

https://dspace.org/

On the use of heterogenous computing in high-energy particle physics at the ATLAS detector

Sacks, Marc

2017

https://hdl.handle.net/10539/24981

Sacks, Marc (2017) On the use of heterogenous computing in high-energy particle physics at the ATLAS detector, University of the Witwatersrand, Johannesburg, <http://hdl.handle.net/10539/24981>

Downloaded from a DSpace repository. https://dspace.org/

UNIVERSITY OF THE WITWATERSRAND

On the Use of Heterogenous Computing in High-Energy Particle Physics at the ATLAS Detector

Author:
Marc SACKS

Supervisor:
Professor Bruce MELLADO

A dissertation submitted in fulfillment of the requirements for the degree of Master of Physics

in the

School of Physics

November 1, 2017

Declaration of Authorship

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signed: 

Date: 1/11/2017

UNIVERSITY OF THE WITWATERSRAND

Abstract

Faculty of Science
School of Physics

Master of Physics

On the Use of Heterogenous Computing in High-Energy Particle Physics at the ATLAS Detector

by Marc SACKS

The ATLAS detector at the Large Hadron Collider (LHC) at CERN is undergoing upgrades to its instrumentation, as well as the hardware and software that comprise its Trigger and Data Acquisition (TDAQ) system. These upgrades are necessitated largely by the planned increase in running luminosity at the LHC from 10^{33} $\text{cm}^{-2}\text{s}^{-1}$ to 10^{35} $\text{cm}^{-2}\text{s}^{-1}$ in around 2022. The increased energy will yield larger cross sections for interesting physics processes, but will also lead to increased artifacts in on-line reconstruction in the trigger, as well as increased trigger rates, beyond the current system's capabilities. To meet these demands it is likely that the massive parallelism of General-Purpose Programming with Graphic Processing Units (GPGPU) will be utilised. This dissertation addresses the problem of integrating GPGPU into the existing Trigger and TDAQ platforms; detailing and analysing GPGPU performance in the context of performing in a high-throughput, on-line environment like ATLAS. Preliminary tests show low to moderate speed-up with GPU relative to CPU, indicating that to achieve a more significant performance increase it may be necessary to alter the current platform beyond pairing suitable GPUs to CPUs in an optimum ratio. Possible solutions are proposed and recommendations for future work are given.

Acknowledgements

I would like to acknowledge first of all my parents for their support, both moral and financial. You value education highly and have instilled this value in your children. Secondly to my supervisor Professor Mellado, thank you for your guidance and for introducing me to the world of high-energy particle physics. Thanks is also due to the HEPP team at Wits for their help and encouragement throughout my MSc and to the NRF for their generous support.

Contents

Declaration of Authorship ii

Abstract iii

Acknowledgements iv

List of Abbreviations viii

1 Introduction 1
- 1.1 Big Data at the ATLAS Detector at the LHC . . . 1
  - 1.1.1 GPGPU in ATLAS . . . 1
- 1.2 Motivation for Research . . . 2
- 1.3 Problem Statement and Layout of Dissertation . . . 3
  - 1.3.1 Problem Statement . . . 3
  - 1.3.2 Layout of Dissertation . . . 4

2 The ATLAS Experiment at the LHC 5
- 2.1 The Large Hadron Collider . . . 5
- 2.2 The ATLAS Detector . . . 6
  - 2.2.1 Inner Detector . . . 6
  - 2.2.2 Calorimeters . . . 7
  - 2.2.3 Muon Spectrometer . . . 7
- 2.3 ATLAS Trigger and DAQ System . . . 8
  - 2.3.1 Trigger Operation . . . 8
    - The Low-Level Trigger . . . 9
    - The High-Level Trigger . . . 10

3 GPU 12
- 3.1 GPGPU . . . 12
  - 3.1.1 The Evolution of the GPU . . . 12
  - 3.1.2 Many-core Processors . . . 14
  - 3.1.3 What Computations Benefit from GPU Implementation? 15
  - 3.1.4 Performance Limits . . . 17
  - 3.1.5 The CUDA Programming Platform . . . 17
    - CUDA Kernel . . . 18

Thread Hierarchy . . . 18
Nvidia GPU Memory Structure . . . 19

4 GPU Characterisations, Benchmarks, and Comparisons 22
4.1 Introduction . . . 22
4.2 Hardware . . . 22
4.3 Methods . . . 23
4.4 Results and Analyses . . . 23
4.4.1 Sum Reduction . . . 23
4.4.2 Matrix Multiplication Benchmarks . . . 25
4.4.3 Memory Bandwidth Benchmarks . . . 26
4.5 Selecting GPUs for GPGPU Application . . . 27

5 GPGPU in ATLAS Level-1 Trigger 29
5.1 ATLAS Level-1 Trigger . . . 29
5.1.1 FPGA Use in Trigger Front-end . . . 29
5.2 GPGPU in the Level-1 Trigger . . . 30
5.2.1 Limitations of GPGPU in Online Environment . . . 30
5.2.2 Heterogeneous Co-Processing Unit . . . 31
A GPU-ARM based PU . . . 31
5.3 Potential Functionality for GPU to Execute . . . 32
5.3.1 Processing of High Quality-Factor Events . . . 32
5.3.2 Facilitating Less Dead-time . . . 34
5.4 Conclusion . . . 35

6 Heterogenous Computing Platforms in the ATLAS HLT 36
6.1 Introduction . . . 36
6.1.1 HLT Upgrade . . . 36
6.2 GPU Demonstrator . . . 37
6.2.1 Accelerator Process Extension . . . 37
6.3 Algorithms Tested by GPU Demonstrator . . . 38
6.3.1 Inner Detector Tracking . . . 39
6.3.2 Calorimeter Clustering . . . 40
6.4 Analysis of GPU Demonstrator Results . . . 41
6.4.1 ID Tracking Results . . . 41
6.4.2 Calorimeter Clustering Results . . . 43
6.4.3 Combined ID Tracking and Calorimeter Clustering . . . 43
6.5 Critical Analysis . . . 44
6.5.1 Necessity of Improved Algorithms . . . 46
6.5.2 Co-PU in HLT . . . 48
Porting APE to ARM Architecture . . . 49

7 Conclusion **50**
7.1 GPU Performance Parameters 50
7.2 GPGPU in the Level-0/1 Trigger 51
7.3 GPGPU in the HLT Trigger 52
7.4 Future Work 53
7.4.1 In the Level-0/1 53
7.4.2 In the HLT 53

References **54**

List of Abbreviations

ATLAS	**A** **T**oroidal **L**HC **A**pparatu**S**
LHC	**L**arge **H**adron **C**ollider
GPGPU	**G**eneral-purpose **P**rogramming with **G**PU
GPU	**G**raphic **P**rocessing **U**nit
HL-LHC	**H**igh **L**uminosity **L**HC
QF	**Q**uality **F**actor
TDAQ	**T**rigger and **D**ata **A**c**Q**uisition
CERN	European Laboratory for Particle Physics
CMS	**C**ompact **M**uon **S**olenoid
LHCb	**L**arge **H**adron **C**ollider **b**eauty
ALICE	**A** **L**arge **I**on **C**ollider **E**xperiment
SM	**S**tandard **M**odel
BSM	**B**eyond the **S**tandard **M**odel
ID	**I**nner **D**etector
SCT	**S**emiconductor **C**entral **T**racker
TRT	**T**ransition **R**adiation **T**racker
PMT	**P**hoto **M**ultiplier **T**ube
FPGA	**F**ield **P**rogrammable **G**ate **Array**
HLT	**H**igh **L**evel **T**rigger
EF	**E**vent **F**ilter
CTP	**C**entral **T**rigger **P**rocessor
sROD	**s**uper **R**ead **O**ut **D**river
ROS	**R**ead **O**ut **S**ystem
RoI	**R**egion **o**f **I**nterest
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
FTK	**F**ast **T**rac**K**er
TilePPr	**T**ile **P**re **P**rocessor
RoIB	**R**egion **o**f **I**nterest **B**uilder
HLTSV	**HLT** **S**uper**V**isor
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SPMD	**S**ingle **P**rogram **M**ultiple **D**ata
FLOPS	**F**loating **P**oint **O**perations **P**er **S**econd
RAM	**R**andom **A**ccess **M**emory
HDL	**H**ardware **D**escription **L**anguage
Co-PU	**C**o **P**rocessing **U**nit

OF	**O**ptimal **F**iltering
APE	**A**ccelerator **P**rocess **E**xtension
EDM	**E**vent **D**ata **M**odel
CA	**C**ellular **A**utomata
IPC	**I**nter **P**rocess **C**ommunication

This dissertation and the energy spent authoring it are in dedication to my eternally beloved friends Eli and Dmitri.

Chapter 1

Introduction

1.1 Big Data at the ATLAS Detector at the LHC

A Large Toroidal LHC Apparatus (ATLAS) is a general-purpose proton-proton collision detector at the Large Hadron Collider (LHC) at CERN in Geneva, Switzerland [1]. Bunches of protons collide in the detector every 25 ns - these collisions are referred to as events. At about 1.7 MB per event, the detector faces data rates of about 70 TB/s. Most of these events represent uninteresting physics processes or background noise, and must be discarded. The decision whether to store or discard the event must be done in real-time for every bunch crossing, i.e. at a rate of 40 MHz. The first stage of this decision process occurs in no more than 2.5 µs. During Run-2 of data taking (2015-2018) the LHC operates at a luminosity of 1×10^{34} $\text{cm}^{-2}\text{s}^{-1}$. At this luminosity approximately only 1 out of every 400 000 events can be accepted for permanent storage so as not to exceed the write speed of the current storage system of about 100 MB/s [2]. ATLAS has achieved this performance with a multi-stage *trigger* system consisting of a custom hardware front-end and commodity server back-end, which reduces the 40 MHz bunch rate to a storage rate of a few 100 Hz.

1.1.1 GPGPU in ATLAS

General-Purpose Programming with Graphics Processing Units (GPGPU) is a Graphics Processing Unit (GPU) based technology which allows users to compute with massive parallelism [3]. In certain contexts this results in lower processing times and greater energy efficiency. GPGPU is a relatively new technology, over the last decade or so it has been used successfully in many fields of science to, for instance, run simulations or solve compute intensive problems which are cumbersome or impossible to solve on CPU processors due to time constraints.

This dissertation concerns the integration of GPGPU into the ATLAS trigger system, for increased processing power at lower costs.

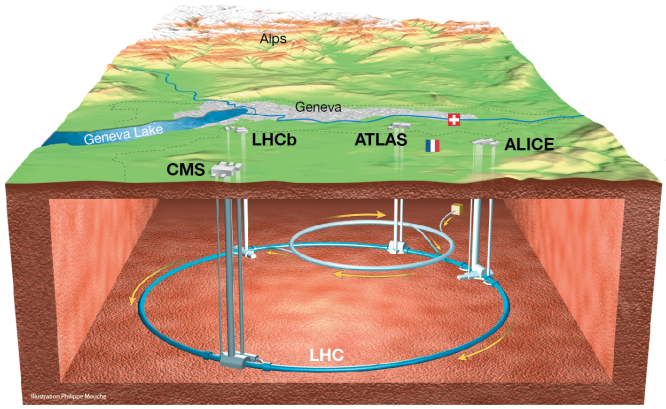
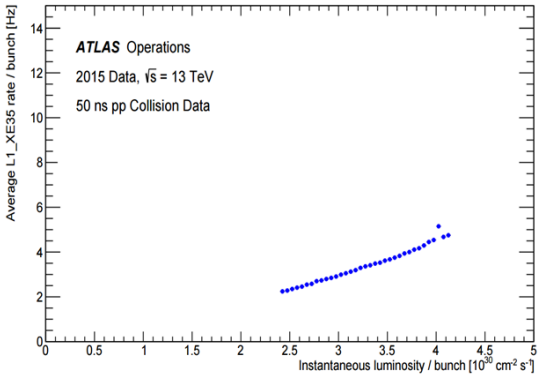


FIGURE 1.1: Illustration of the LHC and its detectors [4].

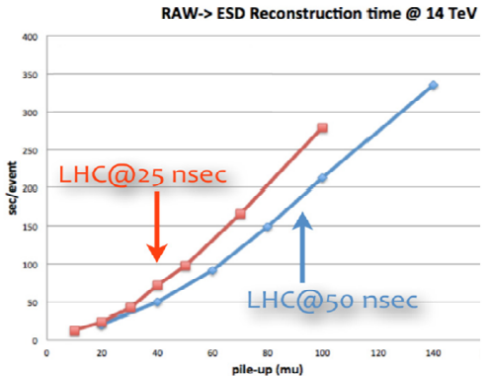
1.2 Motivation for Research

During Run-1 (2009-2013) of data taking at ATLAS, the LHC was operating at a nominal centre-of-mass energy of 8 TeV and a luminosity of 8×10^{33} $\text{cm}^{-2}\text{s}^{-1}$. Run-2 operates at 13 TeV with a luminosity of 1×10^{34} $\text{cm}^{-2}\text{s}^{-1}$. In terms of peak interactions per bunch crossing; in 2011 the maximum was 15, increased to almost 35 in 2012, and a maximum of 80 for Run-2 so far. Circa 2022 the LHC is expected to run at 14 TeV, referred to as the High Luminosity LHC (HL-LHC), with a luminosity on the order of 10^{35} $\text{cm}^{-2}\text{s}^{-1}$. This raised luminosity increases the cross-section of electroweak processes, but also leads to an increase in pile-up (a type of artifact) and an increase in trigger rates. Figure 1.2 illustrates the practical consequences of these increases. Firstly subfigure 1.2a shows how trigger rates increase for increasing luminosity. Subfigure 1.2b illustrates how time-to-process events increases with pile-up. Subfigure 1.2c shows how Quality Factor (QF), a measure of the quality of energy reconstruction, is affected in different pile-up scenarios. Note that quality decreases for increasing QF. A_{in} and A_{out} refer to two types of pile-up, the details of which are unimportant here. What is of note here is the degradation of QF with increased pile-up energy. Finally, subfigure 1.2d illustrates the increase in χ^2 error for amplitude estimation for events with pile-up compared to those without.

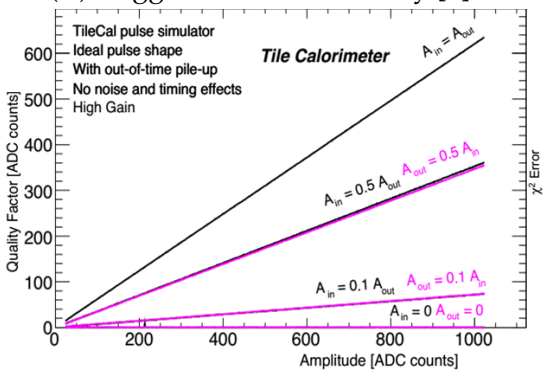
To meet the requirements for physics taking in high luminosity, high pile-up conditions, it is not feasible to scale up the existing trigger and data acquisition (TDAQ) infrastructure as is. More sophisticated algorithms are required and, with or without algorithms of higher complexity, it would be necessary to look towards processors other than CPUs to meet the latency



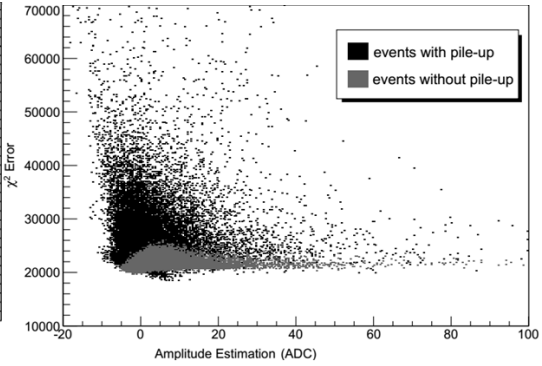
(A) Trigger rate vs luminosity [5].



(B) Processing time vs pile-up [6].



(C) Quality Factor with pile-up [7].



(D) Amplitude estimation in pile-up [8].

FIGURE 1.2: The effects of increased pile-up and trigger rates.

and throughput needs of the HL-LHC TDAQ. It is proposed here that to this end GPGPU technology could be a viable option.

1.3 Problem Statement and Layout of Dissertation

1.3.1 Problem Statement

Given the inverse relationship between quality of physics reconstruction and luminosity, the HL-LHC will require increased computational power to support higher trigger rates and algorithms of increased complexity to more efficiently address increased pile-up. GPU technology presents a low-cost, user-friendly platform with which speed-ups can be achieved through massive parallelism. Their introduction to the TDAQ system would not be trivial. **This dissertation is an analysis of the factors that must be considered for successful integration of GPUs into the TDAQ, namely: GPU platform, GPU:CPU ratio, stage/s of the trigger into which GPU will be**

inserted, and types of reconstruction algorithms suitable for GPU implementation.

1.3.2 Layout of Dissertation

Chapter 2 introduces the LHC and the ATLAS detector, and the trigger and data acquisition systems. Chapter 3 is a discussion of the history and technical characteristics of GPUs of relevance in this dissertation. Chapter 4 presents results from GPU benchmark tests conducted by the author, which will give the reader a better understanding of GPGPU, and allow the reader to follow arguments relating to GPGPU in the ATLAS trigger more readily. Chapter 5 explores the possibility of GPGPU implementation into the ATLAS Level-1 Trigger. Chapter 6 concerns the introduction of GPGPU into the ATLAS High-Level Trigger as well as presents results from the ATLAS GPU demonstrator project, and an analysis thereof. Chapter 7 concludes the dissertation with recommendations for future work.

Chapter 2

The ATLAS Experiment at the LHC

2.1 The Large Hadron Collider

CERN, the European Laboratory for Particle Physics [1], was founded in 1954 [9]. It was responsible for securing the funding for the construction of the LHC, and today is in charge of the operation of the LHC. The LHC is a circular particle accelerator, at 27 km in circumference it is the largest and most powerful accelerator to date, and considered the largest machine ever made [10]. It is located on the Swiss-Franco border, in a cavern approximately 100 m below ground. The LHC began operation in 2008, colliding protons as well as heavy ions. Cross sections of physical processes of interest such as the production of the Standard Model (SM) Higgs Boson are small, to make the study of these processes viable the LHC must therefore operate at high energies [1]. The LHC uses a series of accelerators to get particles up to the energy necessary for injection into the LHC. These are, in order, Linac 2, Proton Synchrotron Booster, Proton Synchrotron, and Super Proton Synchrotron. Protons are injected into the LHC beam pipes in up to 3546 bunches 20 µm in diameter and 2 cm in length. Bunches circulate in two separate beam pipes, in opposite directions. The direction of travel of these particles is controlled by an 8 T magnetic field generated by helium cooled superconducting electromagnets at a temperature of 1.9 K. At four points along the circumference of the LHC, corresponding to four detectors, these particle beams are collided. These detectors are

- ATLAS: A Toroidal LHC Apparatus
- CMS: Compact Muon Solenoid
- LHCb: Large Hadron Collider beauty
- ALICE: A Large Ion Collider Experiment

[1] The acronym is derived from the original French name for CERN; Conseil Européen pour la Recherche Nucléaire.

The first two detectors are considered general purpose, and are used to explore a wide range of physics processes, while LHCb investigates the proportion of matter versus anti-matter and ALICE focuses on quark-gluon plasma. The ATLAS detector is the focal point of this dissertation.

2.2 The ATLAS Detector

The ATLAS detector explores a wide range of physics, from refining SM physics measurements (QCD, electroweak interaction, flavour physics) to studying possible Beyond the Standard Model (BSM) physics (supersymmetry, extra dimensions). ATLAS, along with CMS, made history in 2012 after confirming the existence of the SM Higgs Boson. Discoveries of further novel particles continue to be explored. Figure 2.1 illustrates the dimensions of the detector, the tallest of the detectors. Weighing 7000 tons, ATLAS itself comprises several sub-detectors to measure the energy and trajectories of the particles resulting from the proton-proton collisions occurring in its centre. These sub-detectors are

- The Inner Detector (ID)
- The electromagnetic and hadronic calorimeters
- The Muon Spectrometer

What follows is a short description of the instrumentation of each sub-dectector, a complete description can be found in [1].

2.2.1 Inner Detector

The ID itself comprises three sub-detectors subjected to a 2 T magnetic field: the Pixel detector, Semiconductor Central Tracker (SCT), and the Transition Radiation Tracker (TRT). The ID measures charged particles in $|\eta| < 2.6$. The pixels of the tracker are 50x400 μm^2, and its read-out channels are approximately 80.4 million in number. The pixel tracker has a precision of 115 µm in the direction of the beam axis, and 10 µm perpendicular to the beam axis. The SCT has about 6.3 million read-out channels, an accuracy of 17 µm along the beam axis and 580 µm perpendicular to the axis. The TRT has approximately 351000 read-out channels, and only detects particles orthogonal to the beam axis, at an accuracy of 130 µm [12].

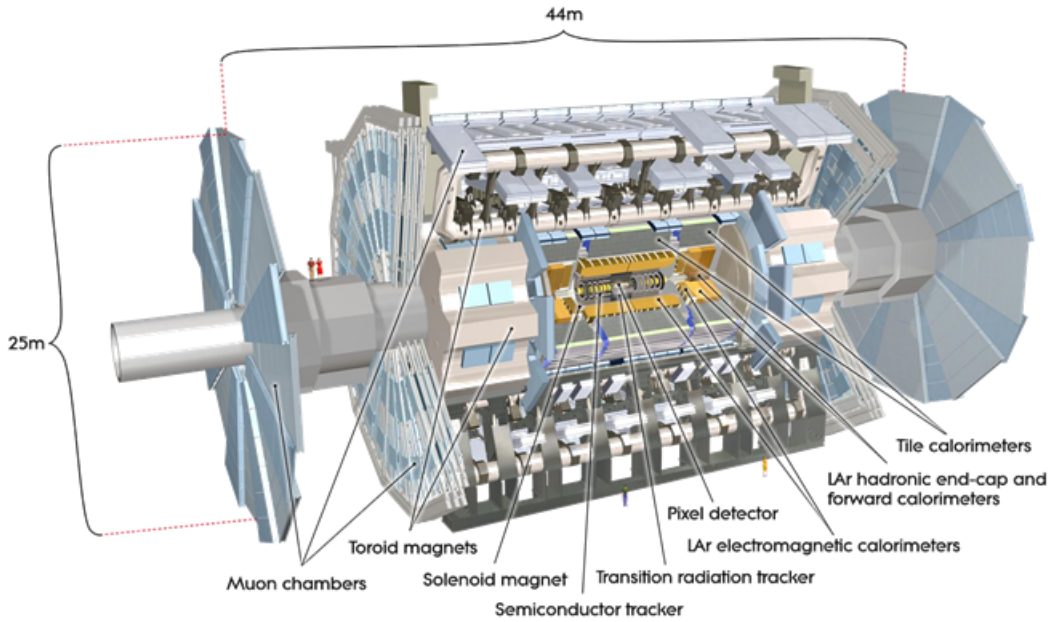


FIGURE 2.1: Illustration of the ATLAS detector [11].

2.2.2 Calorimeters

The electromagnetic and hadronic calorimeters cover the region $|\eta| < 4.9$, measuring electrons and photons. These measurements are used in jet reconstruction and missing transverse energy measurements. The electromagnetic calorimeter is a Lead-LAr detector. The hadronic calorimeter uses tiles of scintillating plastic sandwiched between plates of steel designed to dampen particle energy. The tiles of the hadronic calorimeter have lent it the moniker Tile Calorimeter, or TileCal. Wavelength shifting fibres sitting on two edges of each tile transmit energy to photomultiplier tubes (PMTs) - of which there are also two per tile [13].

2.2.3 Muon Spectrometer

The muon spectrometer uses air-core toroidal electromagnets to deflect muon trajectories in the large barrel toroids in the region $|\eta| < 1.4$, and smaller end-cap toroids at both ends in the region $1.6 < \eta < 2.7$. Muon chambers sit cylindrically around the beam axis in the barrel region and orthogonal to the beam axis in the end caps [14].

2.3 ATLAS Trigger and DAQ System

All of ATLAS' subdetectors interface with a network of electronics and software which, in real-time, integrates their measurements to develop a coherent picture of a collision, referred to as an event. Within seconds of the event, the system decides whether to discard or permanently retain the measurement data of the event for later analysis. The system comprises the Trigger, which makes the decision to discard or store, and the Data Acquistion (DAQ) system which routes information to and from subdetectors in aid of the trigger, and to permanent storage [15]. The trigger is multi-stage algorithm, using detector-wide readings at varying levels of detail (low, medium, or full 'granularity'). It is tasked with analysing the billion collisions per second and, based on their time-energy profiles, deciding whether it is likely that a given event could constitute interesting physics. There is a maximum rate of event acceptance which ultimately reflects the technical limitations of data transfer and storage protocol rates currently available. The compromise between useful data capturing and proper resource management is that of wanting to permanently store as many possible interesting events for further analysis, and wanting to store as few events as possible for logistical reasons.

2.3.1 Trigger Operation

The entrance point of the trigger is the low-level hardware-based portion. The crucial hardware element at this level is the Field Programmable Gate Array (FPGA), a type of customisable processor capable of low latency, parallel calculations. The low-level trigger accepts or rejects an event with a latency of about 2 µs. The trigger has evolved from Run-1 of data taking at ATLAS to Run-2 due to increasing trigger rates. Run-1 saw a collision frequency of 20 MHz, Run-2 is double this. In both runs the latency requirements remained, implying increased computational requirements in Run-2. The TDAQ passes events accepted by the low-level trigger to the next stage of the trigger at rate no higher than 100 kHz in Run-2. This rate reduction is needed due to the next stage of the trigger being software based, thus operating at far larger latencies of around 500 ms. From Run-1 to 2, the split between low and High Level Trigger (HLT) has altered, going from a three stage (Level-1, Level-2, Event Filter (EF)), to a two-stage configuration where Level-2 and the EF have merged. In both runs the trigger selected events based on the following markers: high-p_T muons, electrons, photons, jets and τ-leptons. Additionally large missing transverse energy, and large

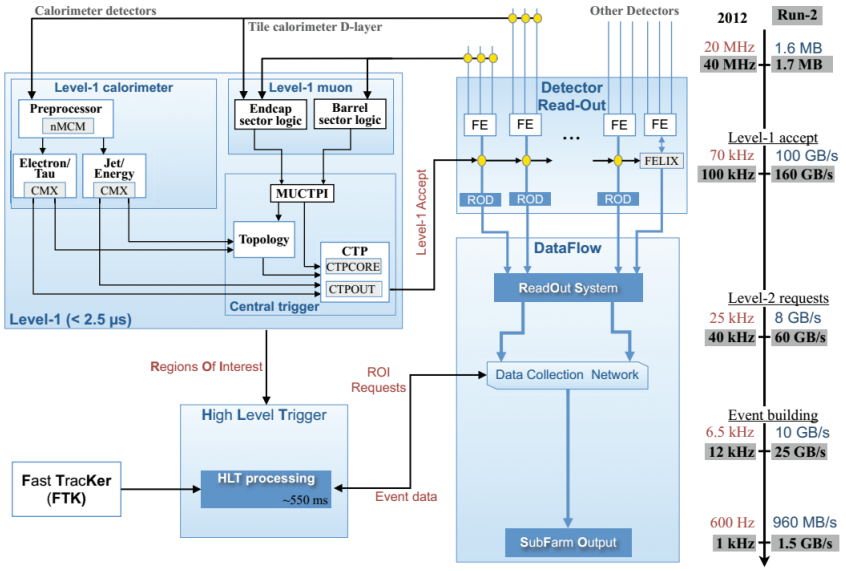


FIGURE 2.2: The ATLAS trigger [17].

total transverse energy. The performance of the trigger is therefore dependent on the quality of its time-energy reconstruction of events. The quality of the reconstruction is inversely proportional to the running luminosity and bunch spacing of the LHC, as these dictate how much time the trigger has to perform the necessary computations.

The Low-Level Trigger

The trigger as a whole functions to reduce both the rate at which events need to be processed and stored, as well as to reduce the absolute number of events that are processed and stored. In Run-2 the Level-1 Trigger consisted of mainly the Calorimeter and Muon Trigger [16]. These systems trigger in the presence of high-p_T muons, electrons, photons, jets and τ-leptons. The Central Trigger Processor (CTP) is seeded by these triggers to make a decision to accept (L1-accept) or reject an event. Upon an L1-accept, the event is read out by the Read-Out Driver (ROD) into the Read-Out System (ROS). The Level-1 trigger defines Regions of Interest (RoI). This information can then be requested by the next stage of the trigger, which is the High-Level Trigger, discussed in the following section. A schematic illustration of the trigger is shown in Figure 2.2.

The CTP is an FPGA-based system with a latency of about 2 µs, capable of performing 15 algorithms. The Calorimeter system was also changed in Run-II; its Application Specific Integrated Circuits (ASICs) were replaced

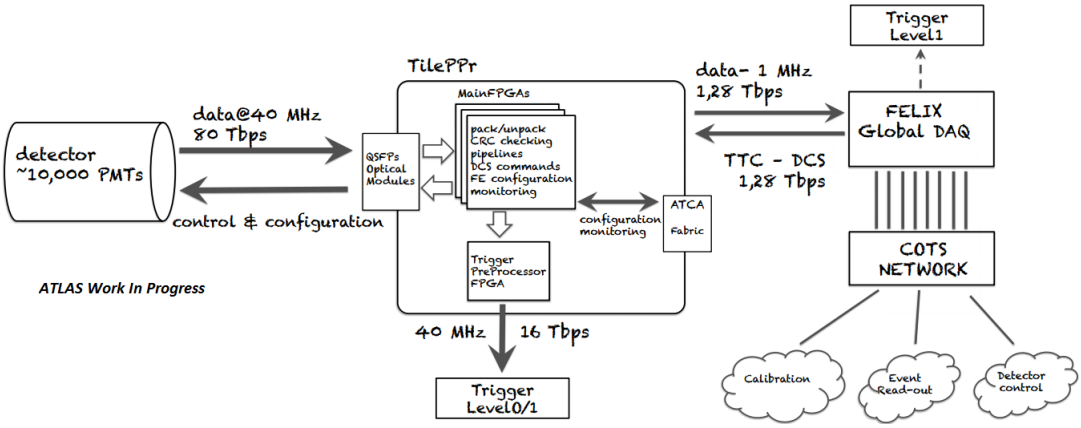


FIGURE 2.3: TileCal Read-Out with TilePPr [18].

with FPGAs to deal with increased pile-up, by giving the system the computational resources to perform an auto-correlation algorithm. In general, the trend in the Level-1 trigger has been to implement trigger and read-out functionality on FPGAs. This is done to keep the trigger rates in a range manageable to the current platform, shown in Figure 2.2. New trigger components are currently being tested: the new Level-1 Topology Trigger will provide topology information to the calorimeter and muon triggers and the Fast Tracker (FTK) will provide them with global tracking information.The ROD is being integrated into the trigger systm as a Trigger Pre-Processor (PPr), and upgraded in bandwidth and functionality. It will interface with the calorimeter front-ends, preparing data for the Level-0/1 triggers, as well as interfacing with the HLT and other platforms related to the detector infrastructure - see figure 2.3. The precise role of the TilePPr system is undecided and as such it is a good candidate for GPGPU experimentation.

The High-Level Trigger

The boundary between the Level-1 and High-Level Trigger is the Region-of-Interest Builder (RoIB) which reconstructs events in small regions of the detector flagged by the Calorimeters and Muon systems, and passes these event fragments to the HLT at a maximum rate of 100 kHz. The fragments are dealt with by the HLT Trigger Supervisor (HLTSV), which distributes them to an HLT node which then processes the event. These systems run on commodity CPU servers, one of which functions as the HLTSV. The HLT nodes run software-based algorithms similar to those used for offline analysis, in order to determine whether an event should be stored permanently. The HLT executes a sequence of algorithms on RoIs known as a trigger chain. More than one trigger chain can be executed on a given RoI, and the

same trigger chain can be used for different RoIs. Algorithms in these chains pass their results to the proceeding algorithm, and each algorithm is more complex than the preceding one, due to using increasingly more complete event data. If a particular algorithm outputs a decision to reject, the trigger chain is marked as failed and the event is discarded. These algorithms are discussed in more detail in section 6.3. The HLT selects events for permanent storage at a maximum rate of 1 kHz, corresponding to a throughput of about 1500 MB/s. The HLT has a latency of about 500 ms.

The software framework utilised by the HLT is known as Athena, based on the HEP software framework GAUDI [19]. The trigger software is a subset of the entire Athena framework, sharing some commonality with offline analysis algorithms, as previously mentioned. It comprises about 4 million lines of C++ code and 1 millions lines of Python code. Code is distributed among approximately 4000 libraries and 2000 packages. Its functionality includes event generation, detector simulation, particle reconstruction, data fragment encoding and decoding, and event management.

Chapter 3

GPU

3.1 GPGPU

General-Purpose Programming with Graphics Processing Units (GPGPU), refers to the use of processors originally designed for graphics rendering (GPUs) for general-purpose computation [3]. General-purpose in this context basically means any computation which is not image-based, frequently simulations of physical systems. It can be defined conversely as the implementation of computations on GPU which are traditionally performed on CPUs. Today GPUs are known as 'accelerators', capable of speeding up computations relative to a CPU by orders of magnitude. However, achieving these speedups is often a complicated task, and sometimes simply not possible. The capabilities and limitations of GPUs, and how this relates to the computations which stand to benefit from GPGPU, are discussed in this section.

3.1.1 The Evolution of the GPU

GPUs as they are known today are a relatively new technology - the term GPU was coined in 1999 [3]. Its predecessors, processors dedicated to rendering images on a screen, go by several names including framebuffers, graphics accelerators, video graphics array, graphics engines, etc. Whatever the case, these processors were designed with the sole purpose of speeding up the process of image rendering. As a result these early GPUs were practically useless for general purpose applications however they have always had a property which is of interest in general purpose computing; that is a high number of processing units, able to hide high latency. In this context latency refers to time of computation. These properties are easily understood in the context of graphics rendering. Firstly, the human eye cannot perceive changes faster than 50 Hz - for instance a light oscillating at 50 Hz will be perceived as a steady light. The refresh rate, or the rate at which the images on a screen are updated, is therefore chosen to be around 50 Hz. In

the case of 1080p HD screens, there are roughly 2 million pixels that must be updated concurrently every 2 ms, and each of those pixels may require hundreds of computations. This leads to some billion computations a second, a highly attractive figure for general purpose computing. Moreover, GPUs have (and continue to be) improved at a rate faster than the familiar Moore's Law predicts - current high-end processors are capable of trillions of computations a second. This scenario, where a large number of computations are performed each clock-cycle, is referred to as high-throughput computing. A 2 ms latency may be sufficiently small for the human eye, but in the context of CPUs where latencies of nanoseconds are possible, it is exceedingly large. This is then a basic principle of GPGPU, it is optimised for processes that are not latency bound: GPUs are optimized for throughput, not latency. This principle will inform its applicability to HEPP processing, as is discussed in chapters 5 and 6. CPUs achieve such low latencies by dedicating much of their hardware to, for instance, complex memory management and branch prediction. In contrast to this, a GPU with a similar transistor count to a given CPU would use those in service of computational activity.

Until around 2000, GPUs were fixed function processors. That is to say they were designed to fulfill the specific role of image processing/rendering, and nothing else. The first attempts to use these GPUs for computations other than image rendering had therefore to frame their problems in the terms of the image processing framework. To explain this concept, an overview of the 'assembly line' nature of early GPUs is listed [3]:

1. The image, or 'geometric primtive' is represented as the collection of vertices of which it is composed.

2. The vertices are then joined to form triangles, all shapes and figures are built up with triangles of varying size.

3. The processor determines how the triangles will be displayed on screen, in terms of the pixel locations they will occupy. It is possible for triangles to overlap, in this case the colour/shading of the pixel will be determined as a combination of the triangles. This process is known as rasterisation.

4. Interpolation, texturing and colouring the pixels.

This is a simplified representation of what is usually referred to as the GPU graphics pipeline. The steps described above are often broken into substeps with for instance a primitive rasterisation occurring after the calculation of the geometric primitive. The hardware itself was optimised to perform these operations, to the exclusion of performing any other operations efficiently. Roughly speaking then, the programmer wishing to use the

GPU for general purpose tasks had to express the task first as a geometrical primitive: the initial states of a problem would be expressed as an image upon which rasterisation/texturing etc. would be performed to determine the evolution of the system. To perform a fluid dynamics simulation, for instance, the numerical solutions to the appropriate partial differential equations would have to be expressed in terms of image rendering operations - the particles would be represented as pixels or groups of pixels. The problem would become similar to the nature of a video, where each frame (or state) is determined from the last frame as per the equations relevant to the real physical system. Suffice it to say this was not a convenient state of affairs for the general-purpose programmer. Nevertheless strides in general-purpose use were made using this paradigm which laid the foundation for future interest in the subject. From this early use of GPGPU it was possible to point to particular characteristics of a problem and gauge whether it would be amenable to implementation on GPU. Subsequent programming languages developed for GPGPU aimed to alleviate the cumbersome task of expressing a given problem in terms of image manipulation, to allow the programmer to express problems in a more familiar way. These languages also aimed to expose and exploit the very characteristics which make a particular problem suitable for implementation of GPU. Section 3.1.3 describes the nature of these characteristics, and also those characteristics which limit the performance of GPU computation. Alongside the transformation of GPU languages from specialised to general-purpose was the transformation of the hardware in the same direction. With fixed-function pipeline GPUs, even the most intuitive user-friendly programming language would have to eventually convert the code to an image manipulation problem - i.e. pose the problem in such a way that it can be computed with the fixed function units available. What was needed for true general-purpose programming was freely programmable hardware, in the same vein as a CPU. From the inception of GPGPU, therefore, the hardware has moved away from a fixed function pipeline with a small amount of programmability, to a large amount of programmability with pipeline included for specialised (image rendering) tasks.

3.1.2 Many-core Processors

Current CPUs are often multi-core, having 2, 4, 8, or 16 cores. This allows the CPU to run threads in parallel, or concurrently. A thread can be understood as a sequence of instructions/operations. Current GPUs have thousands of cores, but they must not be viewed as 'super-CPUs', each core of the GPU is substantially less powerful than a CPU core (see figure 3.1). This

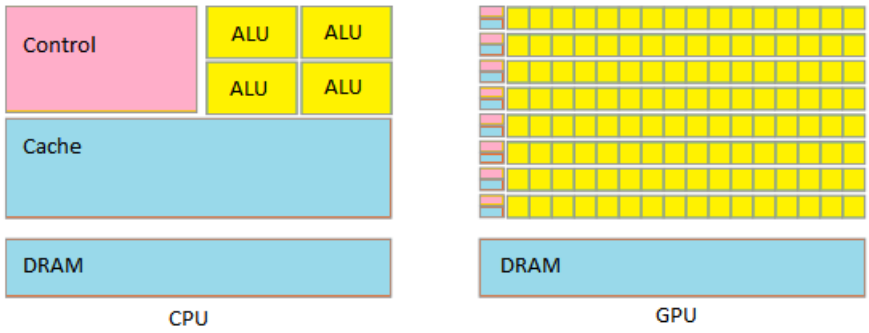


FIGURE 3.1: Schematic representation of CPU vs Modern GPU architecture.

is why compared to a CPU, a GPU can have similar (or better) thermal and energy characteristics despite the large difference in core number. A more detailed description of the GPU architecture is given in section 3.1.5. It is possible to appreciate the many-core structure of a GPU in light of its role as a graphics processor. The GPU is suited to updating millions of pixels simultaneously because each of its many cores is assigned a relatively small number of pixels to deal with at a given time. Importantly, each GPU core functions independently of all the other cores, that is the result given by one core does not depend on the result of any other core.

3.1.3 What Computations Benefit from GPU Implementation?

As mentioned in 3.1.1, GPUs are optimised for throughput as opposed to latency. The analogy given in many texts in comparing GPUs to CPUs is that of several slow cars versus one fast sports-car. To deliver an amount of goods from point A to B, the sports car would need to make several trips while the slow cars would each need to make only a single trip (i.e. deliver them 'in parallel' as opposed to the sports car which delivers them sequentially). Thus, while the sports car (CPU) may be significantly faster than the slow cars (GPU), the slow cars outperform the sports car in this type of problem. The GPU achieves better performance than a CPU because it is capable of massively-parallel computation. The question is then what type of problems lend themselves to massively parallel computation. Two simple examples of problems which benefit from parallel computation are sum reduction and matrix multiplication. Consider a simple sum of integers:

$$\sum_{n=1}^{100} n$$

On a GPU the calculation of the sum of the first and second integers can occur at the same time as the calculation of the sum of the third and fourth integers, which can occur at the same time as the fifth and sixth, and so on. The results of these calculations must then be added together and this can again be done simultaneously.

A CPU would perform this sum sequentially, with only one summation occurring per clock cycle, meaning the entire calculation will take approximately the duration of a clock cycle multiplied by the number of required sums. The time it would take a GPU to do the sum would be approximately the duration of a clock cycle multiplied by the number of required sums, divided by the number of cores. This model is sometimes referred to as single instruction multiple data (SIMD) or single program multiple data (SPMD), here the distinction is unimportant. In reality there are other factors to be considered when determining how long a calculation will take, but for illustrative purposes this explanation is accurate enough. Consider now matrix multiplication whose general form is given by:

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Each element $(\mathbf{AB})_{ij}$ of the product matrix is calculated by an appropriate combination of addition and multiplication (itself just repeated summing), and does not depend on the result of any other element. Therefore each element can be calculated in parallel, resulting in a much reduced computation time with respect to a CPU.

Sum reduction and matrix multiplication are examples of problems that are referred to as embarrassingly parallel, a term intended to convey the ease with which the problems can be implemented on parallel processors. They can be described as inherently parallel or parallelisable - it is not difficult to point to the individual independent elements of the problem that will be assigned to the cores of the GPU. Generally speaking, problems that are suitable for GPU computation have a high amount of parallelism. The problem should also be large enough to benefit from parallel implementation [20]. For instance in a parallel reduce sum, if CPU computation time is $clock\ rate \times number\ of\ sums$ and GPU computation time is $clock\ rate \times number\ of\ sums \div core\ number$, then because the clock rate of a GPU is far smaller than that of a CPU, for the GPU to outperform the CPU it will need as many of its cores occupied as possible. This core-occupancy increases for increasing problem size, so for a program to achieve speed-up from GPU implementation it must be sizable. Lastly, in deciding to use a GPU for a computational problem, latency must not be a priority. To use the analogy of vehicles again, if the goal is to get from point A to B in the

smallest time possible (i.e. low latency), then a GPU probably isn't suitable.

3.1.4 Performance Limits

Usually a problem is neither completely parallelisable nor completely serial, but has portions of both. If there is a non-parallelisable portion then any performance increase will be limited by it. This concept is formalised by Amdahl's law [21]. If T is the total time for a computation, then T/s is the reduced time from using s cores in parallel. If only p percent of the computation is parallelisable then the computation time becomes pT/s plus $(1-p)T$ from the serial portion. Dividing T by $(pT/s + (1-p)T)$ yields Amdahl's Law:

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

Capital S denotes the factor by which computation time can be improved. For instance, if the problem is exactly half serial and half parallel, and the parallel portion can be improved by a factor of 10, then the overall improvement S is only 1.81. If s is increased to 100, S becomes 1.98. It is apparent that the speed-up due to parallelisation is only significant if the parallel portion of the computation is itself significant.

Realistically the speed-up from GPU implementation is not easily predictable based on core count. If a given problem is completely parallelisable, it's speed-up with regard to a CPU will not be T/s but $T/s + O$ where O represents necessary overheads related to the execution of the program from, for instance, memory transfers. Additionally, Amdahl's Law is only helpful if an accurate delineation between parallel and non-parallel portions of a program have been made. To get an accurate sense of the speed-up a GPU can offer, it is usually necessary to actually implement the program on a GPU and take empirical measurements.

3.1.5 The CUDA Programming Platform

Implementing a given program on a GPU in such a way as to allow for the most dramatic speed-up is not trivial. An introduction to the CUDA programming platform is given to illustrate the process a programmer might take when designing a program for massive parallelism. CUDA is a collection of libraries and APIs which allow the programmer to interact with the processing units of the GPU in the most direct and transparent way possible [22, 23]. That is to say, it saves the programmer from having to frame the problem in the way described in 3.1.1. CUDA makes parallel programming more accessible in that it can be used with common programming languages

such as C, C++, and Fortran. It is a propriety platform developed by Nvidia, for use on Nvidia GPUs, and was first released in 2007. A key concept of the CUDA programming model is that of the 'device' versus the 'host'. The GPU, the device, must be connected to a CPU, the host, in order to receive instructions and data.

CUDA Kernel

The CUDA *kernel* can be described as the instruction in the SIMD model. It is similar to a C/C++ function and takes a similar form. Below is an example of a simple kernel, named `Addition`, which adds the integer values stored in vectors `X[i]` and `Y[i]`, and stores the result in `Z[i]`. Aside from syntactical differences, the kernel should not be too unfamiliar to the reader with a basic knowledge of C/C++ and many other languages.

```
__global__ void Addition(float* X, float* Y, float* Z)
{
int i = threadIdx.x;
Z[i] = X[i] + Y[i];
}
int main()
{
...
// Kernel invocation with N threads
Addition<<<1, N>>>(X, Y, Z);
...
}
```

The significant feature is the introduction of the variable `threadIdx`. This variable is not defined by the programmer but rather it is 'built-into' CUDA, explained below.

Thread Hierarchy

The thread in CUDA has a similar meaning to that usually used in reference to computations; the execution of a set of instructions. In CUDA, the kernel is this set of instructions and can be executed on many threads in parallel. In the parallel reduction described in 3.1.3, the sum of integers 1 and 2 can be executed by instructions given in the kernel on thread 1, while the sum of integers 3 and 4 can be executed concurrently using the same kernel but a different thread, say thread 2. The variable `threadIdx` is used to identify the thread number - which each thread has. For instance, in the example kernel above, the instruction `int i = threadIdx.x` is given. In thread

1, this instruction will assign the value one to `i`, in thread 2 the value two, and so on. Thus in the kernel, thread 1 will assign the value 1 to `i`, and in the following line add together `X[1]` and `Y[1]`, the first values of array `X[i]` and `Y[i]` respectively. Each thread can therefore use the same kernel to return the appropriate result.

Thread identification is not limited to a single integer value, because threads are not only organised one-dimensionally. As illustrated in figure 3.2, threads are bundled into collections known as blocks. The size of the blocks, i.e. the number of threads per block, is decided by the programmer. Blocks, in turn, are grouped into grids - the size of which is also determined by the programmer. Also illustrated in figure 3.2 is the fact that the programmer can decide how to arrange the blocks within the grid - as can be seen in the difference between grid 0 and grid 1. In this arrangement each thread has not only an identity `threadIdx.x` (the 'x-coordinate'), but also an identity `threadIdx.y` (the 'y-coordinate'). If the programmer so wishes, grids can be three dimensional, and each thread will be specified by 3 'coordinates'. The size of the blocks and grids is specified when the kernel is called. In the above example the specification is `Addition«<1, N»>(X, Y, Z)`, which results in a one-dimensional grid with N threads. One can begin to see intuitively why GPUs succeed in tasks such as matrix multiplication; the structure of the threads is essentially in matrix format.

Nvidia GPU Memory Structure

Figure 3.2 shows the types of memory available on a (Nvidia) GPU: local, shared, and global memory. There is also a type of memory cache, the texture memory, which is a specialised form of memory not considered here.

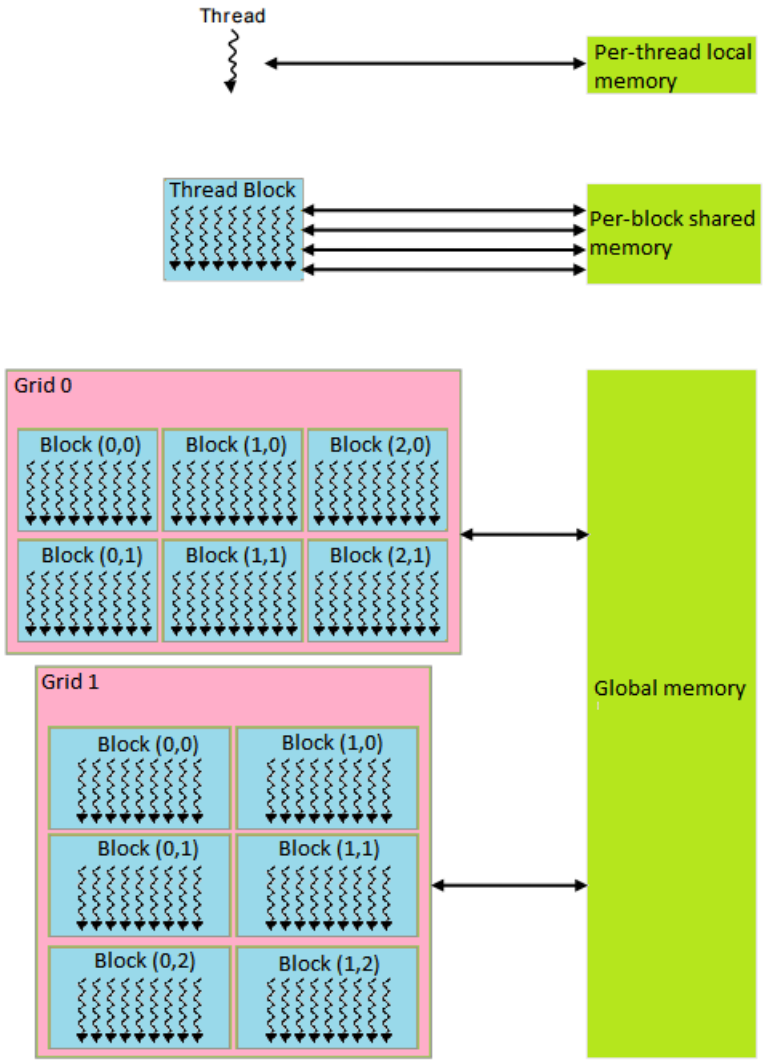


FIGURE 3.2: Memory types on an Nvidia GPU.

As figure 3.2 indicates, local memory is a small, low-latency memory that can be accessed only by the thread which owns it. Shared memory is accessible to all threads within a given block, and is thus a way for threads in the block to communicate with one another. Global memory is a large memory accessible to any thread in any block (hence global). The size of global memory depends on the GPU, but is on the order of 16 GB. Global memory is also the slowest of the three memory types. GPUs 'hide' memory latency, that is compensate for the slowness of memory accesses, by executing a group of threads while the memory accesses for another group of threads is occurring. It is again the prerogative of the programmer to decide how to use the memory available. If data needn't be accessed by more

than one grid, it is of course undesirable to use global memory. However it might be the case that the data is too large for shared memory, in which case the programmer must split it up and send it to shared memory when it is needed. Data local to a single thread must eventually be passed to global memory to make it accessible to the rest of the memory. The size of blocks and grids is limited by memory size. As an example, the Nvidia Jetson TK1 GPU has a limit of 1024 threads per block, and maximum grid dimensions of (1024, 1024, 64).

Optimising all these parameters for a given problem is the challenge for the programmer.

Chapter 4

GPU Characterisations, Benchmarks, and Comparisons

4.1 Introduction

In this chapter data are presented relating to the performance of GPUs. The GPUs tested allow the reader to get a sense of the range of GPUs available and their performance relative to each other and to a CPU.

4.2 Hardware

The devices used for these tests are three NVIDIA GPUs:

1. Tegra K1 [24]: Kepler GPU with 192 cores, 17 GB/s memory bandwidth, with a Quad-Core ARM Cortex A15 CPU on-board. Power consumption of 11 W.
2. GeForce 840m [25]: Maxwell GM108 GPU, 384 cores, 16 GB/s memory bandwidth. Power consumption of 30 W.
3. Tesla K80 [26]: 2x Kepler GK210 GPUs with 4992 cores, 480 GB/s memory bandwidth. Power consumption 300 W.

A CPU was used; the Intel® Core™ i7-5500U [27]. It is a dual-core CPU which can support a maximum of four concurrent threads. 'Maxwell' and 'Kepler' refer to proprietary Nvidia GPU architecture. Maxwell is the newer of the two and the trend has been increased performance with each new generation of architecture. However, the reader will note that the GPU with the best results is the K80, as even with the older architecture its memory bandwidth and large core count gives it the edge. Memory bandwidth is a measure of how quickly the GPU can read in data and supply it to its processing cores. GPUs 1 and 2, as well as the CPU, are located in the High-Throughput Electronics Lab at the University of the Witwatersrand, GPU

3 is located in CERN, Geneva, and was accessed remotely. Tests were conducted by the author.

These GPUs were chosen to represent 1) a low-end GPU intended for mobile devices, 2) a midrange GPU intended for consumer computer platforms and 3) a sever-grade GPU intended for high-end, high-throughput applications. These benchmarks will therefore give the reader an idea of the spectrum of performance available. The CPU chosen is a high-end consumer device chosen to illustrate the performance of a top-of-the-range CPU against these GPUs.

4.3 Methods

Matrix multiplication benchmarking tests were carried out on the GPUs and the CPU. Square matrices of dimension 32 increasing in powers of 2 to 4096, were used. These dimensions are commonly used as they are conducive to maximum GPU performance due to the configuration of the cores and memory. On GPU the tests were carried out with CUDA, using the `matrixMulCUBLAS` code available from Nvidia as part of their linear algebra library. On CPU, matrix multiplication was performed using MATLAB's inbuilt matrix multiplication functionality.

Sum reduction benchmarking was carried out on the GPUs with CUDA code available from [28]. On CPU the tests were conducted using MATLAB's inbuilt array summing functionality.

Memory bandwidth tests were performed on the GPUs using CUDA's `bandwidthTest` code, also available from Nvidia.

4.4 Results and Analyses

4.4.1 Sum Reduction

Tables 4.1 to 4.4 present the results of the sum reduction benchmarks, of the type described in section 3.1.3. The parameter N refers to the number of elements in the array to be summed. 'GB/s' refers to the rate at which data is retrieved from memory, 'Percent' indicates the fraction of total memory bandwidth in use. The time taken for the computation is given in microseconds. The accuracies quoted are conservative, and reflect the fluctuating tendency such benchmarks have in response to temperature, the non-deterministic nature of digital circuits, uncontrollable patterns of memory interaction, and the inability to accurately account for programmatic overhead as a percent of total run-time.

TABLE 4.1: Parallel sum reduction performance for the Tegra K1.

GK20A @ 14.784 GB/s			
N	GB/s ($\pm$0.1)	Percent ($\pm$0.1)	ms ($\pm$0.01)
2^{20}	7.0	47.5	0.59
2^{21}	8.3	56.3	1.00
2^{22}	9.4	63.8	1.77
2^{23}	12.2	82.8	2.74
2^{24}	12.0	81.7	5.55
2^{25}	12.2	82.7	10.96
2^{26}	12.2	82.9	21.88

TABLE 4.2: Parallel sum reduction performance for the GeForce 840m.

GeForce 840M @ 14.400 GB/s			
N	GB/s ($\pm$0.1)	Percent ($\pm$0.1)	ms ($\pm$0.01)
2^{20}	12.7	88.7	0.32
2^{21}	13.2	91.8	0.63
2^{22}	13.4	93.6	0.12
2^{23}	13.6	94.5	0.24
2^{24}	13.6	94.5	0.49
2^{25}	13.6	94.7	0.98
2^{26}	13.6	94.9	1.96
2^{27}	13.7	95.3	3.91

The tables indicate the maximum memory throughput rates achieved by the benchmarking application (not present for the CPU). That each throughput rate is lower than the rates specified in section 4.2 is an illustration of an important point. The performance parameters given by manufacturers regarding their products represent an often unattainable best case scenario. The specifications are either theoretical upper limits, or were actually attained through a combination of software written for the express purpose of extracting the best performance (as opposed to representing a realistic scenario), and extremely efficient cooling systems, for instance liquid nitrogen cooling of the processor during testing.

For each GPU it can be seen that increasing the problem size results in increased throughput. Comparing the performance of the Tegra K1 to the CPU graphically, in figure 4.1, the notion that GPU outperforms CPU for ever increasing problem size as described in section 3.1.3, is borne out. The Tegra K1 was chosen for comparison because it is the worst performer of

TABLE 4.3: Parallel sum reduction performance for the Tesla K80.

Tesla K80 @ 240.480 GB/s			
N	GB/s (± 0.1)	Percent (± 0.1)	ms (± 0.01)
2^{20}	87.8	36.5	0.47
2^{21}	106.3	44.2	0.78
2^{22}	123.4	51.3	0.13
2^{23}	133.1	55.3	0.25
2^{24}	136.5	56.8	0.49
2^{25}	137.0	56.9	0.97
2^{26}	141.6	58.9	1.89
2^{27}	161.1	66.9	3.33

TABLE 4.4: Serial sum reduction on the Intel Core i7-5500U

Intel Core i7-5500U	
N	ms (± 0.01)
2^{20}	0.94
2^{21}	2324
2^{22}	4.39
2^{23}	9691
2^{24}	18.4
2^{25}	35.07
2^{26}	74.84
2^{27}	16.95

the three GPUs. This is to illustrate how even a low-end GPU can outperform a high-end CPU in the right circumstances. As expected, the Tesla K80 outperforms the GeForce 840m, which outperforms the Tegra K1.

4.4.2 Matrix Multiplication Benchmarks

Figure 4.2 illustrates the performance of the GPUs in floating-point operation per second (FLOPS), a measure of the number of calculations per second achieved by the GPU. Again, it can be seen that the highest performance is attained for the bigger problem sizes. It is interesting to note that the Tesla K80 begins to outperform the GeForce 840m only after the matrix dimensions increase beyond 256. This indicates that, GPUs in general perform better for large problem sizes, and GPUs with high core-counts may not outperform those with lower core-counts for certain problem sizes. The same is true for the computation latencies, illustrated in figure 4.3.

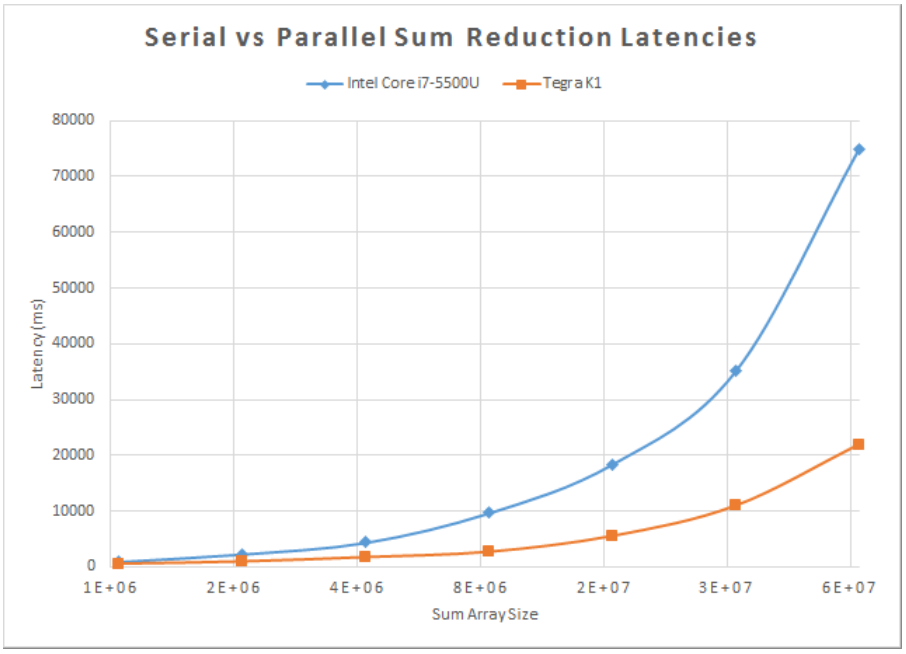


FIGURE 4.1: GPU outperforms CPU for sum reduction problems.

TABLE 4.5: Bandwidth of Host-Device Intercommunication

Device	H-to-D (MB/s)	D-to-H (MB/s)	D-to-D (MB/s)
Tegra K1	6444.5	6461.3	10320.5
K80	7343.2	7341.3	141281.3
840M	1528.0	1615.9	13181.7

Figure 4.4 compares the matrix multiplication latencies of the Tegra K1 and the CPU. Again, the GPU begins to outperform the CPU only for sufficiently large problem sizes.

4.4.3 Memory Bandwidth Benchmarks

Table 4.5 presents bandwidth of communication between host and device, in both directions, and communication between the GPU and its Random Access Memory (RAM), i.e. device-to-device. It is interesting to note that the Tegra K1 performs almost as well as the high-end Tesla K80 for host-to-device communication (and vice versa). This is because, as stated in section 4.2, the Tegra K1 has a CPU on-board, and device-to-host communication is not performed over PCIe interface.

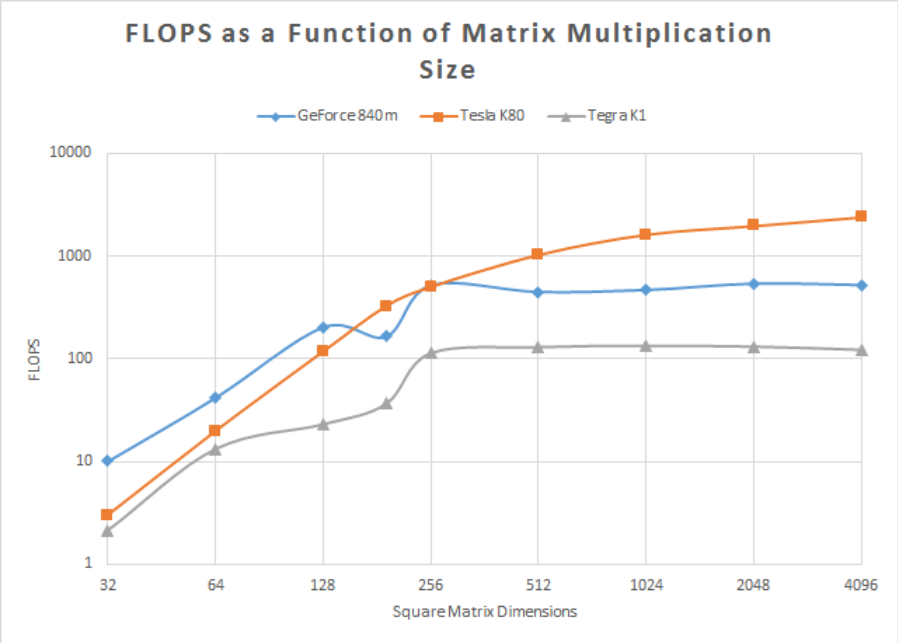


FIGURE 4.2: GPU performance in FLOPS for matrix multiplication.

4.5 Selecting GPUs for GPGPU Application

A number of factors make it difficult to select an appropriate GPGPU platform for a given problem. The technology is relatively new and it is progressing quickly. If a researcher attempts to achieve optimum performance from a particular GPU for the duration of, say, a year, more advanced devices will be available at the termination of the study than at the start. While new devices are intended to be as compatible as possible with code from a previous generation GPU, it is not realistic in the long-run. Additionally manufacturers often do not release detailed technical specifications of their devices, which does not concern the average consumer but it is often essential knowledge in high-throughput environments like ATLAS. Parameters which are provided by manufacturers are to be interpreted as ideal, and not necessarily realistic.

As the results in this chapter have shown, it is not a case of the most expensive, high-end GPU performing the best, but rather the nature of the computation determines which GPU is best suited. Additionally, the best performance may be sacrificed for lower cost and better energy efficiency.

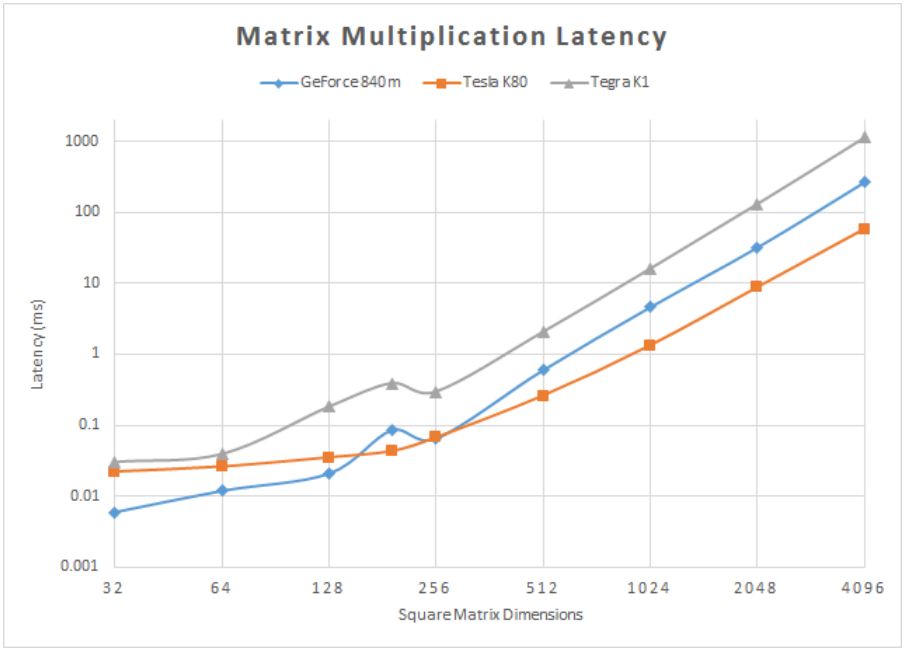


FIGURE 4.3: GPU latencies for matrix multiplication.

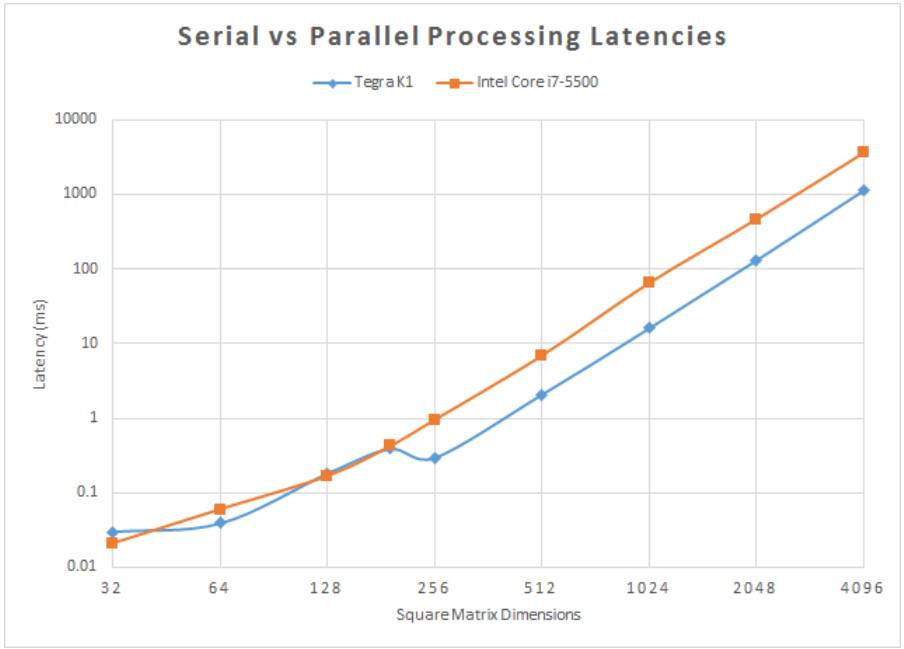


FIGURE 4.4: GPU vs CPU for matrix multiplication.

Chapter 5

GPGPU in ATLAS Level-1 Trigger

5.1 ATLAS Level-1 Trigger

5.1.1 FPGA Use in Trigger Front-end

As mentioned, FPGAs are a key technology used in the Level-0/1 Trigger. They are used in big data environments in general, such as the SKA project. The 'Field Programmable' in FPGA refers to the fact that the individual logic units on-board can be programmed into any type of configuration any number of times [29]. This is in contrast to, for example, CPU and GPU technology where the configuration of logic units is fixed, and only the software programmed on them is customisable. The customisability of FPGAs allows for the programming of functional units tailored to perform a specific job with high efficiency. The disadvantage of this is that FPGAs are not as easy to program as a CPU or GPU. FPGAs are programmed with a low-level Hardware Description Language (HDL). In general, low-level programming languages are more efficient than their high-level counterparts. FPGAs are also more expensive and energy consumptive than CPUs or GPUs.

Because The Level-0/1 Trigger reacts in real-time to data coming from the muon and calorimeter systems at the 40 MHz bunch crossing rate, the latency of these stages is around 2 µs. FPGA technology has been implemented in spite of its drawbacks because of the extreme latency constraints of the Level-1 Trigger. The reason the FPGA portions of the Trigger are referred to as hardware-based as opposed to software-based is that FPGAs do not waste any clock cycles executing software protocols. Each clock-cycle, a useful computation is performed. This, as well as the fact that FPGAs are capable of massive parallelism in a similar way to GPUs, make them an ideal choice for the Level-1 Trigger.

5.2 GPGPU in the Level-1 Trigger

As luminosity is increased at the LHC until its maximum in the HL-LHC, and new Level-1 Trigger components as well as new detector infrastructure are being developed and tested, the question discussed here is how, if at all, could GPGPU be utilised at this level to improve physics results. For GPGPU to be considered for use, it must meet relevant technical requirements and it must meet a reasonable cost-to-benefit ratio relative to other technologies. As GPGPU platforms can be coded in CUDA (see 3.1.5) and other C-like languages, whereas FPGAs must be programmed in HDL, GPGPU is an attractive option for developing more complex algorithms for use in the Level-0/1 Trigger. More advanced algorithms are needed to ensure both that Trigger rates will remain manageable, and that physics reconstruction remains accurate and therefore valuable for analysis. The benefit of a programming language both more familiar and more accessible than HDL should not be underestimated. In researching potential Trigger algorithms for the HL-LHC, researchers would require more time between conception and testing of an algorithm using FPGA than they would using GPU (or CPU) - using GPGPU could expedite this process. In terms of cost, energy-efficiency, and ease-of-use, GPGPU is preferable to FPGA.

5.2.1 Limitations of GPGPU in Online Environment

Due to thermal restrictions, 2 µs is a prohibitively low latency for many GPUs to operate under. GPUs either can't achieve latencies as low as this, or if they can the time is insufficient for producing useful results because under such a small latency, workload is likely to be insufficiently large to benefit from parallelism (see chapter 4). This is partially due to the limited memory speeds at which GPUs can receive data from the host or CPU. New technology is being developed by both academic institutions and corporations such as Nvidia to allow GPUs to transmit and receive data to and from a host at low latencies. At this stage in its development, using non-commercial products to achieve the necessary latencies would likely present difficulties in programming and interfacing greater than the difficulties associated with FPGAs. Current commercially available high-bandwidth technologies such as Nvidia's GPUDirect and NVLink, which could possibly function at sufficiently low latencies, can do so with only a limited number of I/O devices. Any communication with surrounding front-end devices would imply a transfer bottleneck, again rendering GPU use in this environment useless.

Because GPUs perform so poorly in low-latency environments, it would not be possible to simply replace FPGAs or place them between the TilePPr and front-end or between TilePPr and Level-0/1 directly without introducing unacceptably high latencies, hindering Trigger rates.

5.2.2 Heterogeneous Co-Processing Unit

The alternative is to introduce GPGPU outside the main path of data flow in Level-0/1. In this scenario GPGPU would interface with the TilePPr/Level-0/1 via a CPU capable of latencies small enough not to impede the efficiency of the Trigger. The CPU would receive the relevant data and pass it to the GPU for processing, after which the result would be passed back to the CPU and then to the TilePPr. Since the GPU operation would see latencies orders of magnitudes larger than that of the entire Level-0/1 Trigger, it would not function synchronously to the 40 Mhz bunch crossing. Instead it would function as a co-processing unit (Co-PU), in parallel with the Trigger. There are two ways in which the Co-PU could be configured:

- The TilePPr/Level-0/1 Trigger could query the Co-PU, the Co-PU then processes data and passes the result back to the TilePPr or the Level-0/1 Trigger either when it's inactive during running or after running.
- The TilePPr/Level-0/1 Trigger could query the Co-PU, the Co-PU processes the data and passes it to the HLT, either directly or through the TilePPr.

Before evaluating what type of queries could be passed to the Co-PU, i.e. what type of algorithms the Co-PU could be expected to perform, a more detailed analysis of the possible interfaces the Co-PU could have with the TilePPr and/or Level-0/1 is necessary.

A GPU-ARM based PU

One option is to interface GPU to the TilePPr/Level-0/1 Trigger using ARM CPUs, such an interface is illustrated in figure 5.1. ARM CPUs are an open-source, low-cost, energy-efficient alternative to the common x86 CPU architecture of, for instance, Intel. Cox [30] has demonstrated that it is possible to interface ARM CPUs to the TilePPr (in [30] it is still referred to as the sROD) through PCIe technology. PCIe is a type of electronic communication protocol used to, for instance, interface CPUs to peripherals such as GPUs and soundcards. It transmits data serially through a bus of up to 16 lanes. The PCIe standard is set by PCI-SIG, who control the specifications

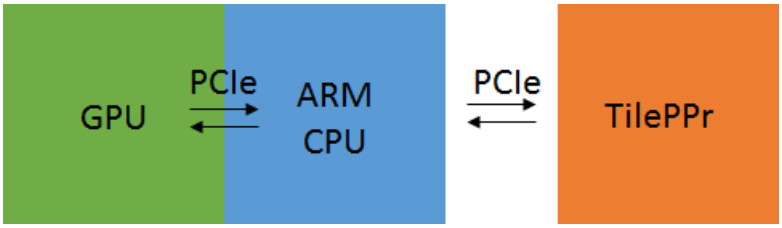


FIGURE 5.1: Schematic of Co-PU - TilePPr Interface

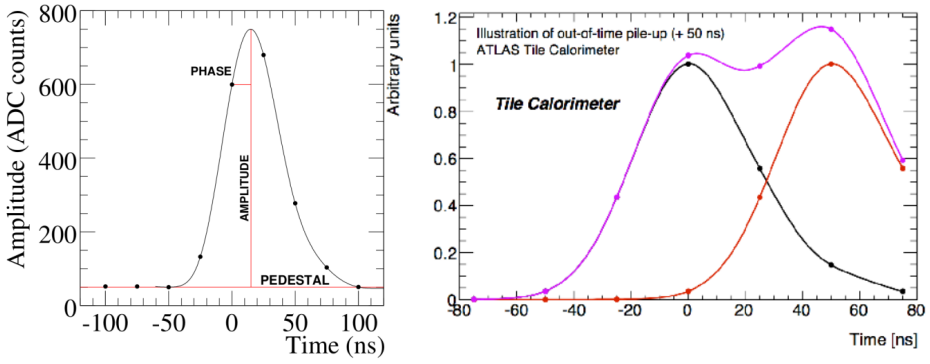
of the technology, which can be found in [31]. Such an interface between an ARM device and the TilePPr would allow for data transfers with latencies in the order of hundreds of nanoseconds. This type of interface would therefore not interrupt Trigger rates. However, the latency of the transfer of data from ARM to GPU and vice versa, as well as the latency of the GPU computation itself, disqualifies a GPU-ARM Co-PU from acting synchronously with the Trigger, or in real-time. In this instance the Co-PU would act either as a quasi-offline or entirely offline platform. This notion is explored further below in section 5.3 with reference to specific functionality the GPU could have.

5.3 Potential Functionality for GPU to Execute

5.3.1 Processing of High Quality-Factor Events

In TileCal, a charged particle will strike the plastic scintillator tiles causing them to luminesce in proportion to the energy lost due to ionisation of the particle. The wavelength of light from the tiles is shifted by wavelength shifting fibres and fed to PMTs which convert the signals to an amplified voltage which can then be digitised for computer analysis. The digitised samples from the PMTs are used with weights in the so-called Optimal Filtering (OF) algorithm (optimal in its signal-to-noise ratio (SNR)) to reconstruct the energy deposition of a particle, resulting in the waveforms of the sort shown in figure 5.2a [32].

The OF algorithm has as its output the phase τ, amplitude A, and pedestal ped of the waveform, as can be seen in figure 5.2a. These parameters are reconstructed with the weighted sums shown in equation 5.1. The seven terms of these sums are the seven digital samples taken by the analogue-to-digital converter (ADC). The weights are chosen so that the variance of the parameters are minimized relative to electronic noise as well as noise from pile-up. The weights are determined with offline calibration, using



(A) (B)

FIGURE 5.2: Ideal pulse-shape and parameters of interest for TileCal samples (left), and modified pulse shape by out of time pulse (right). [32].

Langrange multipliers, and stored in the detector for online operation.

$$A = \sum_{i=1}^{7} a_i s_i \quad \tau = \frac{1}{A} \sum_{i=1}^{7} b_i s_i \quad ped = \sum_{i=1}^{7} c_i s_i \tag{5.1}$$

The signal can then be described with

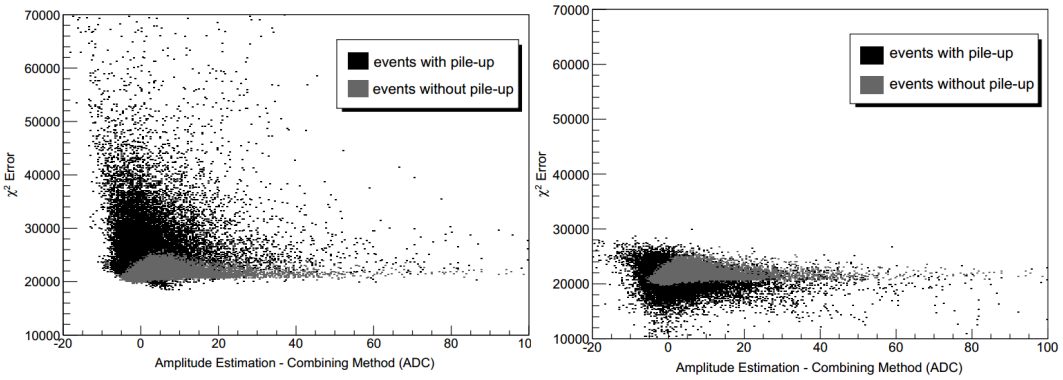
$$s(t) = Ag(t_i - \tau) + ped \tag{5.2}$$

Where $s(t)$ is the reconstructed signal, t_i is the time the sample is taken, and g is the a priori expected pulse shape.

Also defined is the Quality Factor (QF) of the reconstruction, shown in equation 5.3, which calculates the discrepancy between measured and expected signal. Large QFs are therefore an indication of a signal degraded from noise, pile-up, or other errors.

$$QF = \sum_{i=1}^{n} [s_i - (Ag_i + A\tau g_i^{'}) + ped] \tag{5.3}$$

Currently, if energy reconstruction fails, the raw data from these events can be saved for offline analysis. The threshold for a failure decision is preprogrammed and limited by the size of raw event data relative to reconstructed data - raw data consumes more bandwidth. These borderline or failed events are designated for offline analysis because online analysis is limited by latency constraints, and therefore not as accurate. Practically this results in online calculations being done in fixed point arithmetic using the simplest form of the OF algorithm. Offline analyses can use floating point



(A) (B)

FIGURE 5.3: The effect of pile-up on energy reconstruction χ^2 error for the OF algorithm (left), and the same for the signal deconvolution technique (right) [8].

arithmetic as well as a more computationally intensive and accurate iterative form of the OF algorithm or any other kind of reconstruction algorithm. It is possible to offload these failed events to the Co-PU with a more liberal threshold.

The are other techniques which provide more accurate energy reconstruction results than OF, especially in the presence of high pile-up, that the Co-PU could implement in the case of a failed event reconstruction. For instance, a signal deconvolution technique where the signal is recovered using a known template for the pulse shape, in the presence of additive white noise. Figure 5.3 shows how the deconvolution algorithm results in a substantially smaller χ^2 error in the presence of pile-up (figure 5.3b) relative to the OF algorithm's performance (figure 5.3a).

As developing algorithms for GPU is a relatively simple and quick process, the scope is large for improved energy reconstruction algorithms that could be used to reconstruct events that OF could not.

5.3.2 Facilitating Less Dead-time

Dead-time can be applied either between the calorimeter and muon Triggers and the CTP, or between the CTP and the HLT [16]. In the first case, after an L1-accept, a dead-time of four bunch crossings is applied, meaning the Trigger detectors will not pass L1-accept for those crossings. In the second case, dead-time is applied when the HLT passes a busy signal to the CTP indicating that the rate of L1-accepts is exceeding its ability either to read or process data. In this case it is possible that the Co-PU could function as a

buffer between the CTP and HLT, storing events which the HLT cannot accept, instead of increasing the number of HLT nodes. It needn't be restricted to the role of a buffer, and could function as a sort of primitive HLT in its own right, relieving the HLT of these events and reducing dead-time. A reduction in dead-time is an increase in integrated luminosity, an opportunity to decrease dead-time is therefore extremely valuable.

5.4 Conclusion

It is clear that as the TilePPr and Level-0/1 Trigger architecture are still being decided that there is a possibility for the inclusion of a component such as the proposed Co-PU. It also seems possible from the results in [30] to integrate such a Co-PU into the Level-0/1 Trigger without introducing unacceptably high latencies which would hinder Trigger rates. Additionally there appear to be tasks that the Co-PU could perform, as outlined in section 5.3, which could alleviate the Trigger from certain burdens, as well as decrease dead-time and therefore increase effective luminosity. What is not clear, however, is the necessity of the inclusion of GPU in this Co-PU. Since the inclusion of GPU would disqualify the Co-PU from operating online in this environment, it becomes difficult to justify its use. If the Co-PU were to operate offline, this would imply that it would have essentially an arbitrarily large amount of time to complete any operation. Any speed-up from GPU may then be irrelevant or unneeded. However if it can be demonstrated that, over time, including a GPU or GPUs in the Co-PU results in a more energy efficient platform, and if it could be further demonstrated that this energy offset compensated for the difficulty and cost of the design of a heterogeneous Co-PU platform, then the use of GPU may be justified.

Chapter 6

Heterogenous Computing Platforms in the ATLAS HLT

6.1 Introduction

This chapter is structured as follows. First a description is given of how the LHC upgrades will affect the HLT system, and why GPGPU is a likely solution to foreseen issues. The ATLAS GPU demonstrator project is then discussed, this project was initiated at ATLAS to test the viability of GPGPU in the HLT. The tests conducted are described, and the results are presented and analysed in detail.

6.1.1 HLT Upgrade

In the *Letter of Intent for the Phase-II Upgrade of the ATLAS Experiment* published by CERN detailing the upgrades to ATLAS [33], two criteria are given to motivate upgrades to the HLT platform. First, increased Level-1 acceptance rates implying higher throughput for the HLT. The HLT must deal with events of higher complexity by running algorithms of higher complexity. The HLT must also take advantage of evolving computer architecture allowing for more efficient computing while remaining affordable. The use of parallel architecture is mentioned specifically: "[it] is now evident that our code must use multi-threading, vectorization and parallelization". Second, a redesign which does not commit to a single programming language or paradigm and which allows for increased I/O throughput: "ATLAS must plan for a heterogeneous storage deployment using an array of technologies, and its I/O framework and persistent data organization must be flexible enough to be configurable, and to allow optimization for the range of storage implementations encountered." These criteria are considered now in relation to the GPU demonstrator.

6.2 GPU Demonstrator

The ATLAS GPU demonstrator group has been formed to test the performance of GPGPU in the HLT, to determine its viability. The GPU demonstrator aims to determine the type of speed-up that can be expected, how difficult adapting Athena for use on GPU is, and which processes are suitable for GPU implementation. The results of these tests will inform the design of the heterogeneous HLT servers regards infrastructure and software design.

6.2.1 Accelerator Process Extension

Due to the complex hierarchical object-oriented nature of ATHENA, as well as because a lot of existing code was written for serial processing; altering the HLT software to be compatible for use on GPU is not a trivial task. The existing code must satisfy certain lexical differences in software language as well as be converted into parallel structures. Initially, over a period of two years, attempts were made to covert the C++ and Python code directly into CUDA, this proved inviable due to the volume of code that would need to be converted without introducing unexpected bugs and inefficiencies.

A core software designed for the GPU demonstrator allows this problem to be circumvented. It is called the Accelerator Process Extension (APE), and allows the existing ATHENA HLT software to interface with any given platform other than the CPU processors for which it was originally designed [34]. In this scenario, APE allows the HLT to interface with a CUDA GPU platform without changes to ATHENA or CUDA code. The trigger does not need knowledge of what processor it is interacting with, this is concealed by APE. As such APE is not limited to interfacing ATHENA and CUDA but can be altered to interface ATHENA to any platforms. This fulfills the criterion stated above regards flexibility, however to configure APE to run on a given platform may not be realised so easily (see section 6.5.2). Figure 6.1 illustrates the interface between ATHENA and three separate platforms; an Nvidia GPU, an Intel CPU, and a GPU-CPU heterogeneous platform. APE has three components: the Manager, the Module, and the Work item. The Manager is responsible for communication with ATHENA and scheduling requests from ATHENA. The Modules control GPU resources such as memory, and create Work items; which are CUDA kernels.

ATHENA algorithms must format data and query a software component known as Offload Service to communicate to APE. A message passing software, YAMPL, allows ATHENA and APE to receive and pass data in their

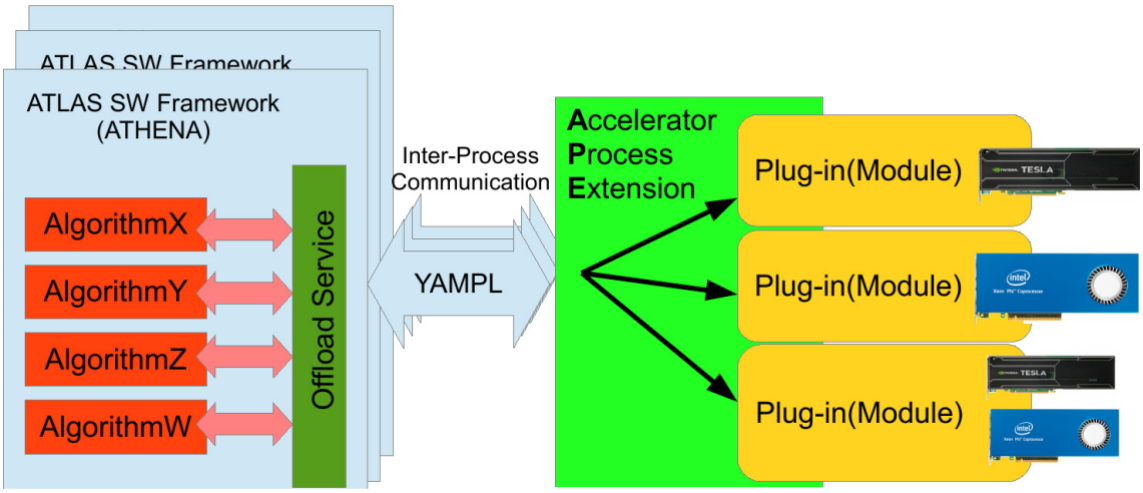


FIGURE 6.1: Through APE, ATHENA can interface with an arbitrary platform [35].

native format. APE then passes requests to Modules which execute algorithms on the destination platform. The result can then be passed back to ATHENA following the reverse path. This is accomplished in the following way, illustrated in figure 6.2. The HLT must make a request to a software component referred to as TrigDetAccelSvc. TrigDetAccelSvc converts the data in question from ATLAS Event Data Model (EDM) format to one more suitable for GPU, using Data Export Tools. Data Export Tools is a serial operation and hence limiting to speed-up through parallelisation. The formatted data is then communicated to APE via OffloadSvc, which also returns the data back to TrigDetAccelSvc, where it is integrated into the Athena EDM, and passed back to the HLT. This framework allows ATHENA to request the GPU platform to run particular HLT algorithms, the following section discusses these algorithms that have been implemented by the GPU demonstrator group.

6.3 Algorithms Tested by GPU Demonstrator

The algorithms identified as most time consuming by the GPU demonstrator team, and therefore of most interest for speed-up, are the ID tracking, calorimeter clustering, and muon tracking algorithms. Inner detector tracking has been tested and iterated the most thoroughly, and will be discussed. Preliminary results of ID tracking speed-up from GPU implementation are presented. Also discussed is the calorimeter clustering algorithm, muon tracking will not be discussed as it is still under development.

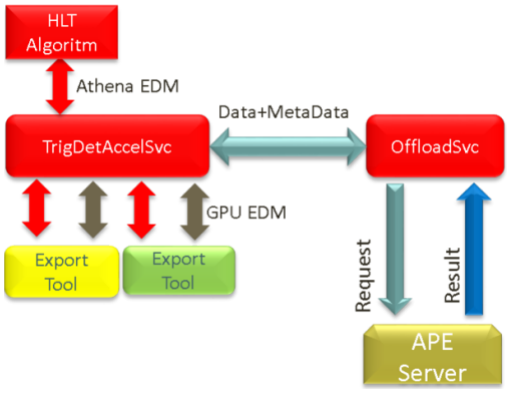


FIGURE 6.2: Schematic of the protocol through which ATHENA communicates with APE [34].

6.3.1 Inner Detector Tracking

ID Tracking is the most time consuming HLT algorithm, and with increased pile-up, required CPU resources increases combinatorially. ID tracking can be broken into four steps [34]:

1. Bytestream decoding: A bytestream comprises information about pixel hits in a format optimised for data transmission. The components forming the bytestream are referred to as words. Each word is assigned to a GPU thread, resulting in decoding done in parallel. The decoded bytestream represents data in a topographical format, more convenient for physics calculations.

2. Clustering: Usually a particle will be detected by more than one sensor module. Groups of sensors activated by the same particle are grouped together into clusters by a Cellular Automata (CA) algorithm. The CA algorithm is naturally suited to parallel implementation. Each sensor module with a hit is assigned a thread, and the algorithm iterates until all adjacent hits are combined into clusters. A cluster is described by a centre coordinate and the sum of the energy of the adjacent hits. This process is illustrated in figure 6.3.

3. Track seeding: As the name implies, track seeds form the base of the tracks that particles trace in the detector. Seeds are formed from pairs of hits in adjacent layers of the pixel detector. A 2D thread array loops over clusters to find suitable pairs of hits. A second 2D thread array repeats the process, forming track triplets. Triplets are merged to form track candidates. This process is illustrated in figure 6.4, read from top left to bottom right.

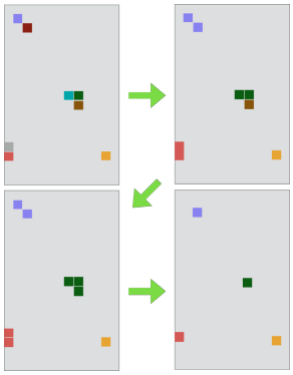


FIGURE 6.3: ID tracking: clustering process [34].

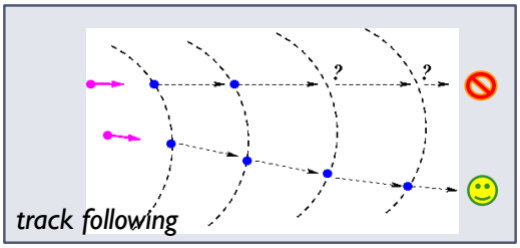


FIGURE 6.4: ID tracking: track generation [36].

4. Clone removal: The final stage is removing track candidates with the same outer layer hits but different seeds, referred to as clone removal.

6.3.2 Calorimeter Clustering

The serial Calorimeter Clustering algorithm classifies calorimeter cells into three categories based on their signal-to-noise ratios. The categories are: Seeds for SNR > 4, Growing for 4 > SNR > 2, Terminal for 2 > SNR > 0. The algorithm joins all Growing cells to adjacent Seeds to form clusters. Growing and Terminal cells continue to be added to clusters until there are no more to add, or the cluster reaches maximum size. On GPU, each cell is assigned to a thread. Each cell is joined to the neighbouring cell with the highest SNR. The algorithm terminates under the same conditions as the serial implementation.

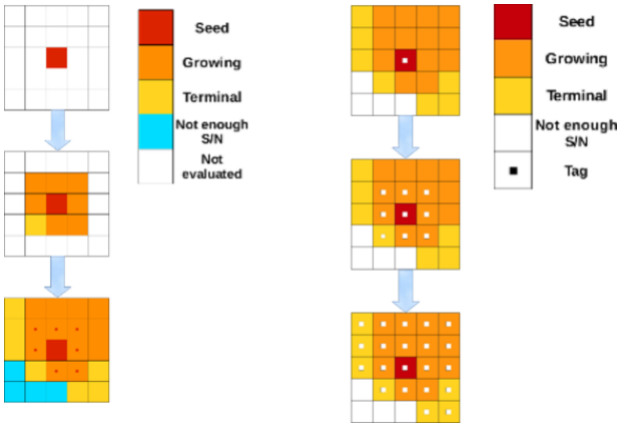


FIGURE 6.5: Calorimeter clustering algorithm. Left: GPU implementation, right: CPU implementation [34].

6.4 Analysis of GPU Demonstrator Results

The results presented in this section were recorded by the GPU demonstrator group, and should be considered preliminary as code is still under development.

6.4.1 ID Tracking Results

The 26x speed-up shown in figure 6.6 is for an Nvidia Tesla C2050 GPU with respect to an Intel E5620 CPU. The speed-up is for the first two steps of ID tracking as per section 6.3.1, i.e. for bytestream decoding and clustering. It should be noted that the CPU in this test is running only a single thread, which is not a particularly realistic scenario for a CPU in the HLT trigger farm, which operate with multithreading. In light of this the relative speed-up is closer to 7x. It can be seen that the speed-up increases with increasing RoI size, i.e. increasing data volume. This reflects the trend of GPUs to exhibit greater speed-up for larger volumes of data, as discussed in section 3.1.3 and illustrated experimentally in section 4.4. Figure 6.7 illustrates the relative speed-up for the same GPU-CPU pairing as above, however for steps 3 and 4 of tracking shown in section 6.3.1; track formation and clone removal. Interesting to note is the gradation in speed-up from processing both steps on CPU, to just clone removal on CPU, and finally both steps on GPU. The speed-up for the last case is about 12x, however this is again for a single-thread on the CPU. Note that the x-axis represents number of spacepoints as opposed to input data volume as in figure 6.6, however the two parameters increase in proportion to each other. Figure 6.8 displays the results for the entire ID tracking process. Additionally it displays the results

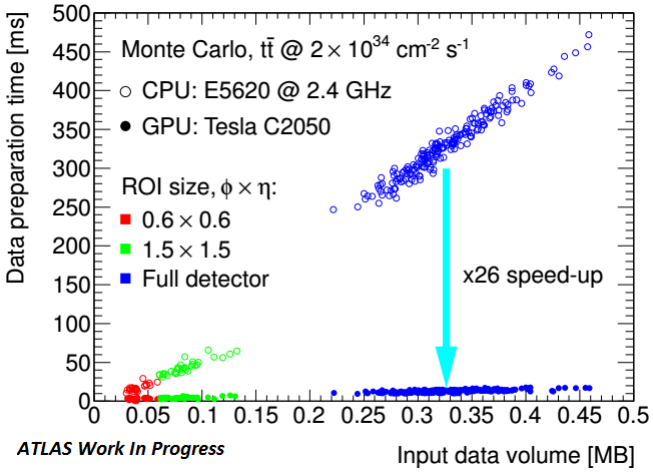


FIGURE 6.6: Decoding and clustering time versus data volume for different ROI sizes in GPU and CPU implementations [35].

for one GPU relative to one CPU, up to four GPUs working in tandem relative to one CPU. Note the x-axis is number of concurrently running Athena threads, again related to input data volume. At around 50 Athena threads, global GPU memory becomes a bottleneck, and one GPU performs worse than one CPU. Using two or more GPUs in tandem alleviates this issue, however it can be seen that continuing to add GPUs does not significantly reduce execution time. These results, as well as the remaining results presented in this section are from a system consisting of two Intel Xeon E5-2695 v3 14-core CPUs with a clock speed of 2.3 GHz and Nvidia GK210GL GPUs in a Tesla K80 module. Figure 6.9 details the break down of time spent on various processes for CPU and GPU processing. The Athena portions of the pie charts refer to all the non-algorithmic, administrative processes such as data reception from the Level-1 trigger. For track seeding on GPU, the algorithmic portion occupies a small part of the total processing time relative to the time breakdown on CPU. The execution of the track seeding algorithms on GPU are 5x times faster than on CPU, the entire process however is about 1.3x faster. Important to note is that the conversions of data between GPU and CPU formats, the transfer of data between GPU and CPU, as well as the Inter-Process Communication (IPC) overheads make up about 13% of total processing time. This is a significant amount of time spent on overheads associated with running the ID tracking on GPU.

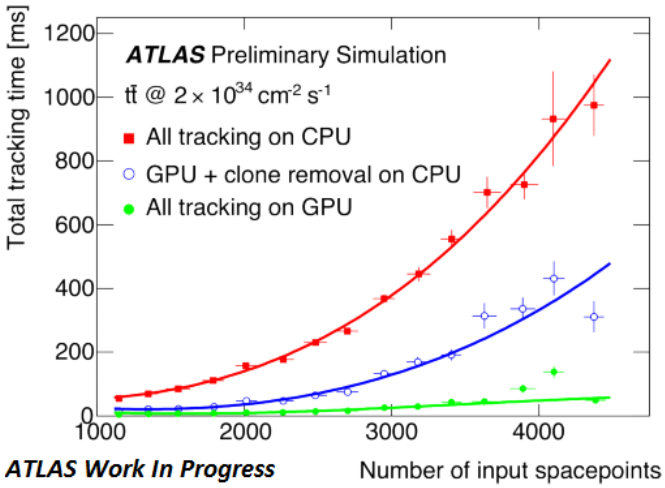


FIGURE 6.7: Tracking and clone removal time versus number of spacepoints in GPU and CPU implementations [35].

6.4.2 Calorimeter Clustering Results

The results of the calorimeter clustering algorithm on GPU show poor speed-ups, however the GPU implementation is still new and not optimised. The speed-up for the calorimeter clustering algorithm shown in figure 6.10 ranges from about 2x to 2.5x for the range of concurrently running ATHENA threads tested. It is interesting to note that the global GPU memory does not impose a bottleneck for this algorithm, as it can be seen that using more than one GPU concurrently does not lead to greater speed-ups. Figure 6.11 shows the time breakdown for the algorithm. Similar to ID tracking the algorithm itself occupies only a small portion of the total time. Of the 4.1% of the time spent on the calorimeter clustering algorithm, 45.7% of that time is spent on communication and conversion overheads. The speed-up associated with the algorithm is about 2x, however the time for the entire GPU implementation is in fact slower than for CPU, about 1.03x times slower.

6.4.3 Combined ID Tracking and Calorimeter Clustering

When running the ID and Calorimeter algorithms concurrently, they effect one another's performance. Figure 6.12 indicates the performance of the algorithms in events/second, running alone and simultaneously, on CPU and one and two GPUs. Using two GPUs only provides a speed-up for ID alone, and for ID and calorimeter together for high numbers of Athena processes. ID tracking shows the largest speed-up, while calorimeter tracking and the two algorithms together show a very small speed-up. The two algorithms

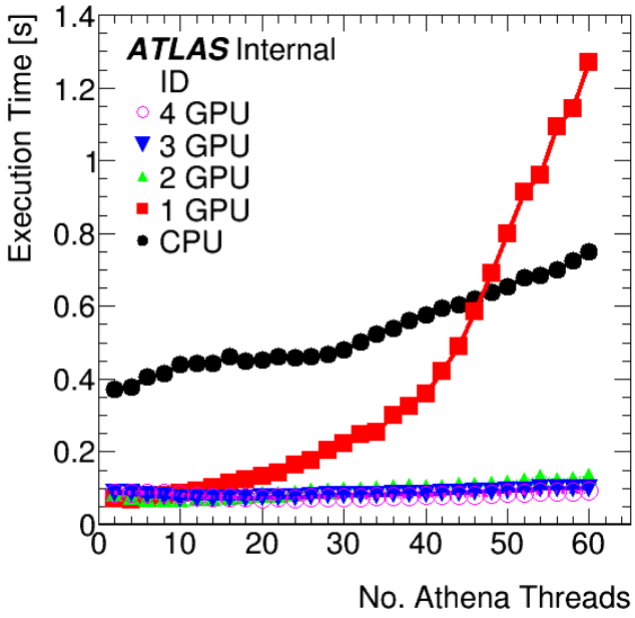


FIGURE 6.8: Overall ID tracking execution time as a function of the number of Athena threads for GPU and CPU implementations [37].

together operate at a substantially smaller rate than either algorithms in isolation, probably due again to a memory bottleneck.

Comparing figures 6.10 and 6.13, the calorimeter algorithm shows a similar performance when run alone and together with ID tracking, however only when using two or more GPUs. Comparing figures 6.8 and 6.14, ID tracking exhibits almost identical behaviour when running alone and with calorimeter tracking.

6.5 Critical Analysis

The preliminary results of the speed-up due to implementation of ID tracking and calorimeter clustering on one or more GPUs are underwhelming. The ID tracking algorithm running in isolation shows a speed-up of approximately 5x with respect to CPU implementation, but the inclusion of the overheads related to ATHENA reduce this to approximately 1.3x. The results for the calorimeter clustering algorithm are less attractive still, with a 2x speed-up for the algorithm in isolation and a slowing-down of 1.03x when ATHENA overheads are taken into account. However as the calorimeter algorithm is still immature compared to ID tracking, improvements can

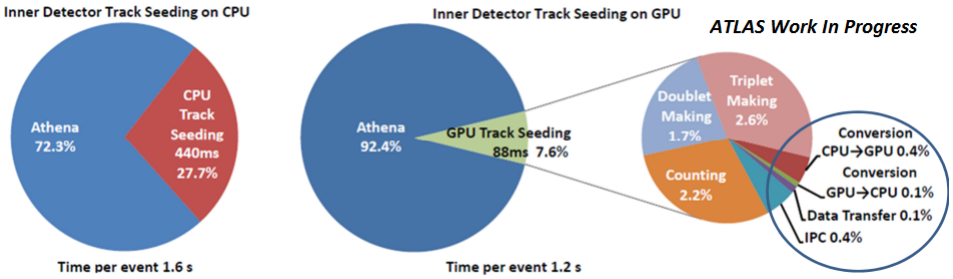


FIGURE 6.9: Time breakdown of ID tracking on CPU and GPU [38].

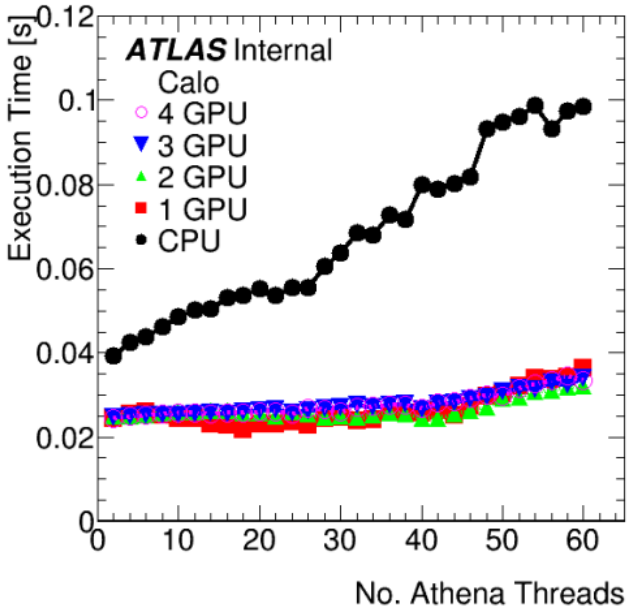


FIGURE 6.10: Overall calorimeter clustering execution time versus number of Athena threads for GPU and CPU implementations [37].

be expected as the code is optimised. The same can be said of a more mature ID tracking algorithm.

The following parameters are approximated by the GPU demonstrator group. To break even in terms of cost-to-benefit ratio the speed-up required is 4x for a GPU:CPU ratio of 1:2 when using server-grade GPUs. The required speed-up to break even drops to 2x when using commodity-grade GPUs intended for gaming in a 1:1 GPU:CPU ratio. These speed-ups refer to overall throughput of the HLT algorithm. Of course, breaking even would not be sufficient cause to overhaul the software and hardware of the HLT. In the case of gaming GPUs, a 4x speed-up (break even+100%) would

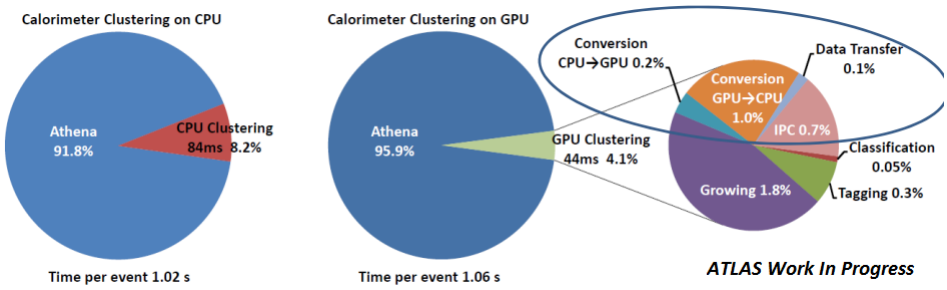


FIGURE 6.11: Time breakdown of calorimeter clustering on CPU and GPU [38].

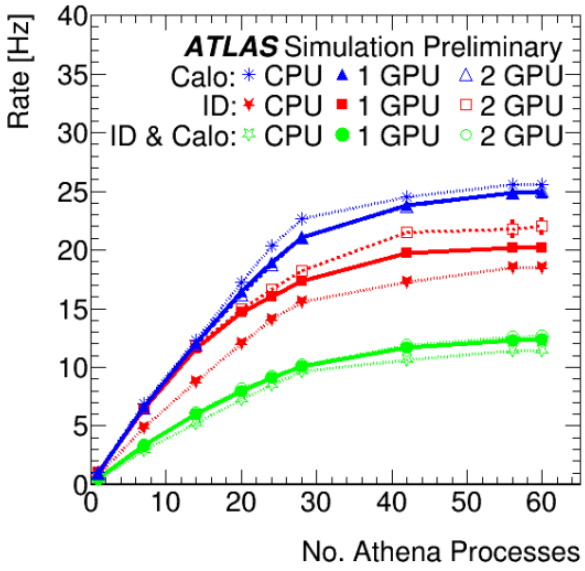


FIGURE 6.12: Speed-up in events/second for ID tracking and calorimeter clustering separately and simultaneously [38].

imply a halving of the cost of the current HLT platform, and this is the figure chosen for minimum viability of introduction of GPU into the HLT. The use of gaming GPUs presents an additional issue of reliability in an environment as critical as the HLT servers; as these commodity GPUs are typically not as reliable as their server-grade counterparts.

6.5.1 Necessity of Improved Algorithms

The speed-ups required for viability mentioned above are with reference to ID tracking and calorimeter algorithms which mimic those already in use in the HLT. The implementation of these algorithms for the purposes of the GPU demonstrator was chosen to provide a basis for comparison and the

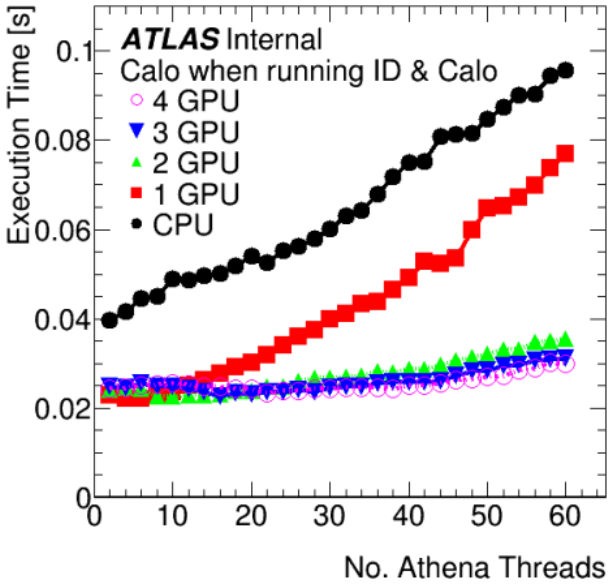


FIGURE 6.13: Performance of calorimeter clustering running together with ID tracking [37].

ability to check the correctness of the results of the algorithms. However, although these algorithms have been made parallel where possible, they do not necessarily represent the algorithms best suited for GPU implementation. The algorithms upon which they are based were chosen and designed exclusively for serial implementation on CPU; as at the time of their design GPGPU was not a viable option. A further speed-up could be expected if the algorithms of the HLT are from their conception designed for massive parallelism. This type of redesign is necessary in terms of allowing for the greatest speed-up but it is also necessary in terms of introducing algorithms with increased complexity. Increased complexity has two implications of note: the algorithms are more adept at detecting events of interest in the face of increased pile-up, and usually they are more computationally intense and hence higher latency. Complexity, in this context, essentially refers to the number of operations comprising the algorithm, and how the number of these operations grow as a function of input size. They may, for instance, grow in proportion to the square or the cube of the input to the algorithm. If the same or similar algorithms to those currently in use in the HLT were retained for the HL-LHC, however sped up they are, this would not imply the possibility of increasing integrated luminosity of events of interest by searching for event signatures currently not feasible to search for, or the ability to mitigate the effects of high pile-up environments. It would allow only for the same results currently achievable, to be achieved with higher

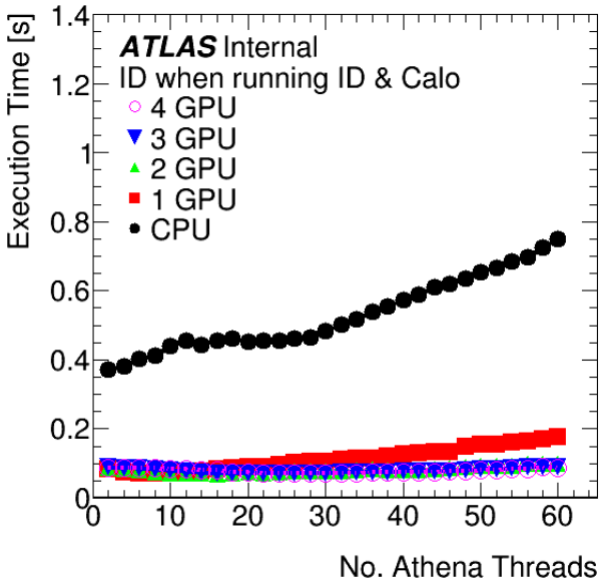


FIGURE 6.14: Performance of ID tracking running together with calorimeter clustering [37].

throughput. Algorithms of higher complexity are therefore necessary, but their higher latency has implications for the break-even analysis described above. Higher latency implies decreased speed-up with respect to current algorithms, and therefore lower throughput. The problem that must be addressed from the present to the inception of the HL-LHC is therefore the problem of designing algorithms of higher complexity capable of minimising the problems of high pile-up while at the same time remaining capable of producing the necessary throughput rates. The reader interested in what such an algorithm may look like can refer to [39] for a novel tracking algorithm based on the Hough transform.

6.5.2 Co-PU in HLT

It is possible that the HLT could benefit from the inclusion of a low-cost, energy-efficient co-processor such as that described in section 5.2.2. The attractive cost and energy parameters are due to the open-source ARM architecture of the GPUs in question. The Tegra K1 described in chapter 4, for instance, operates at 11 Watts, compared to the Tesla K80 used in the demonstrator that operates at about 300 Watts. The prices of the two are 200 USD and 5000 USD respectively. Of course, these differences are a result of the greater computational ability of the K80, but the affordability of the K1 would allow it to be used in clusters while still being relatively cheap and

energy efficient. However since they are a relatively new platform, there is a deficiency in support documentation and compatible software.

Porting APE to ARM Architecture

The author tested the viability of porting APE to ARM architecture, on a Tegra K1 development board. The problem encountered is that APE was developed for a Red Hat Linux platform, while the Tegra K1 supports only Ubuntu natively. The author was able to install a Red Hat variant, Fedora, on the Tegra K1 with some difficulty. In this case it is possible to install an APE server on the Tegra, but the problem then becomes that at this time CUDA is not supported on ARM-Fedora platforms. An attempt was made to adapt the available Ubuntu CUDA package for the TK1 to run on Fedora with some success, however this scenario would not be conducive to the most efficient performance from the TK1.

Chapter 7

Conclusion

In this dissertation GPGPU technology has been discussed with reference to its integration into the trigger system of the ATLAS detector in the LHC at CERN. The sub-detectors comprising ATLAS - the inner detector, the calorimeters, and the muon spectrometer - have been described. The principle of their operation as well as their purpose was outlined. These sub-detectors work in tandem to detect event signatures of interest. The readings from these detectors are integrated and analysed in real-time by a multistage trigger system, the details of which have also been described in chapter 2 and in more detail in chapters 5 and 6.

The LHC is undergoing upgrades which will see the ATLAS detector subjected to higher instantaneous luminosities, leading to increased pile-up and trigger rates. Consequently, the software and hardware platforms of the TDAQ system must also be upgraded in order to ensure that physics results remain accurate, and that the higher luminosity of the LHC can be fully exploited in the search for new physics.

It is this upgrade that forms the motivation for researching the viability of using GPGPU in the Trigger. This dissertation has presented a brief history of GPGPU, its principles of physical operation, the software that it runs, and the type of problems it deals with efficiently. GPGPU is a relatively new technology being used in a wide range of sciences to provide greater performance relative to CPU technology in problems of computation. The performance improvements are due to the massive parallelism of GPGPU.

7.1 GPU Performance Parameters

Chapter 4 presents results of benchmark tests conducted by the author. The results are intended to demonstrate the concepts introduced in chapter 3, as well as to demonstrate the range of performance of GPUs available commercially. In particular, the results demonstrate that when attempting to speed-up a computing problem with GPU, the factors affecting performance are

varied. The nature of the problem dictates, to some extent, which GPU is most appropriate for use. However cost, space, and compatibility limit the choices available; meaning code structure must also be informed by GPU choice. Further convoluting the issue is that GPU vendors release information about their products to present them in the most attractive light, and significant resources must be dedicated to characterising the performance of the GPU in a way which simulates the actual environment in which it will operate.

7.2 GPGPU in the Level-0/1 Trigger

Chapter 5 explores the introduction of GPGPU into the Level-0/1 Trigger. First the platform currently in use is described in greater detail. FPGAs are used extensively despite their high cost, high energy consumption, and difficulty of use. This is because of the extreme latency constraints on this stage of the Trigger requiring the type of performance available from hardware based technology. The level-0/1 Trigger upgrade provides an opportunity to consider the involvement of GPGPU, the questions raised in chapter 5 are how could GPUs function in such a low-latency environment, and what benefits could they offer. Because they operate at latencies far greater than that of the Level-0/1 Trigger they can neither receive nor process information fast enough to operate synchronously with the Trigger. To be used at this stage then, they would need to operate in parallel with the Trigger. The communication of data between GPU and the Trigger would need to be mediated by a CPU for the Trigger rate to be unhindered. It is argued that the low-cost, low-energy ARM technology CPU could be used in conjunction with GPU to form a cost and energy efficient Co-Processing Unit that could run parallel to the Trigger without disrupting it.

To answer the question of whether the Trigger could benefit from such a Co-PU, some potential roles for the Co-PU are suggested. It could ease and expedite the process of testing and implementing new Trigger algorithms - for instance enhanced OF algorithms and reducing Trigger dead time.

Although it seems possible to use GPU from a technical stand-point, and also that there are roles GPUs could fulfill in the Level-0/1 Trigger, it remains unclear whether there is material benefit to gain from massive parallelism at this level. As the Co-PU would need to operate asynchronously, it seems unlikely that any speed-up to be gained would be of much consequence. Thus, benefits regarding ease of use in implementing new algorithms might be more easily gained from CPU in isolation. As the nature

of problems that would be implemented on such a Co-PU become more defined, it will be possible to say with more confidence whether the inclusion of GPGPU would be beneficial.

7.3 GPGPU in the HLT Trigger

The role of GPGPU in the HLT, as discussed in chapter 6, has been researched to a greater extent than in the low level Trigger, by the GPU demonstrator group. Initially efforts were made to convert Athena code directly to CUDA code for GPU implementation. This proved time-consuming and cumbersome, and was abandoned in favour of the creation of an intermediary software to facilitate the interaction of Athena and GPU. This intermediary software, APE, allows HLT Athena processes to query one or more GPUs, and to receive results from GPUs in the format it expects. APE oversees the conversion of data structures from Athena to GPU formats, and these conversion processes introduce additional latencies to the event processing.

The GPU demonstrator group has tested the performance of two HLT algorithms on GPU: inner detector tracking, and calorimeter clustering. Chapter 6 presents, compares, and analyses results collated from several documents ranging from 2012-2016. The execution of the ID tracking algorithms on GPU are 5x times faster than on CPU, when including APE overheads this reduces to 1.3x faster. The speed-up of the calorimeter clustering algorithms is about 2x, however when including APE overheads it is in fact slower than for CPU, about 1.03x times slower. These speed-ups do not yet meet the goal of a 4x overall speed-up specified by the demonstrator group as the improvement which would allow for a 50% reduction in the cost of the HLT computing system.

Greater speed-ups are expected as GPU technology improves, and as algorithms are designed specifically for parallel implementation, as opposed to mimicking the current serial algorithms in use. These undecided algorithms must be able to outperform current algorithms in their ability to deal with increased pile-up and to detect more complex event signatures.

The author attempted to port APE to ARM architecture, based on the premise that the HLT Trigger could benefit from a low-cost, low-energy Co-PU of the type described in chapter 5. Although possible, it is both difficult and not conducive to achieving the best performance from such a Co-PU.

7.4 Future Work

7.4.1 In the Level-0/1

In order to determine whether the inclusion of an ARM-GPU based Co-PU would be beneficial in the Level-0/1 Trigger, future work should focus on determining the size of the problems likely to be offloaded to the Co-PU, and the time constraints of the delivery of associated results.

7.4.2 In the HLT

The GPU demonstrator group has already begun testing existing parallelised algorithms on newer GPU platforms in order to measure any increase in speed-ups. However research should also be done on developing entirely new and more robust algorithms, as well as attempting to eliminate overheads associated with APE and IPCs in general.

References

[1] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08003.

[2] V. Halyo et al. "GPU Enhancement of the Trigger to Extend Physics Reach at the LHC". In: *Journal of Instrumentation* 8.08 (Oct. 2013), P10005.

[3] J. D. Owens et al. "GPU Computing". In: *Proceedings of the IEEE* 96.5 (Apr. 2008), pp. 879–899. URL: http://ieeexplore.ieee.org/document/4490127/.

[4] CERN. *Overall view of the LHC*. 2016. URL: http://cds.cern.ch/record/1708847/. Accessed December 2016.

[5] A. R. Martinez on behalf of the ATLAS Collaboration. *The Run-2 ATLAS Trigger System*. CERN CDS. 2016. URL: https://cds.cern.ch/record/2133909/files/ATL-DAQ-PROC-2016-003.pdf. Accessed February 2017.

[6] Matteo Bauce et al. "Use of hardware accelerators for ATLAS computing". In: *Proceedings, GPU Computing in High-Energy Physics* (2015), pp. 48–54. DOI: 10.3204/DESY-PROC-2014-05/10.

[7] P. Klimek on behalf of the ATLAS Tile Calorimeter group. "Signal reconstruction performance with the ATLAS Hadronic Tile Calorimeter". In: *J. Phys.: Conf. Ser.* 404 (2012), p. 012046.

[8] L. M. de A. Filho et al. *Calorimeter Signal Response Deconvolution for Online Energy Estimation in Presence of Pile-up*. CERN CDS. 2012.

[9] CERN. *About CERN*. 2012. URL: http://cds.cern.ch/record/1997225. Accessed December 2016.

[10] CERN. *The Large Hadron Collider*. 2014. URL: http://cds.cern.ch/record/1998498. Accessed December 2016.

[11] ATLAS Experiment ©2016 CERN. *A detailed computer-generated image of the ATLAS detector and it's systems.* 2016. URL: http://atlasexperiment.org/photos/full-detector-cgi.html. Accessed February 2017.

[12] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (Aug. 2008), pp. 53–109.

[13] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (Aug. 2008), pp. 110–163.

[14] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (Aug. 2008), pp. 164–205.

[15] The ATLAS Collaboration. "The ATLAS Experiment at the CERN Large Hadron Collider". In: *Journal of Instrumentation* 3.08 (Aug. 2008), pp. 6–25.

[16] G. Golster. "ATLAS Trigger: Preparations for Run II". MA thesis. CERN / The Niels Bohr Institute, June 2015.

[17] Yu Nakahama. "The ATLAS Trigger System: Ready for Run-2". In: *Journal of Physics: Conference Series* 664.8 (2015), p. 082037. URL: http://stacks.iop.org/1742-6596/664/i=8/a=082037.

[18] A. Valero on behalf of the ATLAS Collaboration. *A new read-out architecture for the ATLAS Tile Calorimeter Phase-II Upgrade*. CERN CDS. 2015. URL: https://cds.cern.ch/record/2117093/files/ATL-TILECAL-PROC-2015-025.pdf. Accessed February 2017.

[19] The ATLAS Collaboration. *Atlas TDAQ Software Projects and Tools*. 2017. URL: https://gitlab.cern.ch/atlas-tdaq-software.

[20] J. D. Owens et al. "A Survey of General-Purpose Computation on Graphics Hardware". In: *Computer Graphics Forum* 26.1 (Mar. 2007), pp. 80–113. URL: https://www.researchgate.net/publication/227633811_A_Survey_of_General-Purpose_Computation_on_Graphics_Hardware.

[21] Robert G. Brown. *Amdahl's Law and Parallel Speedup*. Aug. 2008. URL: http://www.phy.duke.edu/~rgb/brahma/brahma_old/als/als/node3.html. Accessed December 2016.

[22] NVIDIA Corporation. *CUDA*. 2016. URL: http://www.nvidia.com/object/cuda_home_new.html. Accessed December 2016.

[23] NVIDIA Corporation. *CUDA C Programming Guide*. 2016. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz4TGcIMwea. Accessed December 2016.

[24] *NVIDIA Tegra K1 Embedded Platform Design Guide*. NVIDIA Corporation, 2015. URL: http://developer.download.nvidia.com/embedded/jetson/TK1/docs/3_HWDesignDev/TegraK1_Embedded_DG_v03.pdf. Accessed February 2017.

[25] *GeForce 840M*. NVIDIA Corporation, 2017. URL: http://www.geforce.com/hardware/notebook-gpus/geforce-840m/specifications. Accessed February 2017.

[26] *TESLA K80 GPU ACCELERATOR*. NVIDIA Corporation, 2015. URL: https://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf. Accessed February 2017.

[27] *Intel® Core™ i7-5500U Processor*. Intel Corporation, 2017. URL: https://ark.intel.com/products/85214/Intel-Core-i7-5500U-Processor-4M-Cache-up-to-3_00-GHz. Accessed February 2017.

[28] J. Pettersson. *Fast GPU based reduction sum*. 2015. URL: http://pastebin.com/sZCwbHVH. Accessed February 2017.

[29] Xilinx Inc. *Field Programmable Gate Array (FPGA)*. 2017. URL: https://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm. Accessed February 2017.

[30] M. A. Cox. "Energy Reconstruction on the LHC ATLAS TileCal Upgraded Front End: Feasibility study for a sROD Co-Processing Unit". MA thesis. University of the Witwatersrand, Oct. 2015.

[31] PCI-SIG. *Specifications Library*. 2017. URL: https://pcisig.com/specifications.

[32] J M Seixas and ATLAS Tile Calorimeter System. "Quality Factor for the Hadronic Calorimeter in High Luminosity Conditions". In: *Journal of Physics: Conference Series* 608.1 (2015), p. 012044. URL: http://stacks.iop.org/1742-6596/608/i=1/a=012044.

[33] Collaboration ATLAS. *Letter of Intent for the Phase-II Upgrade of the ATLAS Experiment*. Tech. rep. CERN-LHCC-2012-022. LHCC-I-023. Geneva: CERN, Dec. 2012. URL: https://cds.cern.ch/record/1502664.

[34] S Kama et al. "Triggering events with GPUs at ATLAS". In: *Journal of Physics: Conference Series* 664.9 (2015), p. 092014. URL: http://stacks.iop.org/1742-6596/664/i=9/a=092014.

[35] M. Bauce et al. "Use of hardware accelerators for ATLAS computing". In: *Proceedings, GPU Computing in High-Energy Physics (GPUHEP2014): Pisa, Italy, September 10-12, 2014*. 2015, pp. 48–54. DOI: 10.3204/DESY-PROC-2014-05/10. URL: http://inspirehep.net/record/1386621/files/10.pdf.

[36] D. Emeliyanov on behalf of the HLT GPU demonstrator group. *HLT GPU demonstrator project*. Indico. 2016. URL: https://indico.cern.ch/event/438204/contributions/1939229/attachments/1235352/1813380/HLT_GPU_demonstrator_project_SC_Week_01_03_2016.pdf#search=HLT%20GPU%20demonstrator%20project. Accessed April 2017.

[37] J. Baines. *Trigger GPU Demonstrator Update*. Indico. 2016. URL: https://indico.cern.ch/event/575539/. Accessed April 2017.

[38] J. Baines and T. Bold. *ATLAS Trigger GPU Demonstrator Performance Plots*. CERN CDS. 2016. URL: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/TriggerSoftwareUpgradePublicResults. Accessed April 2017.

[39] V. Halyo et al. "GPU enhancement of the trigger to extend physics reach at the LHC". In: *Journal of Instrumentation* 8.10 (2013), P10005. URL: http://stacks.iop.org/1748-0221/8/i=10/a=P10005.