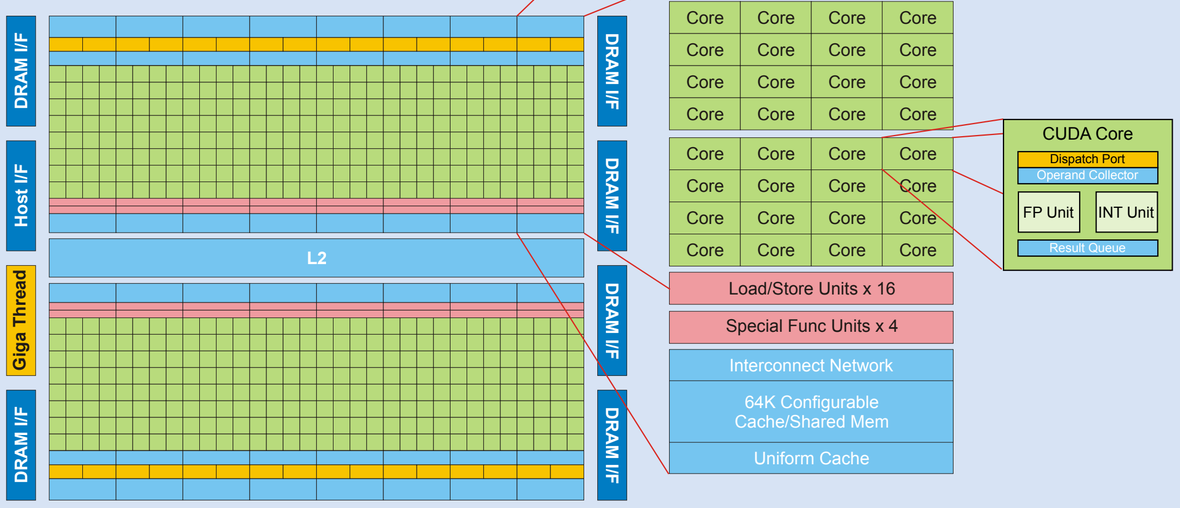
Track finding in ATLAS using GPUs

Johannes Mattmann and Christian Schmitt on behalf of the ATLAS Collaboration

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

GPUs at a glance / Track reconstruction principle

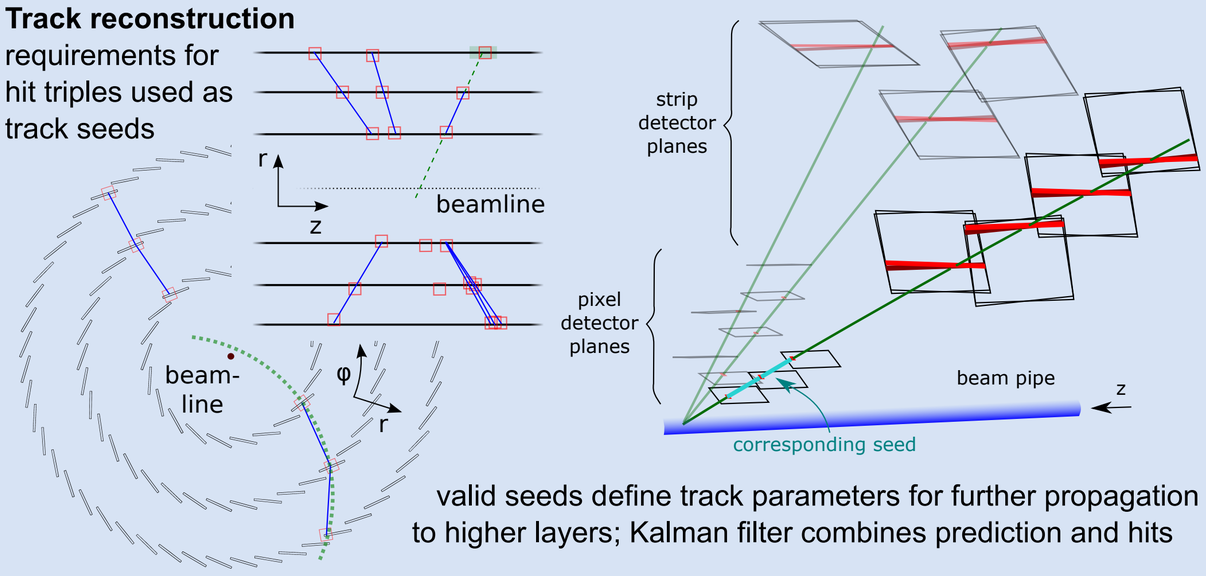
The following image is a schematic representation of the current Nvidia CUDA archtecture. The GPU contains a set of "streaming multi-processors" which are equipped with local storage and control units. Each one again contains a number of CUDA cores.



These cuda cores are computing elements containing each a floating point and an integer unit and some control logic. Operations are executed in parallel on a subset of CUDA cores simultaneously. Access latencies to global memory or special functions (sin, cos, sqrt,...) can be covered by hardware thread handling allowing for quasi-lossless thread switches. A hierarchy of memory areas lets the developer adjust data locality vs. memory requirements.

Track reconstruction

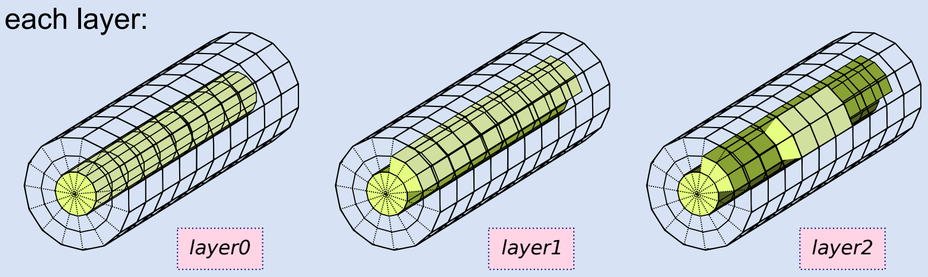
requirements for hit triples used as track seeds



valid seeds define track parameters for further propagation to higher layers; Kalman filter combines prediction and hits

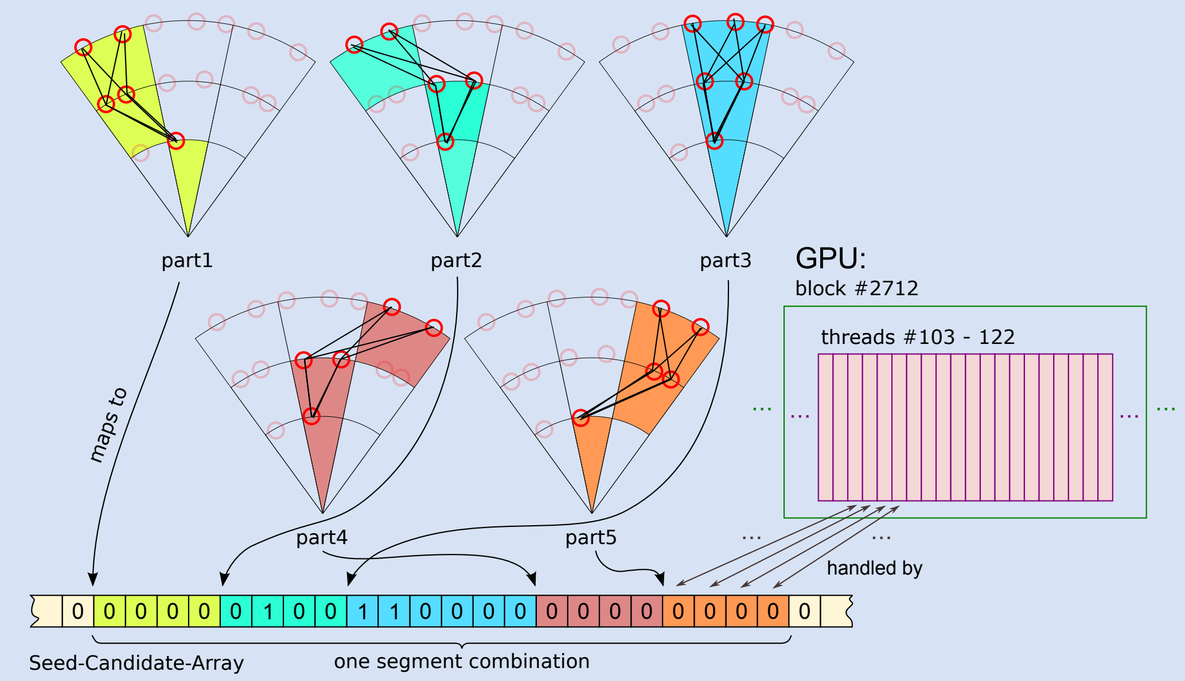
Seed finder

Currently hit triples on the three innermost central barrel layers are checked if they form a viable seed for track finding. To reduce combinatorics by a priori excluding invalid seed candidates, seed search is constrained to a defined set of angular and longitudinal segments on each layer:

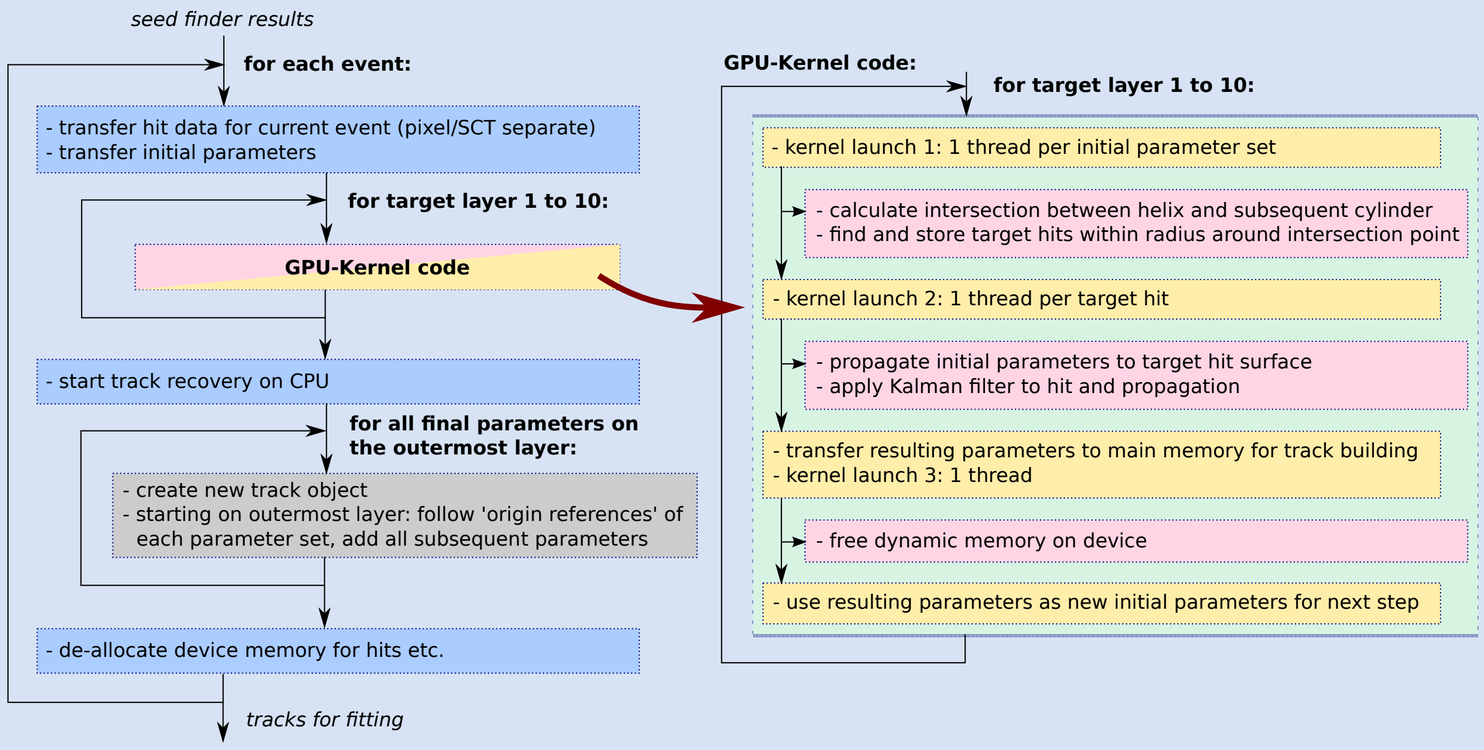


Basic approach for GPU seed finder:

- global coordinates and float types used (sufficient precision plus higher efficiency)
- every hit triple is checked by a dedicated thread (using the GPU as a vast thread pool); approach for next step: use block structure to optimize data locality/hit data access
- results are stored in boolean array indicating whether seed candidate is valid or not; in the near future to be optimized using a more dynamic storage method
- mapping between linear array and hit combination via segmentation-based lookup step (see figure below); according mapping between thread indices and hit combinations
- memory scaling for huge events/small device memories implemented; multiple iterations possible, each handling a subset of seed candidates
- seed reconstruction (lookup of actual hits) performed afterwards by CPU code



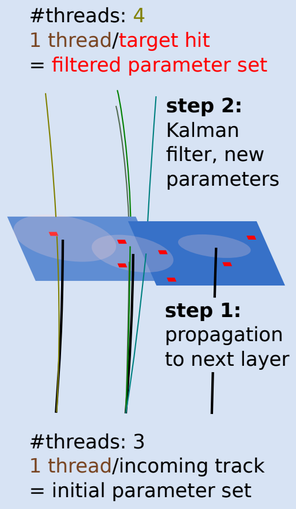
Propagation and Kalman filter



In general the implementation requires a CPU/GPU interplay (see outer left figure). For each event hit data (local x/y position, errors, detector identifier) and reconstructed seeds are transferred to global device memory. Propagation and Kalman filter application is performed in parallel on a per layer basisin each step. Afterwards all found tracks are reconstructed from the track parameters on each layer.

General procedure (GPU-Kernel code/right figure):

- **first GPU method:** every thread propagates a parameter set (initially gained from related seed) to next cylinder layer (faster, neglects panel)



- all hits within an acceptance region around intersection are stored for further processing → change of multiplicity, requires global sync.
- **second GPU method:** one thread per stored hit; propagation of related track parameters (+ covariance matrix) to hit surface is performed
- hit position and error are merged with propagated parameters and covariance matrix using Kalman filter
- **third GPU method:** dynamically allocated memory is released

Results and Outlook

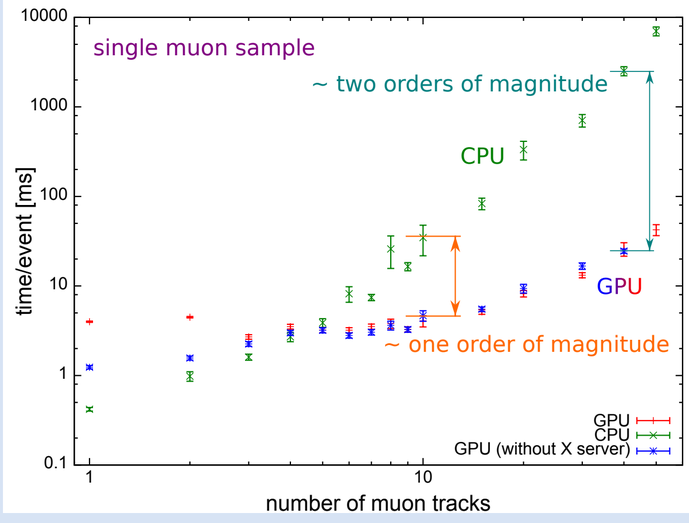
Runtime comparison seed finder:

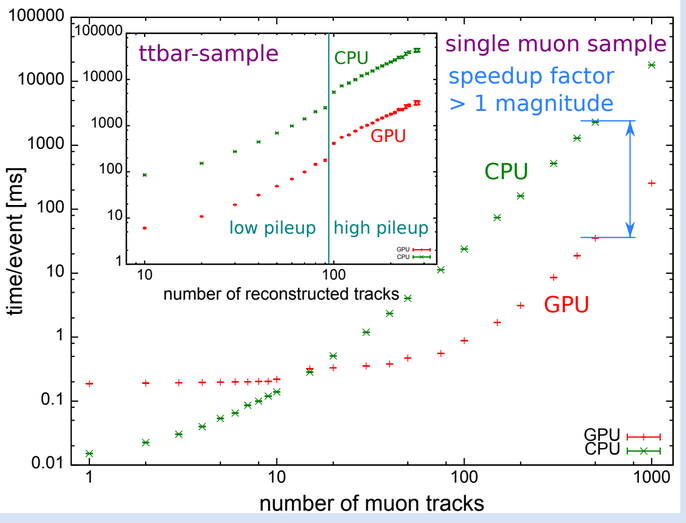
- working for entire test range of single muon and simulated $t\bar{t}$ tracks
- GPU outperforms CPU for more than 15 single muon tracks
- fit result for $n_{tracks} \geq 100$:

$$\frac{\text{slope}_{CPU}}{\text{slope}_{GPU}} = 13.7 \pm 0.5 \text{ const. speedup}$$

Propagation/Kalman filter:

- working for single muon tracks
- higher runtime fluctuations, X server state dependence observable
- for $n_{tracks,\,\mu} \geq 8$: CPU/GPU time ratio run disproportionate; 165 ± 30 max. speedup factor (50 single μ tracks)





Outlook:

- extend current model to full feature set: include endcaps and transitional region, implement all special cases from official framework (in progress, to be finished by the end of 2012)
- implement optimized seed finder; thus optimize memory usage and speed of execution
- build an official ATLAS reconstruction software package from GPU implementation code
- promote equipment of data centers with GPUs during shutdown period