidia. *Whitepaper - NVIDIA GeForce GTX 680*, 2012.

[63] R. Brun and F. Rademakers. *ROOT - An Object Oriented Data Analysis Framework*. In *AIHENP'96 Workshop, Lausane*, volume 389, pp. 81–86. 1996.

[64] J. Wuttke. *lmfit - a C/C++ Routine for Levenberg-Marquardt Minimization with Wrapper for Least-Squares Curve Fitting, based on Work by B. S. Garbow, K. E. Hillstrom, J. J. Moré, and S. Moshier. Version 3.2, retrieved December 2011 from* `http://www.messen-und-deuten.de/lmfit/`.

[65] `http://ark.intel.com/products/29762/Intel-Core2-Duo-Processor-T7700-%284M-Cache-2_40-GHz-800-MHz-FSB%29`.

[66] `http://ark.intel.com/products/29765/Intel-Core2-Quad-Processor-Q6600-%288M-Cache-2_40-GHz-1066-MHz-FSB%29`.

[67] `http://ark.intel.com/products/37147/Intel-Core-i7-920-Processor-%288M-Cache-2_66-GHz-4_80-GTs-Intel-QPI%29`.

[68] http://en.wikipedia.org/wiki/Nvidia_Quadro#Quadro_FX_M.

[69] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-280/specifications.

[70] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications.

[71] *Intel Compilers.* http://software.intel.com/en-us/articles/intel-compilers/.

[72] *The GNU Compiler Collection.* http://gcc.gnu.org/.

[73] *MINUIT — Function Minimization and Error Analysis.* http://wwwasdoc.web.cern.ch/wwwasdoc/minuit/.

[74] N. Whitehead and A. Fit-Florea. *Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs*, 2011.

[75] J. Dopke. *Studies for Automated Tuning of the Quality of the Back-of-Crate Card for the ATLAS Pixel Detector.* Master's thesis, University of Wuppertal, 2007.

Acknowledgements

"I propose we leave math to the machines
and go play outside." [1]

First of all, I want to thank *Prof. Dr. Reinhard Männer* and *Dr. Andreas Kugel* for giving me the opportunity to carry out my diploma thesis in their research group. I would also like to kindly thank *Prof. Dr. Peter Fischer* for acting as a second referee of this thesis.

I would especially like to thank my advisor *Dr. Andreas Kugel* for not only giving me the opportunity to attend the International School on Trigger and Data Acquisition and to visit CERN, but also for excellent help with my work and steering me in the right direction at the right time.

For proofreading this thesis, but also for helpful suggestions, I would further like to thank *Michael Engelhart*, *Michael Krieger*, *Katharina Lohnert*, *Niko Roßkopf*, and *Lukas Sauermann*.

Alexander Lenhart has been a great help to me with his advice on typography and graphics, as has *Michael Engelhart* for sorting out issues with LaTeX.

I would like to thank *Jens Dopke* not only for showing me around CERN, but also for his very helpful insights on the Pixel DSP code ("Talking or writing to the right people is faster than any [search] engine you can find out there." [75, p. 57]).

My office mates *Thomas Gerlach* and *Nicolai Schroer* have contributed a great deal to a relaxed and inspiring working atmosphere and also helped me out many times, thank you!

Last, I am grateful for my parents for always supporting me, however crazy my plans are.

Erklärung zur selbständigen Verfassung

Ich versichere, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 1. Juni 2012.

Moritz Kretz