|epcc|

GPUs and the ATLAS experiment - Project Plan

Christopher Jones

March 25, 2010

MSc in High Performance Computing

The University of Edinburgh

Year of Presentation: 2010

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 ATLAS

ATLAS (A Toroidal LHC ApparatuS [1]) is one of the main experiments in the Large Hadron Collider (LHC) found at CERN (European Organisation for Nuclear Research), near Geneva, Switzerland. One of two general-purpose detectors in the LHC, ATLAS possesses a wide range of search channels for evidence of new fundamental particles. The main focus of the search is the Higgs boson, the last remaining undetected particle in the Standard Model of particle physics. There is also potential for physics beyond the Standard Model, such as the detection of supersymmetric particles, and progress towards a Grand Unified Theory; the combination of the electromagnetic, weak, and strong interactions into a single theory.

The bunch crossing rate[1] inside the detector is 40 MHz, with there being $\sim$20 collisions per crossing [3]. With around a billion collisions occurring every second, there is far too much data produced to store in its entirety. The trigger is responsible for selecting out the events of potential scientific interest, which can then be stored for further processing.

The trigger works in three separate stages, each of which rejects a significant fraction of the events, as seen in Figure 1.2. Level 1 runs on customised hardware; Levels 2 and 3 in software on a server farm. Overall, the trigger reduces the total number of events output by a factor of around 200,000.

[1] Protons travel around the collider in bunches, each containing 10^{11} particles [3].

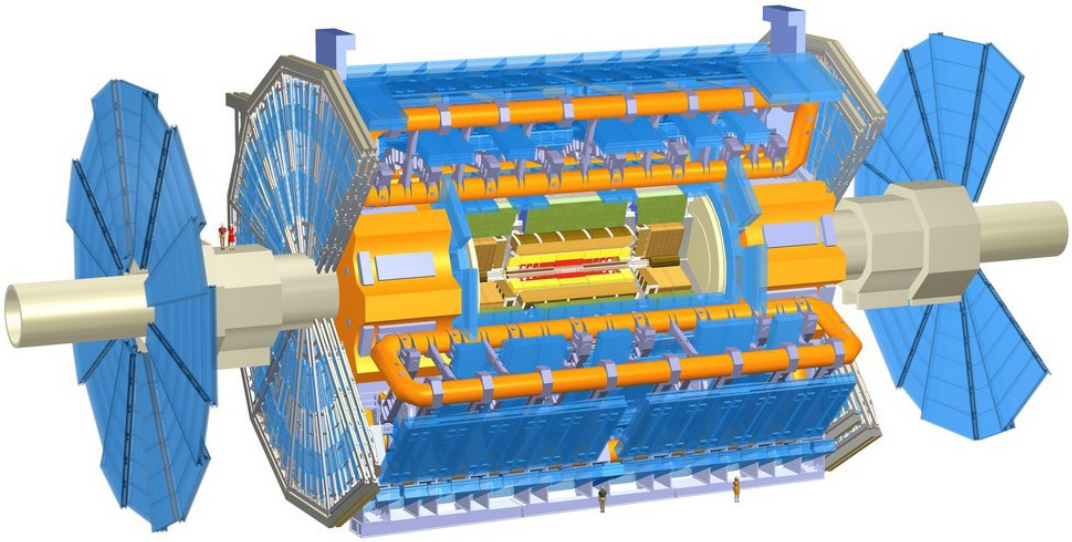

Figure 1.1: The ATLAS experiment [2]. The inner detector is coloured yellow.

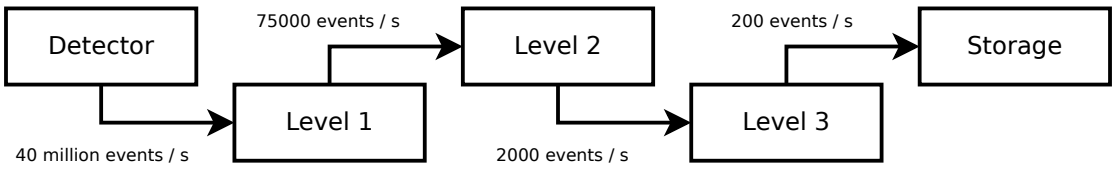

Figure 1.2: Schematic of the ATLAS trigger.

The z-finder algorithm

Level 2 of the trigger runs many different programs which operate on different parts of the detector output. One of these, IDScan, is a track-reconstruction package taking input from the inner detector [4]. IDScan employs a series of algorithms of which the z-finder algorithm, defined in [5], is the first to execute.

The purpose of this algorithm is to produce a good estimate for the z-coordinate of the collision event along the axis of the detector, z_V. This may vary by up to 15 cm from the centre. Acquiring the position of the interaction is necessary for reducing the execution time of the track reconstruction algorithms later in the chain.

Due to the geometry of the detector, it is most natural to perform the calculation in cylindrical polar coordinates; the z-axis of the coordinate system aligned with the axis of the detector and thus also the magnetic field. The space points[2] are divided into slices in the ϕ direction. Within a given slice and its two neighbouring slices, every combination of pairs from different layers of the detector are used to generate a z-coordinate. This is performed by a linear extrapolation of the trajectory through the two points back to the beam axis. The resulting z-coordinates are placed in an appropriate bin of the z-histogram. One such histogram can be seen in Figure 1.3.

False pairings of points will just add random noise to the histogram. However, pairs of points representing real particle trajectories should produce a consistent value for z_V, as seen by a peak in the histogram. This peak is taken as the true value of z_V.

We can operate on the separate slices of points independently, making this algorithm 'embarrassingly parallel', to the first approximation. Several parameters, such as the number of ϕ slices may be tuned.

1.1.2 GPGPU

In the last few years, general-purpose computing on graphics processing units (GPGPU) has become a major research area in high performance computing (HPC), due to dramatic speed-up available to suitable applications. The rapid increase in the performance of GPUs, driven primarily by the games industry, has resulted in many personal computers containing a powerful, low-cost computing resource, with peak floating-point performance well in excess of that of the CPU.

The Compute Unified Device Architecture (CUDA [6]) is the propriety computing engine for GPUs from manufacturer Nvidia. CUDA is the most mature GPGPU architecture and, at present, the most widely used. It provides a fairly low-level mapping of hardware features, but is relatively simple to program in the C for CUDA language [7], which is an extension to the C99 dialect of the C programming language.

[2] Space points are those locations in the detector which have registered a hit.

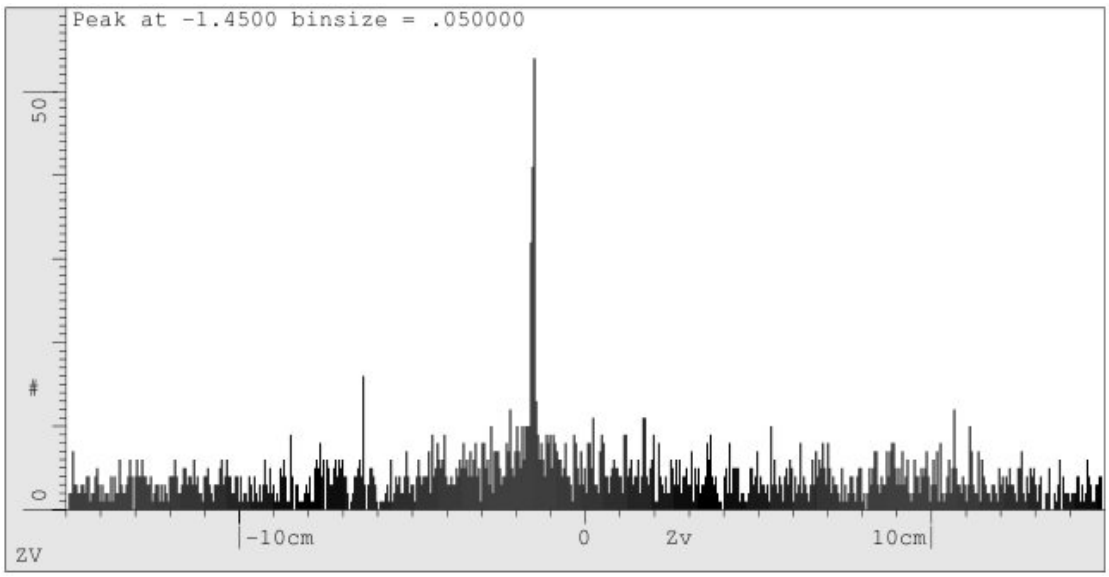

Figure 1.3: A z-histogram for a low luminosity event, using pairs of space points [5].

1.2 Motivation

This project is a feasibility study of the usage of GPUs for accelerating trigger algorithms in the next major upgrade of the LHC. This upgrade will see the luminosity of the beam boosted by around an order of magnitude to increase the chance of seeing rare events, which will correspondingly raise the workload on the trigger.

The z-finder algorithm will be used as our example trigger algorithm. The independent ϕ slices should map well onto individual CUDA blocks. Typically, around 1000 slices are used [5], giving sufficient parallelism for GPU devices. The code provides the opportunity to explore a wide range of optimisation strategies, exploiting the architecture to the full.

The key to the trigger is high throughput, making this an interesting departure from typical HPC applications, in which programs generally run for considerable amounts of time. Considerable thought will have to been given to achieving this target.

From this investigation, it should be possible to determine if GPU accelerated programs are capable of providing the throughput required of the trigger in the next-generation LHC. It will also indicate whether or not significant modifications to the existing framework will be required to achieve this.

Should it become apparent that GPUs are unable to provide the necessary speed-up, the ATLAS collaboration would have to pursue more demanding alternatives, such as moving the Level 2 trigger onto bespoke hardware.

Chapter 2

Previous Dissertation Review

2.1 Paul Woodhams, 2008

The default behaviour of MPI programs is to crash upon encountering a fault. Woodhams' project entitled 'Investigation of Fault Tolerance in MPI Applications' [8] aims to create a fault tolerant library. Fault tolerance becomes an ever more pressing issue with the expanding size of HPC systems.

A fault simulation library was created to simulate real failure events, generating random errors. All processes were made aware that a process had 'died' using a global memory, stored on the master process, accessed by single-sided communications. The 'dead' process was then removed from the global communicator. Two typical example applications were tested: a task farm and an image processor. The latter, using a regular domain decomposition, required frequent disk checkpoints to recover their state after a failure, adding significantly to the overhead.

The project was successful in some aspects and the task farm application worked consistently, even with multiple failures. In the image processing code, the use of single-sided communications introduced some synchronisation issues not present in the regular MPI code, as this is usually implicit in the communication. These issues were not entirely solved in the available time, but some attempt was made to explain them in the report. As this is clearly the more difficult of the two use cases, it would have been good to see more effort concentrated here.

There are a lot of diagrams in the report which aid the understanding of the prose. In many cases these diagrams are redundant, as the writing is sufficiently clear, but they do help to break up the page and make it the report more visually appealing.

2.2 Alan Richardson, 2009

Richardson's project, 'GPU Acceleration of HPC Applications' [9], investigated the potential for speed-up in five existing HPC applications through porting them to GPU using Nvidia's C for CUDA language. Of these five candidate codes, three were considered suitable to be ported.

The results were mixed, with ported codes in some instances failing even to match the performance of the original CPU code. After optimisation, two of the codes did achieve good speed-up of kernel execution, though the overall speed-up was reduced considerably when memory transfer times were factored in.

Despite the explanations of the CUDA architecture being clear and extremely thorough, they are rather dry. Omitting some of the detail may serve to emphasise the most important points and make for more interesting reading. The section on porting the codes strikes a better balance, however, and the narrative is good. Many of the graphs and diagrams could look more professional.

Chapter 3

Work Plan

As already mentioned, the aim of the project is to port the z-finder algorithm of the IDScan program to CUDA.

Though a standalone version of IDScan exists, it makes many assumptions about its running environment and the presence of certain dependencies, so it is not trivial to build and run. The first stage of the project will be to get the serial code running correctly on the necessary machines; primarily `ness`. Once this has been performed, the serial code can be benchmarked and profiled.

The first step in porting the code to GPU will be to isolate the main computational kernel of the serial code and encapsulate it in a function. This kernel may then be ported into C for CUDA and the necessary CUDA wrapper code added.

A testing suite will be created at this stage to ensure that neither the porting process nor any future optimisations change the functionality of the code.

The bulk of the project will likely be optimising the code for the GPU architecture. This may include a few serial optimisations, but the main focus will be GPU-specific changes. Programming for this hardware requires many special considerations unnecessary on CPUs; for example, memory accesses must be carefully structured. Overlapping memory transfers and calculations will also be investigated. The code will be benchmarked at every stage of the optimisation process to quantify the effects of each modification.

Time permitting, the code will be executed on the GPU cluster at Daresbury to give an indication of the performance which might be achieved in a real-world. This may require writing a simple task-farm using the MPI library.

The last month of the plan is set-aside to the completion of the dissertation. Along with write-up itself, this will include the production of the necessary graphs and diagrams. Finally, the presentation will be prepared.

The work plan, including estimates of the time required for each step, may be seen in Figure 3.1.

Name	Work
Get original code to function correctly as a standalone executable	7d
Benchmarking / profiling serial code	7d
Port code to GPU	**21d**
Isolate computational kernel(s)	7d
Port kernel(s) to GPU	7d
Write correctness tests for GPU code	7d
Benchmarking / profiling GPU code	42d
Optimization of GPU code	**35d**
Serial optimizations	7d
GPU specific optimizations	21d
Overlapping memory transfers and calculations	7d
Write MPI task farm for GPU cluster	7d
Dissertation	**28d**
Create diagrams and graphs	7d
Write-up	21d
Presentation	7d

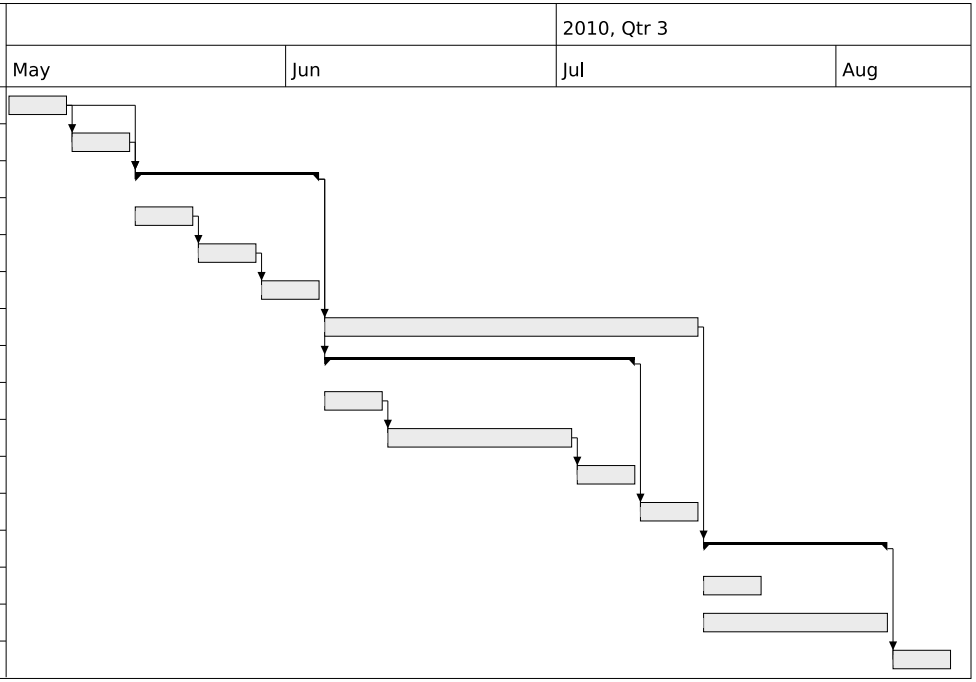

Figure 3.1: Gantt chart of project work plan.

Chapter 4

Preliminary Investigations

I have completed the C for CUDA exercises and written several test programs for GPUs, such as a Mandelbrot set generator. I have read extensively about the device hardware, and have a solid knowledge of the memory architecture, the thread scheduling system, and various other idiosyncrasies of GPU architecture.

A CUDA program is only able to make use of a single device at a given time. Therefore, to fully utilise a machine with multiple GPUs, an application must be multi-threaded; for example, using OpenMP or MPI. Using a pre-existing OpenMP test [10], I was able to confirm that it is possible to use both devices installed on `ness` simultaneously.

I have completed an inspection of the existing C++ code to be ported, highlighting certain elements of the algorithm which might cause performance problems in a parallel environment; such as the requirement for atomic adds when constructing the z-histogram.

With some effort, it has been possible to build the IDScan program as a standalone executable. Suitable input files for testing the program have been acquired, and more can be created as needed. However, the program takes many input parameters, and one of the first tasks upon commencement of the project will be to ascertain which are most appropriate for our needs. Once this is done, benchmarking the program should be fairly trivial.

Chapter 5

Dissertation Outline

The dissertation will be structured roughly as follows. The ‘Optimisation’ chapter is likely to occupy a large proportion of the dissertation, and will therefore be broken into numerous sections.

1. Abstract
2. Introduction
3. Background
    (a) The ATLAS experiment
    (b) The CUDA architecture
4. Porting to CUDA
5. Testing
6. Optimisation
    (a) Serial optimisations
    (b) Memory optimisations
    (c) Program flow optimisations
    (d) Overlapping memory transfers and calculations
7. Performance on a GPU cluster
8. Conclusions
9. Bibliography

Chapter 6

Risk Analysis

Following is a selection of the risks identified in this project. All of the risks are deemed acceptable with the mitigation correctly invoked.

Risk	**Impact**	**Likelihood**	**Mitigation**
Difficulties in running the program standalone	Medium	Medium	Set aside time in work-plan for understanding and running the original code.
Algorithm impossible to parallelise on GPUs	V. High	Low	Thorough investigation of the code in advance of commencing project.
Failure to achieve improved performance on GPUs	Low	High	Give a clear explanation of the poor results and the reasoning behind them in the final report.
Unavailability of hardware for testing / benchmarking	Medium	Medium	Acquire access to several different systems to perform this activities.
Difficulty contacting supervisors	Medium	Medium	Contact supervisors with problems as soon as they occur, rather than letting them linger.
Falling out with supervisors	High	Low	Be polite and patient with supervisors at all times.
Long-term illness	High	V. Low	Eat well, take regular exercise.

Bibliography

[1] ATLAS Collaboration. ATLAS: technical proposal for a General-Purpose pp experiment at the LHC. *CERN/LHCC*, 94(43), 1994.

[2] ATLAS photos. http://atlas.ch/photos.

[3] ATLAS fact sheets. http://atlas.ch/fact_sheets.html.

[4] A. Gesualdi Mello et al. Overview of the high-level trigger electron and photon selection for the atlas experiment at the lhc. *Nuclear Science, IEEE Transactions on*, 53(5):2839 –2843, October 2006.

[5] N. Konstantinidis and H. Drevermann. Determination of the z position of primary interactions in ATLAS prior to track reconstruction. *ATLAS-DAQ-2002-014*, 2002.

[6] CUDA zone. http://www.nvidia.com/object/cuda_home.html.

[7] Nvidia Corporation. *CUDA Programming Guide*, 2.3 edition.

[8] Paul Woodhams. Investigation of fault tolerance in MPI applications. Master's thesis, EPCC, University of Edinburgh, 2008.

[9] Alan Richardson. GPU acceleration of HPC applications. Master's thesis, EPCC, University of Edinburgh, 2009.

[10] Yuan Wan. Use multiple GPU devices with OpenMP and CUDA. https://www.wiki.ed.ac.uk/display/ecdfwiki/Use+multiple+GPU+devices+with+OpenMP+and+CUDA.