Use of hardware accelerators for ATLAS computing

Matteo Bauce [2], *Rene Boeing* [3] [4], *Maik Dankel* [3] [4], *Jacob Howard* [1], *Sami Kama* [5] *for the ATLAS Collaboration*

[1]Department of Physics Oxford University, Oxford, United Kingdom
[2]Dipartimento di Fisica Sapienza Universit di Roma and INFN Sezione di Roma, Roma, Italy
[3]Fachbereich C Physik, Bergische Universitaet Wuppertal, Wuppertal, Germany
[4]Fachhochschule Muenster - University of Applied Sciences, Muenster, Germany
[5]Southern Methodist University, Dallas TX, US

DOI: http://dx.doi.org/10.3204/DESY-PROC-2014-05/10

Modern HEP experiments produce tremendous amounts of data. This data is processed by in-house built software frameworks which have lifetimes longer than the detector itself. Such frameworks were traditionally based on serial code and relied on advances in CPU technologies, mainly clock frequency, to cope with increasing data volumes. With the advent of many-core architectures and GPGPUs this paradigm has to shift to parallel processing and has to include the use of co-processors. However, since the design of most of the existing frameworks is based on the assumption of frequency scaling and predate co-processors, parallelisation and integration of co-processors are not an easy task. The ATLAS experiment is an example of such a big experiment with a big software framework called Athena. In this proceedings we will present studies on parallelisation and co-processor (GPGPU) use in data preparation and tracking for trigger and offline reconstruction as well as their integration into the multiple process based Athena framework using the Accelerator Process Extension APE.

1 Introduction

The Large Hadron Collider (LHC) is a 27km long circular particle accelerator near Geneva, situated about 100m below the Swiss and French border [2]. It is designed to collide proton bunches with a center-of-mass energy of 14TeV every 25ns. It is equipped with 4 detectors namely ALICE and LHCb, two relatively small special purpose detectors, and CMS and ATLAS two larger general purpose detectors. The ATLAS detector is the biggest of them and composed of concentric cylindrical detectors with end-caps [3]. The inner Detec-

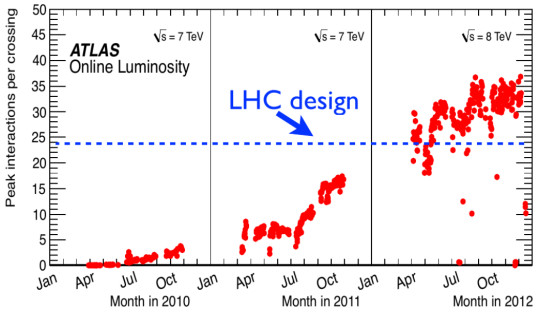

Figure 1: Peak interactions per bunch-crossing per time. LHC exceeded design value in 2012. [1]

tor(ID) is the closest to the beam and it is composed of Pixel, SCT and TRT trackers. Calorimeters are located around the ID. Calorimeters have electromagnetic and hadronic components comprised of Liquid Argon and Tile calorimeters. Muon detectors form the outermost shells of the detector. Toroidal and solenoid magnets provide the magnetic field for momentum and charge determination.

During the Run-1 period, LHC operated below its design energy, at 7TeV and 8TeV with a bunch spacing of 50ns. However the single bunch luminosity was increased which led to a higher number of collisions at each bunch crossing(pile-up) than design expectations as shown in Figure 1. Towards the end of Run-1, average pile-up at ATLAS exceeded 40 interactions per crossing, creating more than 1200 tracks. Since the end of 2012, LHC is being upgraded and will operate at full design energy and bunch crossing period in the Run-2 phase, starting in 2015. Pile-up is expected to increase up to 80 interactions per bunch crossing leading to many more tracks. Predictions for Run-3 with a peak luminosity of $10^{35} cm^{-2} s^{-1}$ and a pile-up of up to 140 interactions per bunch crossing are even higher. Since track finding is a combinatorial problem, total processing time will also increase exponentially. The estimated pile-up dependency of average reconstruction time is given in Figure 2.

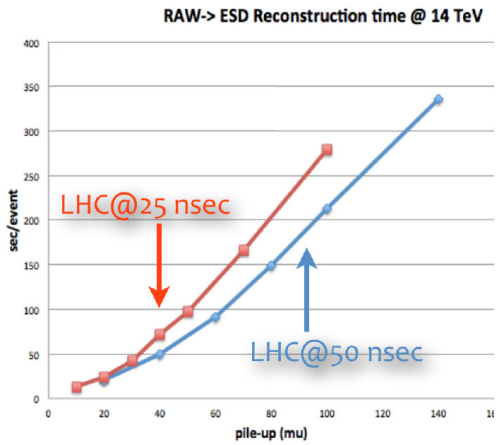

Figure 2: Average event reconstruction time versus pile-up at different bunch crossing rates. [4]

Up until a few years ago, the computing capacity of CPUs increased mostly due to increases in clock frequency. This increase usually compensated the increase in data rates with no changes in the code. However due to physical constraints, clock frequency of the CPUs has plateaued and the increase in computing capacity is provided by adding more cores and vector units to processors or in the form of co-processors. Unlike CPUs, co-processors tend to work effectively on certain types of highly parallel problems. However, they have a higher computing capacity per watt at a lower cost than CPUs. These properties make them attractive solutions for highly parallelizable workloads such as tracking.

2 ATLAS Software Framework

The ATLAS software framework, Athena, is composed of more than 4 million lines of C++ and about 1 million lines of python code written by hundreds of developers [5, 6]. The code is spread over more than 4000 shared libraries in about 2000 packages. Its design and implementation predates multi-core CPUs and thus was designed to run serially in a single process. Multi-core CPUs are exploited by running multiple independent processes at the expense of increased memory usage. Because of the design and complexity of the existing code, porting it to co-processors is not feasible if not impossible. However it is still possible to utilize co-processors

such as GPUs for some self contained parts of the software, yet it is still a challenge due to the multi-process nature of the framework.

2.1 Accelerator Process Extension (APE)

The Accelerator Process Extension (APE) is a framework designed for managing and sharing co-processors between different processes. It is based on a Server-Client model and its working diagram is given in Figure 3. Algorithms that are running in Athena have to prepare and serialize the input data and issue a processing request to the Offload Service. The Offload Service manages the communication between the APE Server and algorithms. Processing requests are forwarded to the APE Server via Yampl, an inter-process communication (IPC) abstraction layer. It abstracts various IPC technologies, such as shared memory, pipes and ZeroMQ [7] based layer for network communication. This enables running the APE server on a local host or a dedicated server host. The APE server does bookkeeping of requests and relays them to appropriate module which is a dynamically loaded plug-in that manages resources and contain algorithm implementations for a given hardware like GPUs or Intel MICs. They execute the requested operations on input data and return the results to the APE server. The APE server passes the results back to the Offload Service which in turn returns the results to algorithms that made the request.

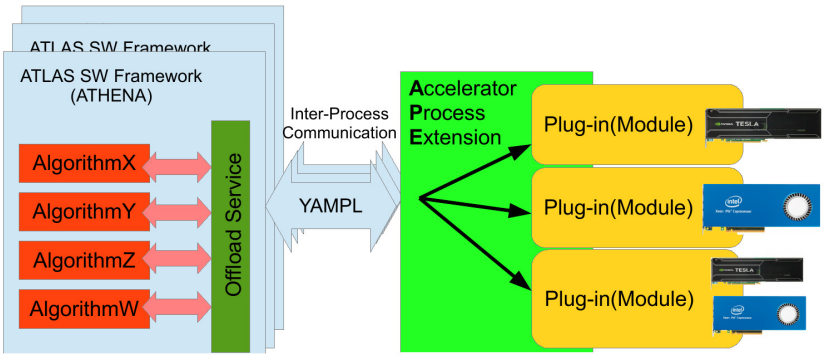

Figure 3: APE flow chart. Processing requests from algorithms together with data are sent to the server. Data are processed in appropriate module and results are sent back.

The APE framework also simplifies the development cycle and enables use of different languages and compilers which would not be possible inside the Athena framework. Since the client is hidden from algorithms and only input data is important, it is possible to use a standalone application to send a request with some predefined data sets without running the full reconstruction framework. IPC time between server and client is small compared to time budgets of the offloaded algorithms. In spite of that, it introduces a serial section and thus reduces the scaling as per Amdahl's Law. On the other hand, having multiple requests from different processes increases the throughput and utilization of GPU as per Gustafson's Law thus reducing the effect of the overhead.

3 GPU studies in ATLAS

There are several ongoing studies in ATLAS to use GPU resources. The Tracking for Muon and ID in Trigger and Reference Kalman-Filter for offline reconstruction are briefly described below.

3.1 GPUs for track finding in ATLAS Triggers

The ATLAS Trigger systems filter out more than 99.9% (in particular about 100 over 1 billion events are retained) of the events and select only interesting events. In order to decide whether an event is interesting or not, an incremental reconstruction is performed and a decision is produced in a few seconds. Data preparation and track reconstruction typically consume 50%-70% of the processing time budget. These processes contain some parallelization possibilities and they are implemented as GPU algorithms composed of various steps.

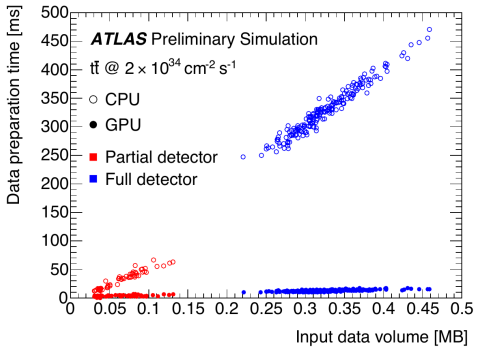

Figure 4: Bytestream decoding and clustering show a 26x speed-up on NVIDIA C2050 GPU vs single-threaded Intel E5620 CPU. [8]

Data preparation starts with decoding of detector output, the Bytestream. In this step, the compactly-encoded pixel and SCT hit information from the detector's readout buffers are retrieved, and decoded into hits within individual pixel and SCT modules. The Bytestream itself is divided into 16/32-bit words. While the CPU implementation of decoding iterates over the words sequentially, the GPU implementation maps each word to a GPU thread and does context detection and value decoding in GPU threads.

After the Bytestream is decoded, clusterization of neighboring hits in each module is done in order to take the activation of multiple adjacent detector cells by a single traversing particle. In the CPU, the pixel clustering is done in two nested loops and the SCT clustering is done in a single loop. In the GPU, a special cellular automaton-based algorithm is used to parallelize the comparisons done between hits. This approach allows parallelization in a module as well as across modules by assigning modules to different thread blocks. These clusters then converted to Space Points by using their locations in space and calculating their centers. A comparison of timings for CPU and GPU based data preparation implementations for different Bytestream sizes is given in Figure 4.

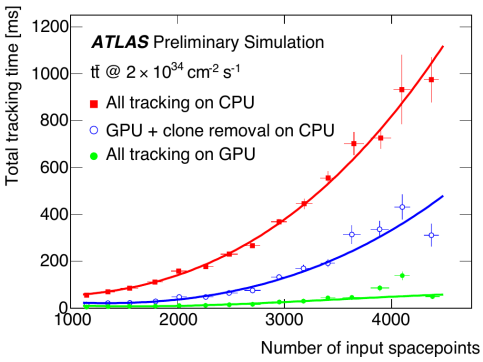

Figure 5: Track formation and clone removal show a 12x speed-up on NVIDIA C2050 GPU vs single-threaded Intel E5620 CPU. [8]

After data preparation is completed, track formation starts. The first step in track formation is the creation of track candidates. In this step,

space points in different layers are paired to create seeds. These seeds are then extended into outer silicon layers by looking for tertiary hits. This is a combinatoric algorithm which does not scale well with hit multiplicity. The CPU version is implemented as a series of nested loops while the GPU version essentially takes the Cartesian product of different detector layers by using a 2-dimensional block of GPU threads.

Once track candidates are formed, clone removal is done. During the clone removal step, track candidates which are formed by the same outer layer hits but different inner layer seeds are merged and copies are removed. Due to nature of the task, the GPU implementation splits the task in to identification and merging/removal while on the CPU it is done in a single step. Timing results of the track formation steps are shown in Figure 5.

Another place in the Trigger where GPUs are being studied is the Muon triggers. The ATLAS Muon trigger tries to select online events with relevant physics properties, based on the presence of a muon in the event. A muon is expected to leave hits in the inner tracking systems of ATLAS, as well as in the outer Muon Spectrometers while traversing the detector. On the other hand it is expected to leave a very small amount of energy in the calorimeter. To achieve best-possible muon reconstruction, all these have to be taken in account within allowed time budget. Initial implementation is aimed to calculate energy deposition in calorimeter cells around expected muon trajectory, which involves looping over several hundreds of calorimeter cells. However, flattening data structures for GPU utilization was found to be a limiting factor.

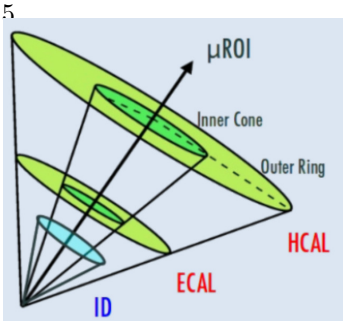

Figure 6: Muon regions of interest. Data around expected muon track at different detectors is taken into account.

3.2 GPU Based Reference Kalman-Filter for Offline Reconstruction

To allow a fair performance comparison between different systems, a reference Kalman- Filter[9] was implemented in four independent versions. A serial C++ version of the Kalman-Filter is used to generate a base performance index for further comparison. Then we implemented three different parallelized Kalman-Filter algorithms using OpenMP, OpenCL[10] and CUDA[11]. Each of these uses the same flat data structures and produces the same results. To measure the performance of each implementation a set of test data is used which consists of 96 events containing 19500 tracks and a total of 220000 hits. The test system uses an Intel XEON E5-1620 processor with 8 GB RAM and a nVidia GeForce GTX Titan 6 GB graphics card.

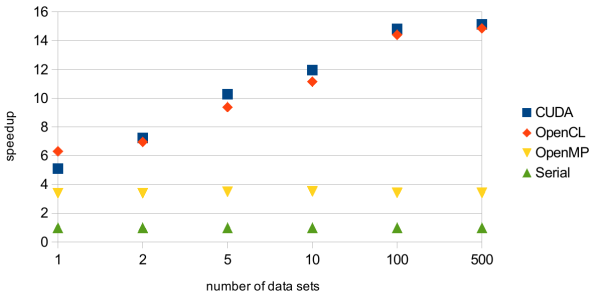

Figure 7: relative speedup of OpenMP, CUDA and OpenCL implementations compared to serial C++ code

To achieve the results shown in Figure 7, we improved the code base of both the OpenCL and CUDA implementation so that the whole Kalman-Filter chain is processed on the GPU. This includes forward and backward filtering and the smoothing step. All tracks per event are

processed in parallel using 5x5 GPU threads (corresponding to co-variance matrix entries). For an even higher performance, we use GPU-side matrix inversion which is done completely in Shared Memory of the GPU. With the used hardware we were able to invert up to 224 matrices in parallel on the GPU.
To simulate a higher load we ran the implementation in a benchmark fashion by calculating the same dataset several times in a loop (up to 500 times) as shown in Figure 7. The GPU implementations (CUDA and OpenCL) then achieved a relative speedup of about 15x compared to the serial C++ implementation.

4 Conclusion

ATLAS is continuously producing a tremendous amount of data that has to be processed. The possibility of compensating this rising amount of data just by the increase of CPU clock frequency will be no longer an option due to physical constraints. CPU clock frequencies are nearly at their maximum and therefore simply more cores and vector units are added so that one now has to make use of parallel programming. Co-processors such as GPUs have a relatively higher computing power per watt at a lower cost compared to CPUs, such that the use of co-processors looks like a promising solution.

We introduced the APE framework, an approach for integrating co-processors into the existing ATLAS software framework Athena. This study allows us to use GPUs within Athena with as little changes in the existing code as possible. We therefore have a working solution for short- to medium-term software framework integration.

We also have first implementations for online computing tasks in the ATLAS Trigger as well as a Kalman-Filter approach for offline computing. We are still evaluating other possible parallelizable problems from which we could gain reasonable speedups in computation time. Especially because of its combinatorial nature, track reconstruction seems to be promising.

We already achieved several encouraging results. Performing the data preparation steps of the ATLAS trigger on a GPU reached a relative speedup of up to 26x compared to a serial version run on a CPU. Parallelizing a Reference Kalman-Filter implementation achieved a speedup of 16x also compared to a single threaded CPU version.

References

[1] ATLAS Collaboration. Atlas experiment luminosity public results. `https://twiki.cern.ch/twiki/bin/view/AtlasPublic/LuminosityPublicResults`.

[2] Lyndon Evans and Philip Bryant. LHC Machine. *JINST*, 3:S08001, 2008.

[3] The Atlas Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008.

[4] M Elsing. Online and Offline Tracking for High Pileup. (ATL-SOFT-SLIDE-2014-513), Aug 2014.

[5] Marco Clemencic, Hubert Degaudenzi, Pere Mato, Sebastian Binet, Wim Lavrijsen, et al. Recent developments in the LHCb software framework Gaudi. *J.Phys.Conf.Ser.*, 219:042006, 2010.

[6] S Binet, P Calafiura, M K Jha, W Lavrijsen, C Leggett, D Lesny, H Severini, D Smith, S Snyder, M Tatarkhanov, V Tsulaia, P VanGemmeren, and A Washbrook. Multicore in production: advantages and limits of the multiprocess approach in the atlas experiment. *Journal of Physics: Conference Series*, 368(1):012018, 2012.

[7] iMatix Corporation. ZeroMQ web page. `http://zeromq.org/`.

[8] JTM Baines, TM Bristow, D Emeliyanov, JR Howard, S Kama, AJ Washbrook, and BM Wynne. An evaluation of the potential of GPUs to accelerate tracking algorithms for the ATLAS trigger. Technical Report ATL-COM-DAQ-2014-094, CERN, Geneva, Sep 2014.

[9] R. Fruehwirth, M. Regler, R.K. Bock, H. Grote, and D. Notz. *Data Analysis Techniques for High-Energy Physics*, chapter 3.2.5, pages 244 – 252. Cambridge Univerity Press, 2nd edition, 2000.

[10] Maik Dankel. Implementierung eines GPU-beschleunigten Kalman-Filters mittels OpenCL. Master's thesis, Fachhochschule Muenster, 2013.

[11] Rene Böing. Implementation eines CUDA basierten Kalman-Filters zur Spurrekonstruktion des ATLAS-Detektors am LHC. Master's thesis, Fachhochschule Muenster, 2013.