Computing in High Energy Physics, CHEP2012

GPU-Based Tracking Algorithms for the ATLAS High-Level Trigger

Abstract

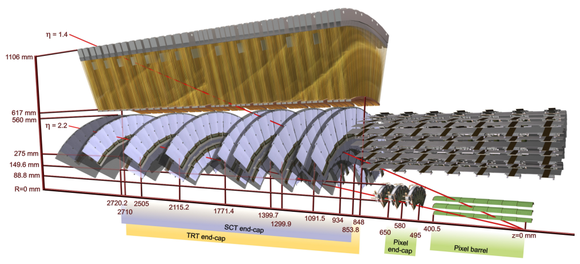
GPU-accelerated event processing is one of the possible options for the ATLAS High-Level Trigger (HLT) upgrade for higher LHC luminosity. This poster presents data preparation and track finding algorithms specifically designed to run on a GPU using a "client-server" solution for hybrid CPU/GPU event processing and integration of the GPU algorithms into existing ATLAS HLT software. The resulting speed-up of event processing times obtained with high-luminosity simulated data is presented and discussed.

Introduction

The ATLAS HLT consists of the Level 2 Trigger and Event Filter. The most time-critical part is Level 2, with a target execution time of 40 ms per Region-Of-Interest (ROI). Meeting this target at higher LHC luminosities is difficult. One of the options is GPU-based acceleration of the Level 2 algorithms. Technically, this is possible as HLT runs on dedicated farms which can be equipped with GPU cards. To study the feasibility of this option, Level 2 inner detector data preparation and track finding algorithms have been ported to GPUs using NVIDIA's Compute Unified Device Architecture (CUDA).

1. Inner Detector data preparation in the Level 2 Trigger

The ATLAS Inner Detector (ID) includes the Pixel and SCT subdetectors – large silicon trackers with thousands of detector modules.



The Pixel and SCT data preparation chain consists of
1. Decoding of bytestream (raw data) to a series of pixel and SCT strip hits
2. Clustering of pixels and strip hits
3. Formation of spacepoints (SP) in 3D space from pixel clusters and pairs of SCT clusters

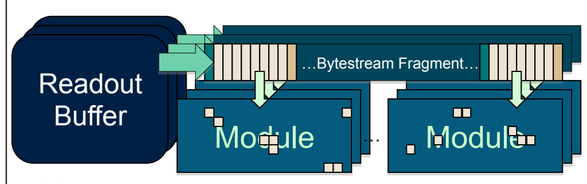
The data preparation has significant parallelism at different levels: ROIs, detector modules, raw data words – all are ideal for GPU-based code. The original code is CPU-oriented – not completely suitable for parallel execution. Some of the algorithms, in particular, pixel clustering, had to be fully redesigned.

2. Parallel algorithms for Inner Detector data preparation

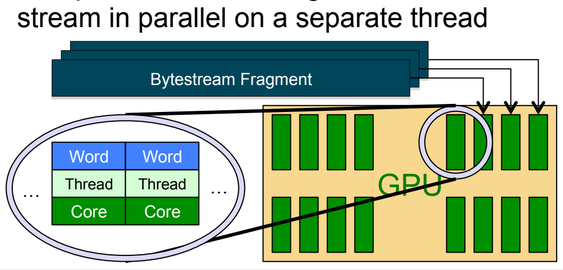
Data preparation for the ID has two primary steps: bytestream decoding and hit clustering:

Decoding

Decoding consists of converting the readout-optimized data format to a collection of hits in the silicon detectors

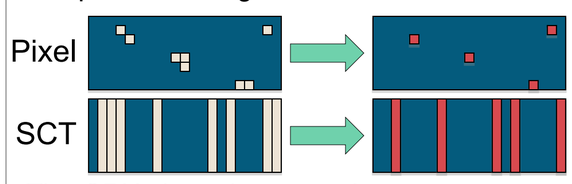


GPU decoding performs the same hit extraction as on the CPU, but maps the bytestream fragment from each readout buffer to a different CUDA block/streaming multiprocessor, decoding each word in the stream in parallel on a separate thread

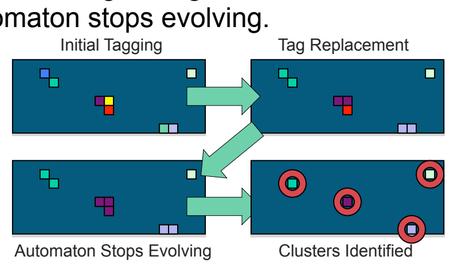


Clustering

Clustering groups adjacent hits into clusters which represent where a particle has passed through the silicon

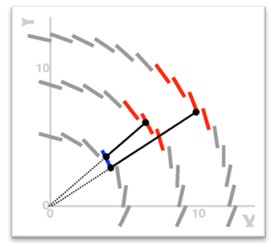


The GPU clustering procedure uses a cellular automaton to iteratively combine hits into groups. All hits are assigned an initial tag and then retagged by adjacent hits with a higher tag index until the automaton stops evolving.

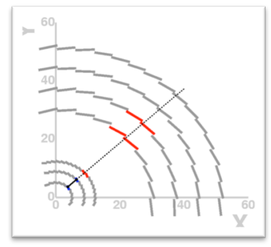


3. Combinatorial tracking algorithm in the Level 2 Trigger

The ATLAS SiTrack algorithm uses a combinatoric approach to track finding. Track seeds are formed using SPs from the two innermost detector layers. These seeds are then matched to SPs in outer layers to form triplets. Triplets with similar parameters are merged into track candidates. Candidates which share a large number of SPs are merged or removed based on the number of SPs they contain and their quality.



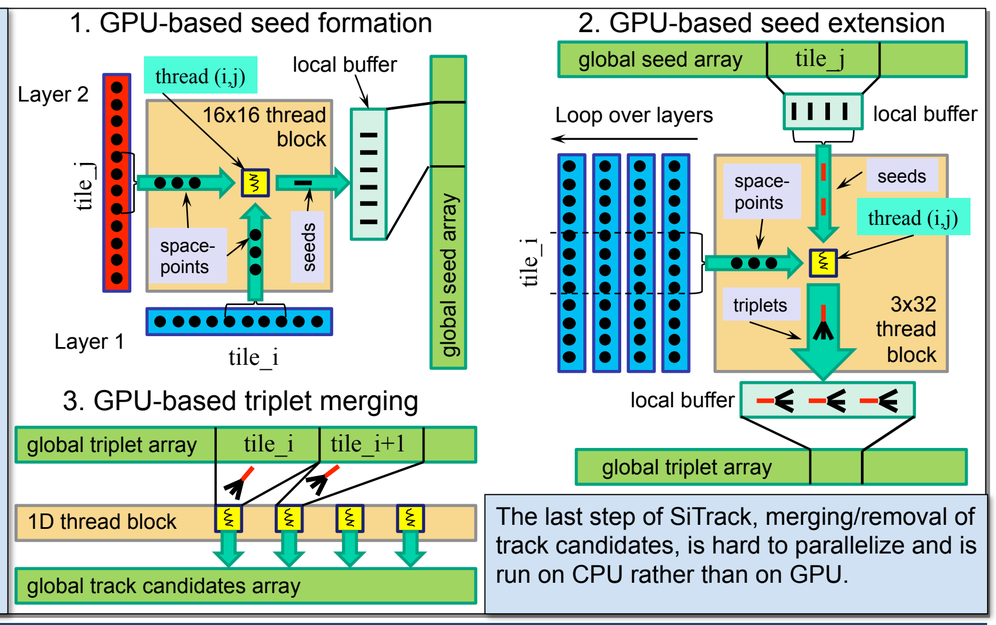
Track seeds are formed from spacepoints in the innermost detector layers



These seeds are extended by spacepoints in the outer layers to form triplets

4. GPU-accelerated SiTrack algorithm

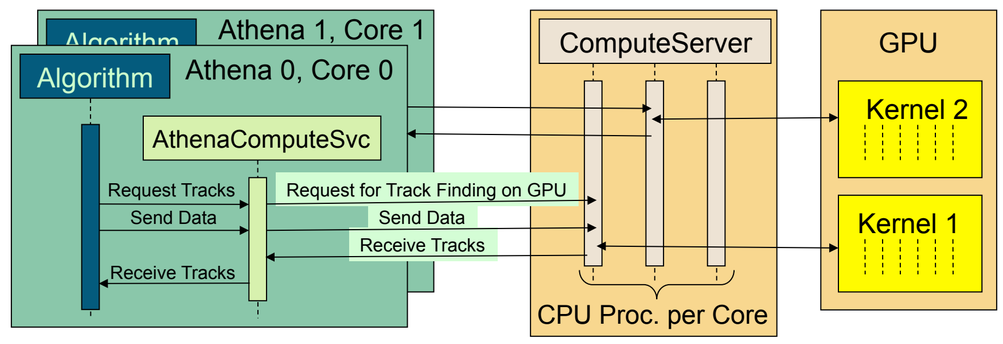
Each SiTrack step is split among threads arranged into 1-D or 2-D blocks, with each thread working on a small subset of input data (spacepoints, seeds, or triplets). Fast local buffers are used to hide latency of the global GPU memory.



The last step of SiTrack, merging/removal of track candidates, is hard to parallelize and is run on CPU rather than on GPU.

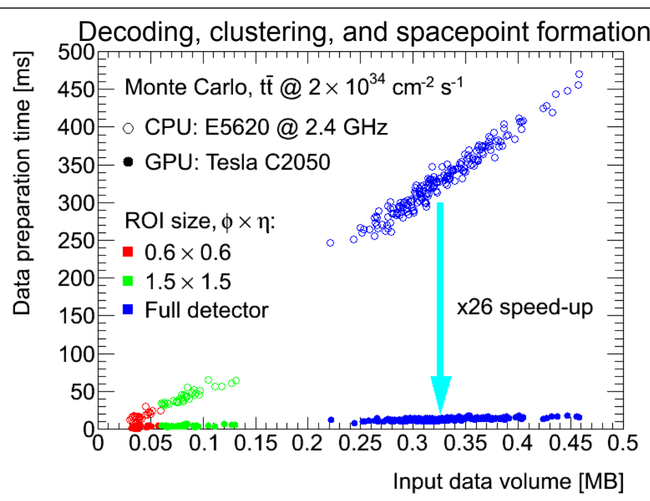
5. The "client-server" architecture for hybrid CPU/GPU processing

Separation of code using NVIDIA CUDA (or any other GPU API) from the ATLAS event processing framework (Athena) is achieved via a "client-server" model, with a compute server based on NVIDIA CUDA, and CUDA-free clients.

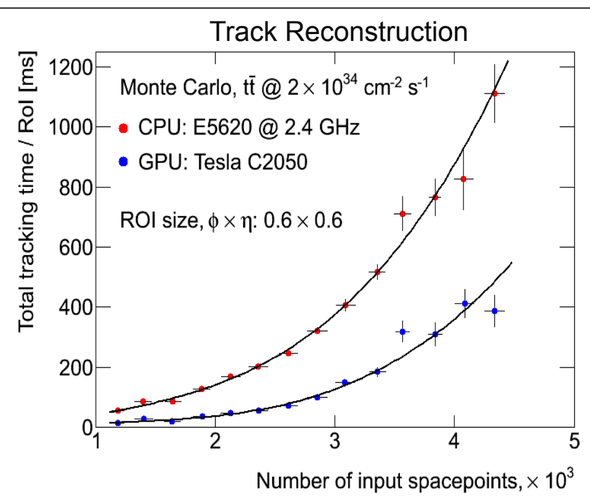


- High-level abstraction of the GPU API via AthenaComputeSvc which:
  - Provides a set of high-level routines: e.g. track finding – ported CPU-bound code which could benefit from GPU acceleration
- AthenaComputeSvc sends commands and data to the ComputeServer which starts the corresponding CUDA kernel on the GPU
- Multiprocessing server allows for sharing of the GPU between clients

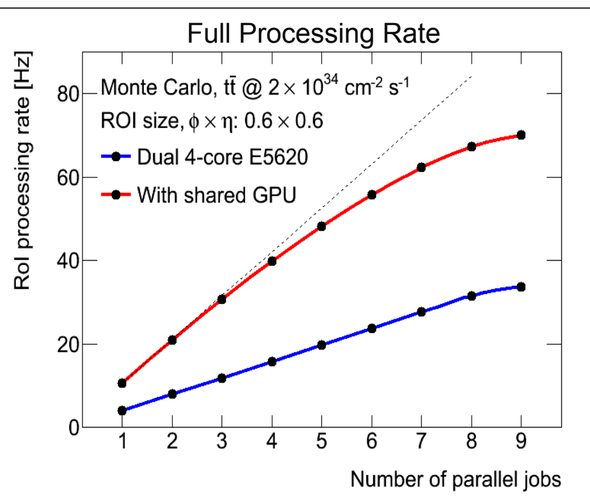
6. Results and discussion



Data preparation time is reduced by a factor of 26 and shows very little dependence on input data volume



Track finding also shows increased performance, though it is limited by the track merging/removal being run on the CPU



Overall processing throughput is increased when using the GPU, limited only by the number of GPU multiprocessors

Tests with high-luminosity simulated data show increased performance in data preparation and tracking. They also demonstrate the viability of the client-server CPU/GPU model. Overall rate of processing is significantly increased with the GPU, and the limiting factor is no longer processing time, but memory available to run instances of Athena on the host system.

Conclusion and outlook

Significant speed-up for both data preparation and track finding is observed and the client-server architecture viability demonstrated. Future goals include porting kernels and server code to OpenCL to perform tests with ATI FirePro cards and multi-core CPUs working as co-processors.

Dmitry Emeliyanov (STFC RAL) and Jacob Howard (University of Oxford)

ATLAS EXPERIMENT