Journal of Physics: Conference Series **396** (2012) 022035 doi:10.1088/1742-6596/396/2/022035

Track finding in ATLAS using GPUs

J Mattmann and C Schmitt on behalf of the ATLAS Collaboration
Institut für Physik, Johannes Gutenberg-Universität Mainz, Staudingerweg 7, D - 55128 Mainz
E-mail: mattmanj@uni-mainz.de, schmittc@uni-mainz.de

Abstract. The reconstruction and simulation of collision events is a major task in modern HEP experiments involving several ten thousands of standard CPUs. On the other hand the graphics processors (GPUs) have become much more powerful and are by far outperforming the standard CPUs in terms of floating point operations due to their massive parallel approach. The usage of these GPUs could therefore significantly reduce the overall reconstruction time per event or allow for the usage of more sophisticated algorithms.

In this paper the track finding in the ATLAS experiment will be used as an example on how the GPUs can be used in this context: the implementation on the GPU requires a change in the algorithmic flow to allow the code to work in the rather limited environment on the GPU in terms of memory, cache, and transfer speed from and to the GPU and to make use of the massive parallel computation. Both, the specific implementation of parts of the ATLAS track reconstruction chain and the performance improvements obtained will be discussed.

1. Introduction

The ATLAS experiment at the LHC opens the way to a deeper insight into fundamental physical processes. For this, an unprecedented amount of recorded and simulated data needs to be processed within a given time budget. In order to cope with the requirements of event reconstruction, new approaches in data processing are needed. Since the reconstruction time per event starts to be dominated by the track reconstruction due to the large combinatorics that are involved, the latter is a crucial point for optimization. This applies in particular for increased pileup conditions that will be seen in the future.

In this context a subset of the track reconstruction algorithms that already exist within the ATLAS reconstruction framework have been ported to GPUs to test the speedup that can be achieved. For this feasibility study the GPU code has been implemented for the cylindric central region of the detector only. Technically the port is currently based on the CUDA framework by Nvidia but may as well be ported to the vendor-independent OpenCL standard.

2. The ATLAS Inner Detector

The ATLAS Inner Detector (figure 1) has three constituent parts: a (silicon) pixel detector, a silicon strip detector and a transition radiation tracker [1]. In the following only the pixel and strip detectors are considered in the track reconstruction since the transition radiation tracker requires different algorithms. The pixel detector has a very high resolution for precise vertex reconstruction whereas alignment and resolution of the silicon strip detector have been chosen to satisfy p_T measurement requirements. To be able to gain at least a limited position information parallel to the beam axis, each silicon strip detector layer consists of two panels with a stereo angle between them.

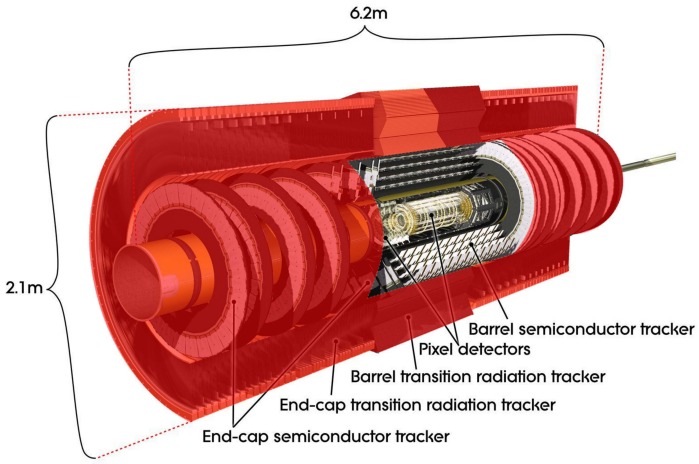


Figure 1. View of the Inner Detector; unmodeled regions are marked in red

Both silicon detectors are for their part composed of axis aligned detector modules in the central 'barrel' region and circularly aligned modules in the 'endcap' region on both sides of the central region.

3. GPUs at a glance

The desire for flexible rendering of extensive three-dimensional imagery (especially for computer games) led from a fixed function pipeline with hardware defined functionality to programmable shader units. By now a huge number of general purpose computing units ("CUDA cores" for Nvidia devices) is available and can be programmed using e.g. *C for CUDA*, an adapted subset of C++ [2].

Making use of a hierarchy of memory areas allowing for a fine-grained data locality management, a high memory bandwidth, hardware thread handling (covering memory access latencies) and massively parallel thread execution, enormous speedups can be achieved for suitable tasks.

4. Track reconstruction

The track reconstruction process is composed of three subsequent steps: The first step is the so-called seed finder, checking triples of hits on the innermost layers ("seeds") if they are a valid origin for a track. In the second step, the track parameters (position, direction and bending) obtained from every seed are used to propagate through the detector layers along the assumed track and match the track prediction with real hits on the detector layers. This is done using the Kalman algorithm [3]. Both steps have been ported to GPUs to benefit from their massively parallel architecture. Since seed validation as well as track propagation via Kalman filter application can be performed independently track by track, the many-core GPU architecture is predestinated for both problems.

The last step that still has to be performed by CPU code is a final track fit based on the Kalman filter results. In this last step, a number of corrections are applied such as a more precise description of the magnetic field, material layers causing multi-scattering etc. Since all these corrections need a lot of additional input information, porting this part of the algorithm was postponed. As the number of tracks that need to be fitted in the last step is rather limited

compared to the combinatorics that appear in the seed finder for example, the anticipated speed gain would be lower.

In order to speed up the development of this feasibility study, a few complications present in the full ATLAS reconstruction have been neglected, such as endcaps and transitional regions, seed finding making use of the first silicon strip layer and tracks with single hits missing along the course. These are not expected to bias the results significantly as the self-implemented CPU reference code used for the runtime comparisons has the same simplifications built-in.

5. GPU seed finder

The GPU seed finder makes use of the fact that the huge workload resulting from combinatorics can be processed independently by the single cores of the GPU. To reduce the number of seed candidates that have to be tested, all hits are sorted into segments dividing the single detector layers into z and ϕ segments. At this the angular segmentation is chosen to match the minimal p_T that needs to be considered (in this study 100 MeV). This means that for any segment on the innermost layer valid seeds can only be found within the same or adjacent angular segments on the following layers (see figure 2).

Each hit combination or seed candidate is checked by a dedicated thread which performs all tests necessary to verify the combination as a valid track seed. The result of each test is stored in an array indicating whether to reject or accept the corresponding hit combination. Therefore all hit combinations within all segments are indexed in a way that allows a mapping between a multi-dimensional hit combination and a linear index. The process is indicated in figure 2. The same mapping applies for the look-up step prior to the seed candidate tests in which the unique index of each thread is used to determine the hit triple to check.

To enable scaling of the algorithm especially with respect to very high pileup events (higher luminosity or heavy ion events), the results array can be emptied and its content copied to the main memory within the seed finder process. The final reconstruction step in which the array indexes are mapped to actual hit indices generally happens in the CPU part of the algorithm. Thought has been given to this fact for the following time measurements by adding the time needed for this step to the GPU runtime.

Currently, work is going on to optimize and improve data localilty and head for a more sophisticated work balancing. Furthermore the workload will be reduced using a more detailed segmentation scheme and memory usage will be optimized.

6. GPU propagation and Kalman filter

The results of the seed finder are subsequently fed into the GPU propagation and Kalman filter tool. The parallelization approach in this case is rather different from the previous one. Whilst for the seed finder each thread completely processes one seed candidate, in the propagation/Kalman filter step track processing is performed on a per-layer basis. Again each track candidate is handled by a dedicated thread but a redistribution of workload is applied within each step to handle occurrent multiplicity changes since tracks may split up or disappear.

A schematic outline of the single steps is presented in figure 3. Herein figure 3(a) shows the part of the algorithm that prepares the actual GPU code execution and processes the returned data whilst figure 3(b) shows the actual calls of GPU functions. These are called "kernel launches" and they always imply a parallel execution of a desired number of GPU processes.

In a preliminary stage not shown in the scheme, the detector geometry information (position and orientation as well as dimensions of the detector panels) must have been transferred to the GPU memory. On a per event basis the actual processing starts with the transfer of all hit information for the current event. Furthermore the initial track parameters as calculated from the seeds found in the previous step are transferred. The actual track reconstruction is performed

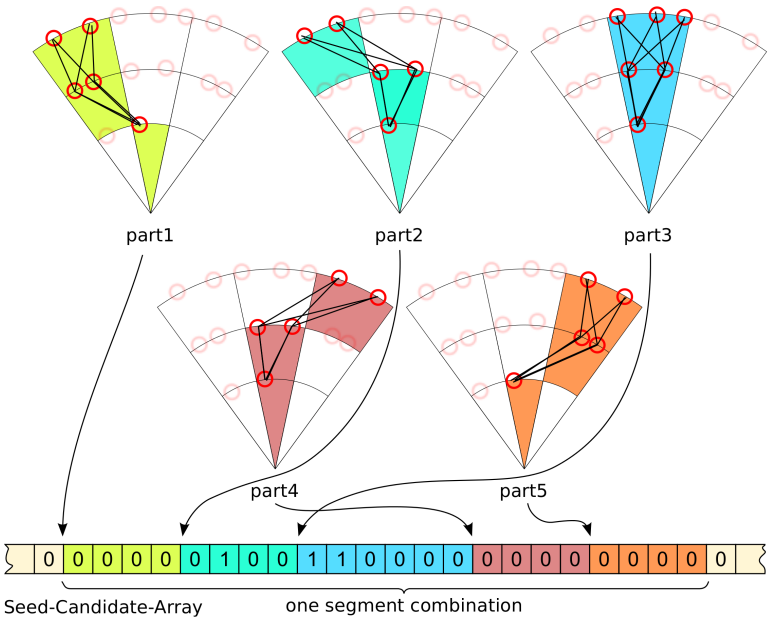
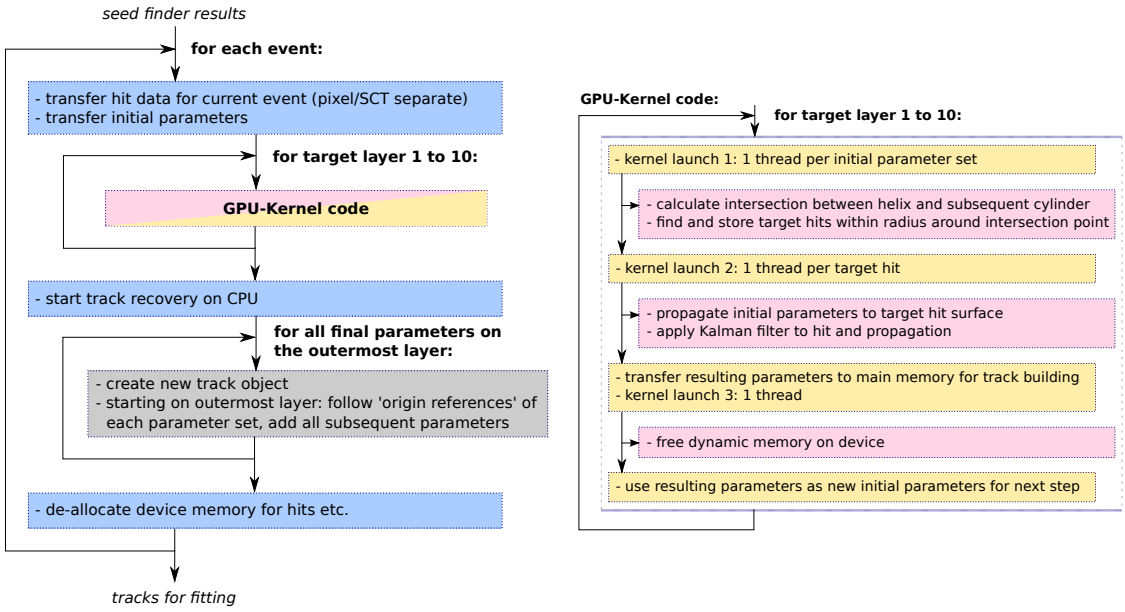


Figure 2. Seed finder parallelization approach: each thread handles a hit triple, combinatorics are reduced via segmentation; hit combinations are mapped to a one-dimensional index to store seed verification/rejection



(a) CPU flow (b) GPU kernel calls

Figure 3. GPU propagation and Kalman filter tool outline

in parallel for each set of initial parameters (seeds) and subsequently for each destination layer within the detector. The actual GPU code is called in a loop beginning with the second layer (index 1) as the first target. Altogether 11 layers in the barrel are considered: three layers of the pixel detector as well as each single layer of the four double layered silicon strip detector layers.

The first GPU call within the propagation loop (figure 3(b)) calls one thread per initial parameter set which first calculates the intersection between the track helix (B field considered homogeneous in the barrel region) and the subsequent cylinder layer. All hits in a region around this virtual intersection point are then considered to be possible track constituents and stored for further processing. In this step dynamical memory allocation is used to make better use of the GPU memory.

As the number of hits to be checked usually differs from the number of initial parameters (ambiguities, missing hits, noise), a new kernel is invoked. This time the number of threads corresponds to the number of goal hits. Each thread propagates the respective initial parameters along with their error matrix to the surface of the current hit. Here, the local hit information and the propagated parameters (each one with their errors) are merged using the Kalman filter. The result of this step is a new set of initial start parameters for the next propagation step and equally an intermediate state within the current track. All results are transferred back to the main memory for later reconstruction and a third kernel call frees the dynamically allocated memory on the GPU.

Having reached the outermost layer, the actual track building is started in the CPU. A loop iterates over all parameter sets on the last layer and creates an empty track object for each entry. Step by step for each parameter set on the outer layer the reference to the respective parent parameter set is traced until the innermost layer is reached thus building the current track in reverse. In a last step all event related data (hit information, parameter sets, maintenance data) is deleted from the GPU memory.

7. Results

As mentioned above as a result of the firstly limited model all speed comparisons had to be made using an own implementation of the seed finder and propagation/Kalman filter algorithm. For this reason the measurement results provide an indicator or a tendency comparing the sequentially executed CPU implementation with the massively parallel GPU implementation.

Since in the current implementation seed finder and propagation/Kalman filter are split into two seperate tools, speed measurements have been performed for both independently. The results are shown in figure 4 for the seed finder and figure 5 for the Kalman filter. Measurements have been performed using events with different numbers of generated muons (single particles) and simulated $t\bar{t}$ events with different numbers of tracks found by the official ATLAS software. Both implementations provide similar track reconstruction efficiencies and fake rates. Precise efficiency comparison maesurements are planned for the future.

For technical reasons, the tests have only been performed for the GPU Kalman filter using single muon files with up to 50 simulated tracks.

The results show that, as far as the seed finder is concerned, the GPU code outperforms the CPU code for more than 15 single muon tracks and reaches an almost constant speedup for $n_{\text{tracks}} \geq 100$ of $\frac{\text{slope}_{\text{CPU}}}{\text{slope}_{\text{GPU}}} = 13.7 \pm 0.5$. For low muon multiplicities the advance of the CPU code results from the overhead involved in the GPU usage (copying data between graphics card memory and main memory, launching kernels). This disadvantage is for larger events overcompensated by the speed gain from parallelization.

For the Kalman filter implementation the improvements are even more striking as a disproportionate run of the CPU/GPU time ratio curve is observed for $n_{\text{tracks}}[\text{single } \mu] \geq 8$ reaching a maximum speedup of 165 ± 30 for 50 muon tracks per event. It should be noted that

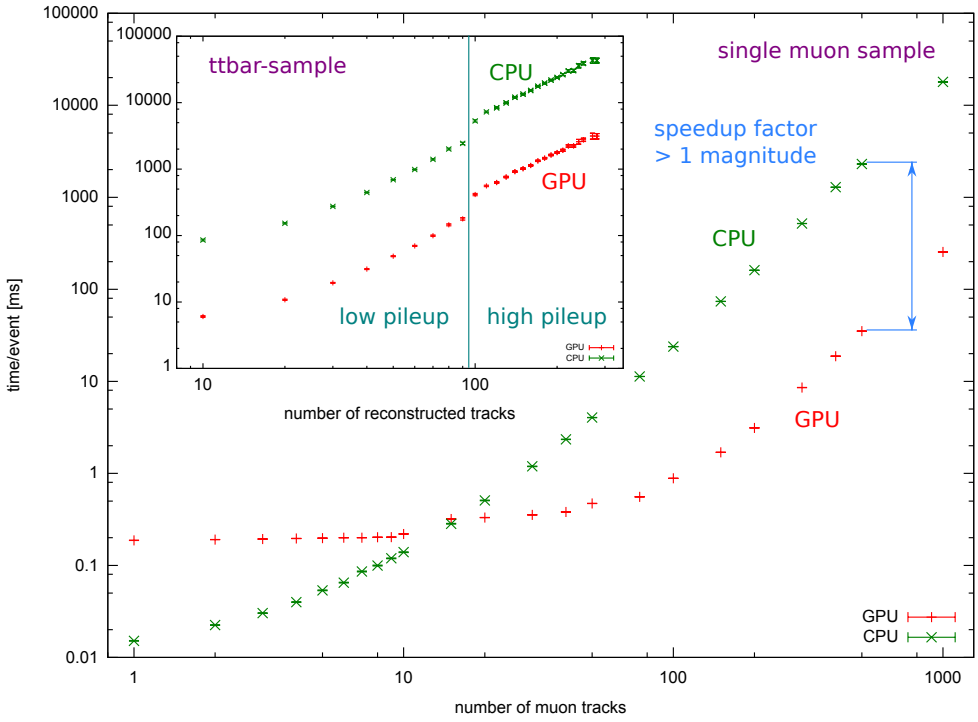


Figure 4. Runtime comparison between GPU seed finder and **self-implemented** CPU seed finder. The discontinuity indicated by the vertical line in the $t\bar{t}$ sample is caused by a change between low and high pile-up samples hence changing the track composition.

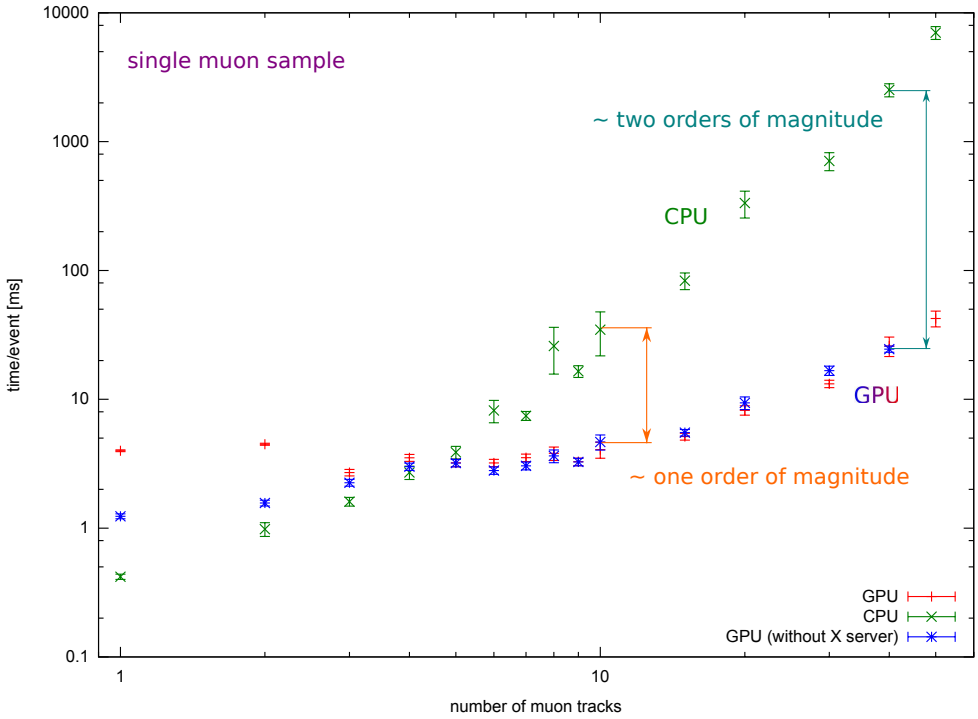


Figure 5. Runtime comparison between GPU propagation and Kalman filter and **self-implemented** CPU version (partially based on ATLAS reconstruction code)

(unlike for the seed finder) the GPU code runtime is influenced by the X server state: for low multiplicities the runtime clearly increases whenever the code is run with an active X server. Interestingly, in that case the runtime is even higher than for a slightly higher muon multiplicity ($n_{\text{tracks}}[\text{single } \mu] \sim 6$).

8. Conclusions

Based on the results of this feasibility study, an implementation with a fully modeled detector and the entire feature set of the ATLAS software track reconstruction for the full range of event sizes, will mean a major improvement for offline event reconstruction. Such a considerable increase of processing speed (implying a much higher event throughput) would bring about a very significant speedup of the overall event reconstruction time for ATLAS. Even if the addressed technical issue might require a less flexible workaround for the Kalman filter, the runtime results are still very promising and the full potential of this new approach should be utilized in the official software release. For this reason, an extension of the capabilities to the full feature set of the ATLAS software is intended. Simultaneously, a wrapper for the inclusion of the GPU tracking tools as parts of the official ATLAS reconstruction software is currently being developed.

Acknowledgments

This publication is supported in part by GRK Symmetry Breaking (DFG/GRK 1581).

References

[1] ATLAS Collaboration 2008 *JINST* **3** S08003
[2] NVIDIA Corporation 2009 *NVIDIA's Next Generation CUDA Compute Architecture: Fermi* 1st ed
[3] Cornelissen T, Elsing M, Gavrilenko I, Liebig W, Moyse E and Salzburger A 2008 *J. Phys.: Conf. Ser.* **119** 032014