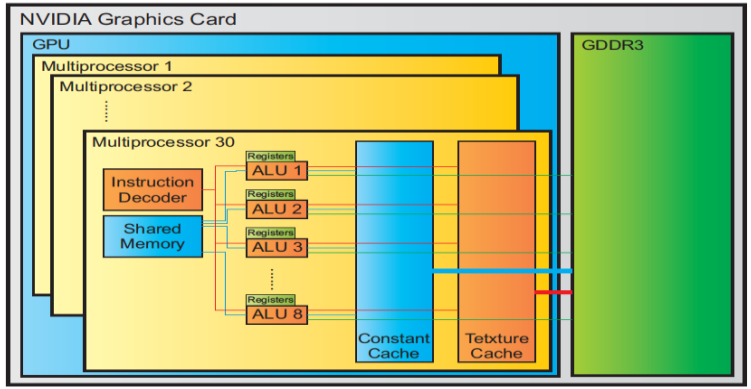
ALICE R&D with GPUs

Dr. David Rohr, drohr@cern.ch
ALICE, High Level Trigger
Frankfurt Institute for Advanced Studies
Second Joint Workshop on DAQ@LHC, 14.4.2016

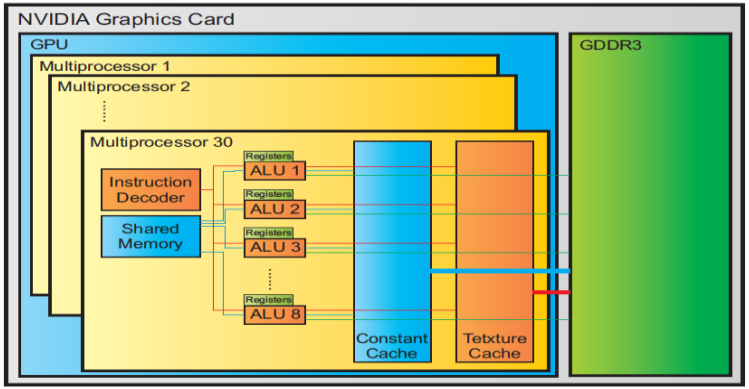
Questions

- **General approach:**
  - Which application / which part of reconstruction can benefit from GPU offload.
  - How to parallelize?
    - How to use the various GPU multiprocessors (cores)?
    - What about all threads (vector-units) of one multiprocessor?
  - Why to use GPU?
    - Better latency
      - GPUs can be very fast at a special task.
        - E.g. for a trigger where latency matters.
    - Better throughput
      - We can usually just buy more computers – so throughput means events / seconds / CHF.
  - How many GPUs per server, and do we need a client-server approach.
- **Technical challenges?**
  - Don't want to maintain multiple code bases.
    - We will probably always need a CPU version as well, at least for reference.
    - How to make sure, that the GPU result is correct?
    - Can we expect the same results on CPU and GPU?

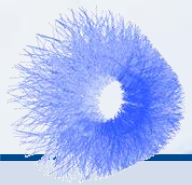


Questions

- **General approach:**
  - Which application / which part of reconstruction can benefit from GPU offload.
  - How to parallelize?
    - How to use the various GPU multiprocessors (cores)?
    - What about all threads (vector-units) of one multiprocessor?
  - Why to use GPU?
    - Better latency
      - GPUs can be very fast at a special task.
        - E.g. for a trigger where latency matters.
    - **Better throughput**
      - **We can usually just buy more computers – so throughput means events / seconds / CHF. This is what we currently aim for!**



- **Technical challenges?**
  - Don't want to maintain multiple code bases.
    - We will probably always need a CPU version as well, at least for reference.
    - How to make sure, that the GPU result is correct?
    - Can we expect the same results on CPU and GPU?

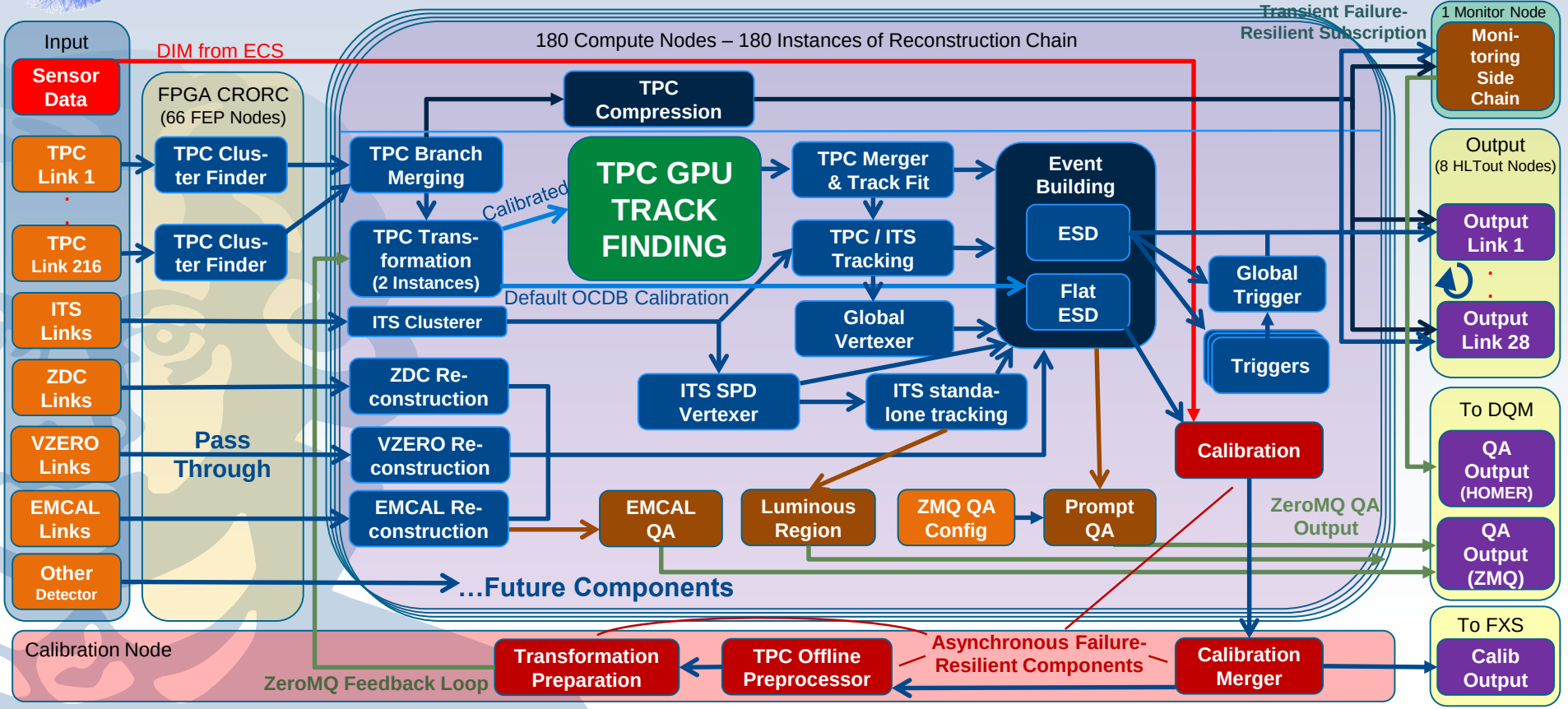


FIAS Frankfurt Institute for Advanced Studies

ALICE HLT RECONSTRUCTION

Overview of current HLT components

FIAS Frankfurt Institute for Advanced Studies



ALICE HLT Components

CPU Utilization – 100% ~ 1 CPU core – Reconstruction of PbPb data at 800 Hz

- 0.67% EMCAL Digit Maker
- 43.53% EMCAL Raw Analyzer
- 0.59% EMCAL Stu Analyzer
- 44.69% EMCAL Tru Analyzer
- 0.45% EMCAL Trigger Data Maker
- 1.55% EMCAL Trigger Maker
- 62.29% EMCAL QA
- 0.56% CTP RE Trigger
- 1.8% Global Trigger
- 5.85% ITS SDD Cluster Finder
- 2.06% ITS SPD Cluster Finder
- 5.02% ITS SSD Cluster Finder
- 19.64% ITS SPD Standalone Tracker
- 10.68% ITS SPD Vertexer
- 118.18% ITS-TPC Tracker
- 43.22% Global Vertexer
- 1.11% Luminous Region
- 6.8% Beam-Gas QA
- 2.83% PromptQA
- 34.4% TPC Branch Merging
- 78% TPC Cluster Transformation
- 81.98% TPC Cluster Transformation (Calibrated)
- **97.22% + GPU TPC CA Tracker – Alternative: 20 CPU cores**
- 41.63% TPC Track Merger and Track Fit
- 347.32% TPC Huffman Compression
- 298.97% TPC Calibration
- 1.3% VZERO Reconstruction
- 1.07% ZDC Reconstruction
- 183.29% Global ESD Converter
- 145.58% FLAT-ESD Converter
- FPGA (CRORC) TPC Cluster Finder

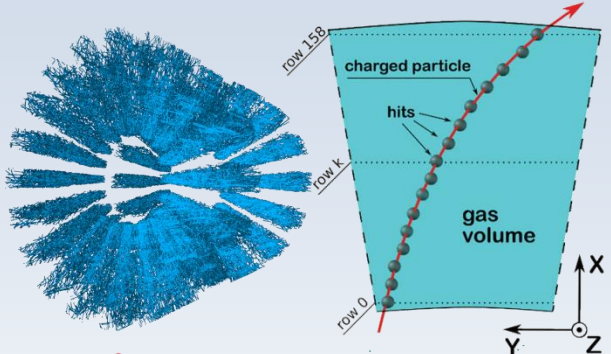
The compute hotspot is TPC tracking – best suited for GPU offload in ALICE – This is more difficult for other experiments.

FIAS Frankfurt Institute for Advanced Studies

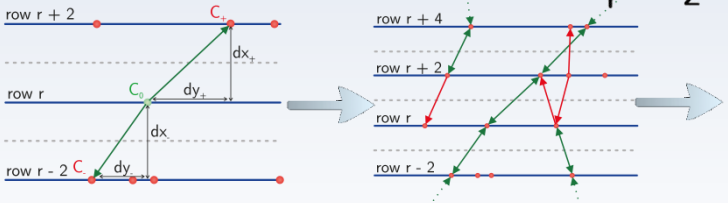
TRACKING ALGORITHM

Tracking Algorithm

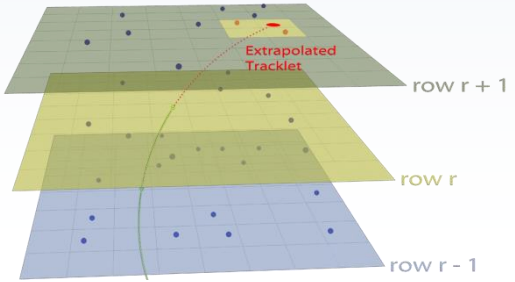
- **TPC Volume is split in 36 sectors.**
  - The tracker processes each sector individually.
  - Increases data locality, reduce network bandwidth, but reduces parallelism.
  - Each sector has 160 read out rows in radial direction.



- **1. Phase: Sektor-Tracking (within a sector)**
  - Heuristic, combinatorial search for track seeds using a Cellular Automaton.
    - A) Looks for three hots composing a straight line (**link**).
    - B) Concatenates links.



  - Fit of track parameters, extrapolation of track, and search for additional clusters using the Kalman Filter.



- **2. Phase: Track-Merger**
  - Combines the track segments found in the individual sectors.

Structure of HLT TPC Tracking

Tracking split in 4 main (abstract) steps.

- Each step is internally split in technical substeps.
- Phase 1 (Steps 1 and 2) on GPU, Phase 2 (Steps 3 and 4) and CPU!

#	Task	How	Locality	Description	Time	Device
1	Seeding	Cellular Automaton	Very Local (hit and adjacent hits)	Find short track candidates of 3 to about 10 clusters.	Ca 30%	GPU or CPU
2	Track following	Kalman Filter (simplified)	Sector-local	Fit parameters to candidate, find full track segment in one sector via track following with simplified Kalman filter (e.g. constant B-field, y and x uncorrelated, …)	Ca 60%	
3	Track Merging	Combinatorics / Mathematics	Global	Merge track segments within a sector and between sectors	Ca 2%	CPU only
4	Track Fit	Kalman Filter (full)	Global	Full track fit with full Kalman filter (polynomial approximation of B-field)	Ca 8%	CPU (GPU possible)

GPU Tracking

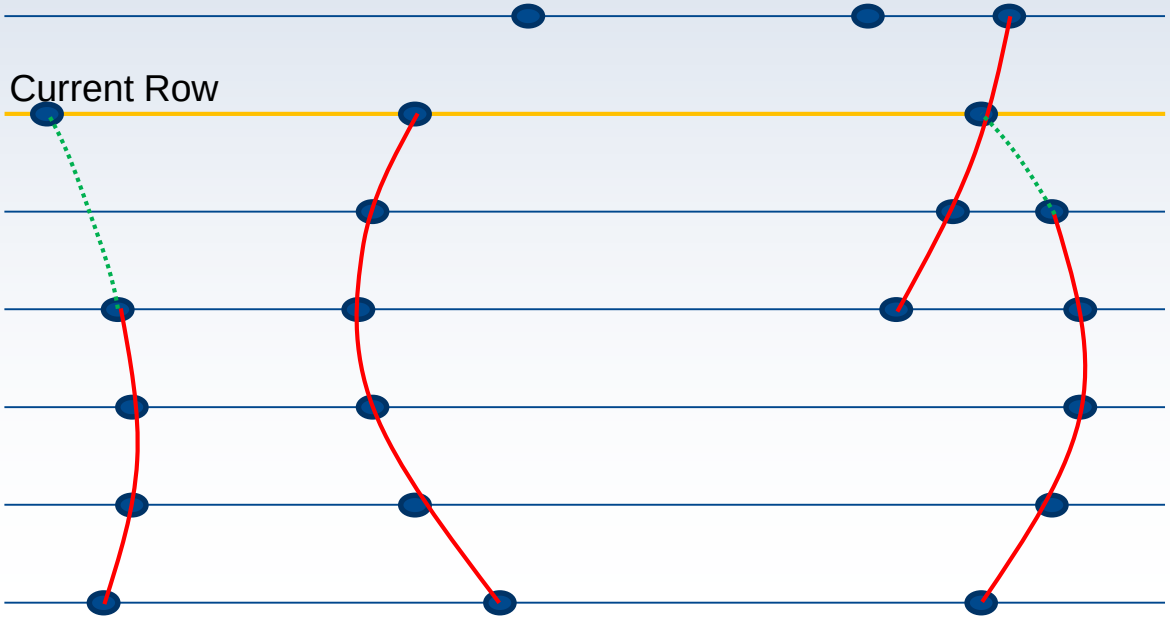
- **Parallelization scheme suited for threads inside the multiprocessor.**
- **Today, different multi-processors can execute different programs.**
- **When ALICE GPU tracking was developed, this was not possible.**

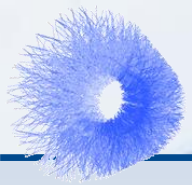
→ **Currently, we use the same scheme for parralelize over multiprocessors.**

- Only works, if there are many tracks / clusters.
- OK for PbPb with TPC.

Parallel Track Construction

Tracks are independent and can be processed simultaneously.
Clusters can be assigned to multiple tracks, this is solved later.





CPU AND GPU CODE / CORRECTNESS

Common tracker source code

- **CPU and GPU tracker (in CUDA and OpenCL) share common source files.**
- **Specialist wrappers for CPU and GPU exist, that include these common files.**

common.cpp:

```
__DECL FitTrack(int n) {
….
}
```

cpu_wrapper.cpp:

```
#define __DECL void
#include ``common.cpp``

void FitTracks() {
  for (int i = 0;i < nTr;i++) {
    FitTrack(n);
  }
}
```

cuda_wrapper.cpp and opencl_wrapper:

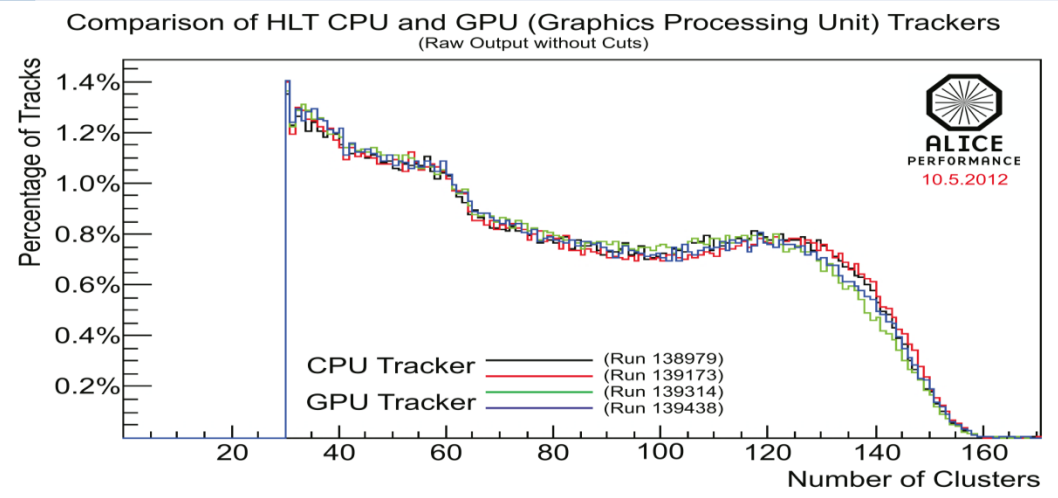
```
#define __DECL __device void
#include ``common.cpp``

__global void FitTracksGPU() {
  FitTrack(threadIdx.x);
}

void FitTracks() {
  FitTracksGPU<<<nTr>>>();
}
```

→ **Same source code for CPU and GPU version**

- The macros are used for API-specific keywords only.
- The fraction of common source code is above 90%.

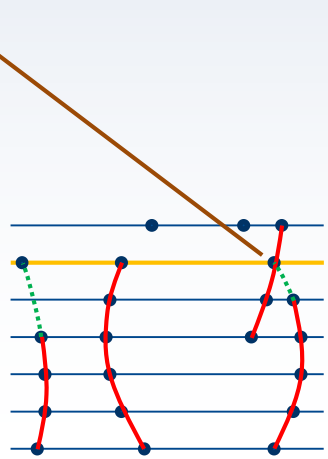
Consistency of Tracking Results

- **The first runs showed some inconsistencies in cluster to track assignment statistics, but not in physical observables. Three causes were identified:**
  - Cluster to track assignment
  - Track Merger
  - Non-associative floating point arithmetic



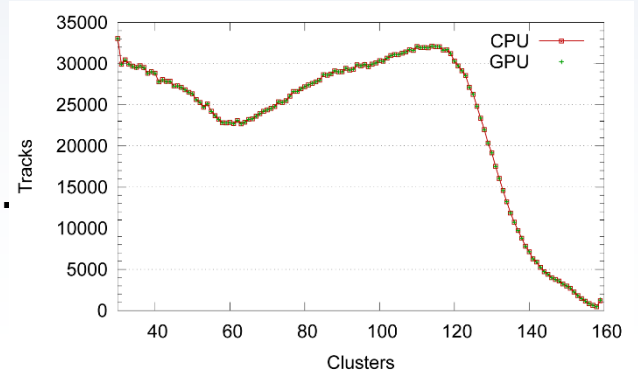
Consistency of Tracking Results

- **Cluster to track assignment**
  - Problem: Cluster to track assignment was depending on the order of the tracks.
    - Each cluster was assigned to the longest possible track. Out of two tracks of the same length, the first one was chosen.
    - Concurrent GPU tracking processes the tracks in an undefined order.
  - Solution: Both the chi² and the track length are used as criteria. It is extremely unlikely that two tracks coincide in both values.
- **Similar problem in track merger, which depended on track order.**



Consistency of Tracking Results

- **Non associative floating point arithmetic**
  - Problem: Different compilers perform the arithmetic in different order (also on the CPU).
  - Solution: Cannot be fixed, but...
    - Slight variations during the extrapolations do not matter as long as the clusters stay the same.
    - Inconsistent clusters: 0,00024%
- **Now, perfect match of CPU and GPU results in plots…**
- …But not binarily.

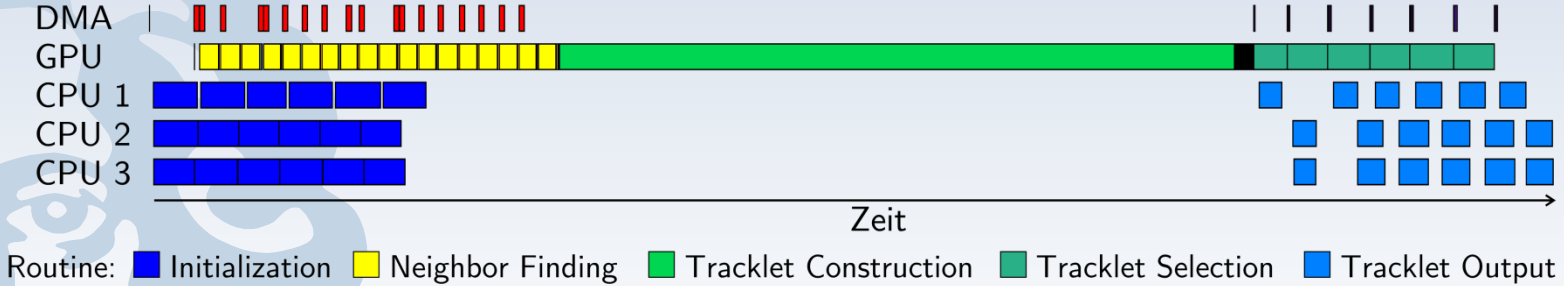


FIAS Frankfurt Institute for Advanced Studies

PERFORMANCE

Optimizations (Performance)

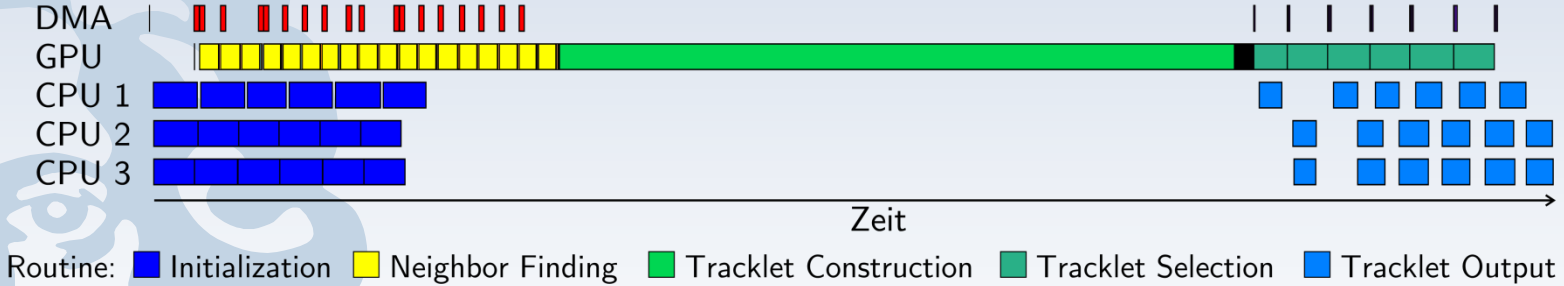
- **Separation of event in sectors enables the use of a pipeline Pipeline:**
  - Tracking on GPU, pre-/postprocessing on CPU, and data transfer run in parallel.



- **ALICE processes one event at a time (approach for old GPUs) – and one GPU kernel at a time!**
- No client-server mode, 1 GPU per compute node.
- Very good GPU utilization for large events: >= 95%.
- Today, we can overlap multiple events (see later), i.e. run different programs on different multiprocessors.
- Even within one event, we can overlap kernels to exploit parallelism better.

Optimizations (Performance)

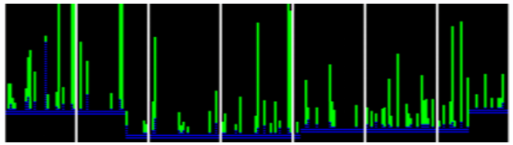
- **Separation of event in sectors enables the use of a pipeline Pipeline:**
  - Tracking on GPU, pre-/postprocessing on CPU, and data transfer run in parallel.



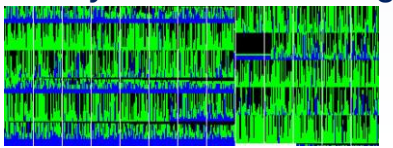
- **Problem: tracks are differently long. Through dynamic scheduling, GPU can be fully used.**

- Black : Idle
- Blue : Track Fit
- Green : Track Extrapolation

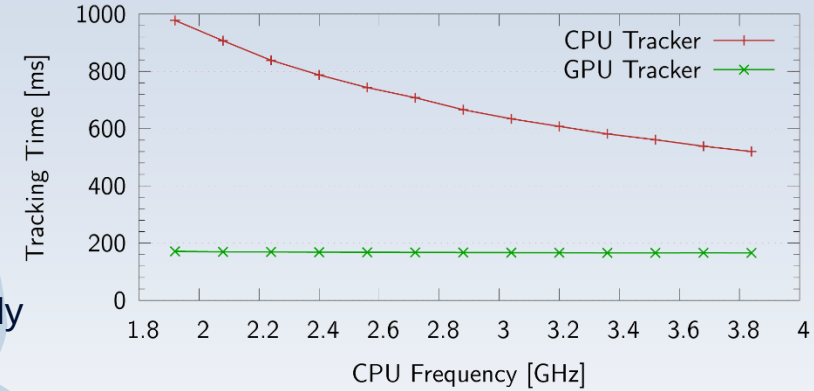
No dnymic scheduling



With dynamic scheduling

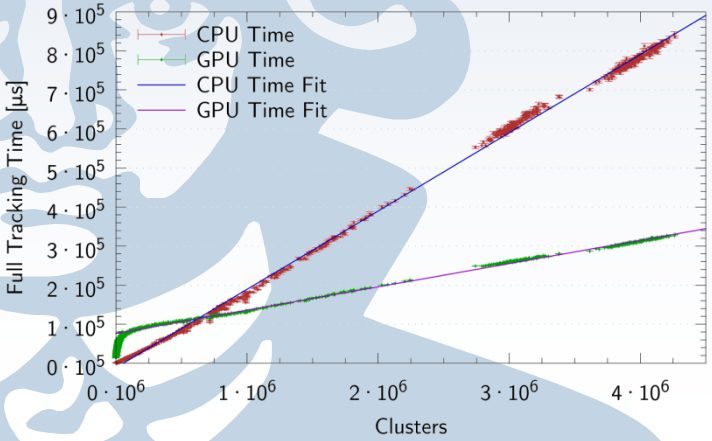


Results

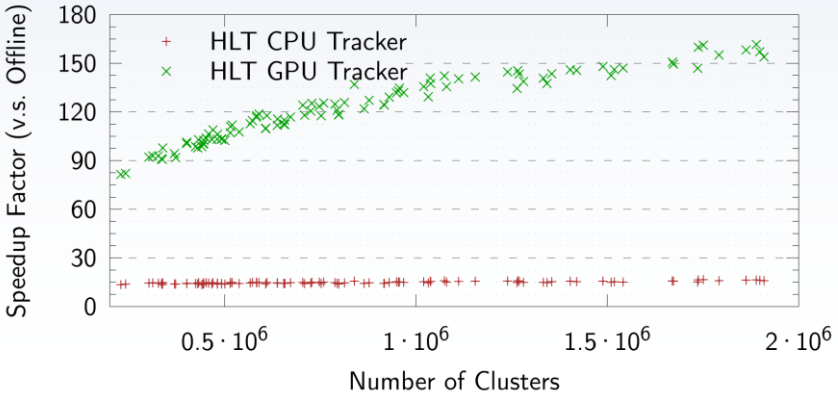


GPU performance independent from CPU.

Tracking times scales linearly with input data size.



Approx. 150 times faster than offline tracking.



Maximum processing rates

- **Maximum TPC readout bandwidth: ~ 50 GB/s.**
- **Corresponds to about 1 kHz event rate for PbPb at maximum Luminosity.**
  - HLT TPC tracking as able to process this data / event rate.

- **The situation becomes more complicated for smaller events:**
  - 4 kHz at 12,5 MB / event is more challenging than 1 kHz at 50 MB / event.
  - With out current parallelization scheme (single event at a time), smaller events lead to worse GPU utilization.

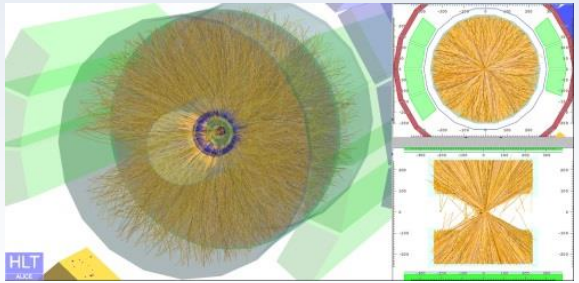
- **The TPC readout frequency is limited by the architecture.**
  - HLT TPC tracking still works at the maximum possible frequency.

- **Performance of HLT TPC tracking sufficient for any possible data taking during Run2.**

- **We still try to improve for Run 3, and test this for Run 2.**

Summary (current ALICE Tracking)

- **HLT Tracking 15 times faster than offline tracking.**
- **With GPU additional speedup of 10 compared to CPU → Total speedup 150.**
- **GPU and CPU results consistent and reproducible.**
- **GPU Tracker runs on CUDA, OpenCL, OpenMP – one common shared source code.**
- **Now: 180 compute nodes with GPUs in the HLT**
  - First deployment: 2010 – 64 GPUs in LHC Run 1.
  - Since 2012 in 24/7 operation, no problems yet.
- **Cost savings compared to an approach with traditional CPUs:**
  - About 500.000 US dollar during ALICE Run I.
  - Above 1.000.000 US dollar during Run II.
  - Mandatory for future experiments, e.g.. CBM (FAIR, GSI) with >1TB/s data rate.



FUTURE PLANS

Possible Future Improvements

- **Move more reconstruction steps onto the GPU.**
  - This will perhaps not decrease time to solution, but reduce CPU usage.

- **Alice TPC is operated at 500 Hz to few kHz during Run 2.**
  - Current GPU tracking is fully sufficient for all data the TPC can deliver in Run 2.
- **Situation changes in Run 3:**
  - Target interaction rate is 50 kHz.
  - Number of GPU will not increase by 25x.
    - We should process multiple events on the GPU concurrently.
    - Better exploition of event-base parallelism.

Concurrent event processing

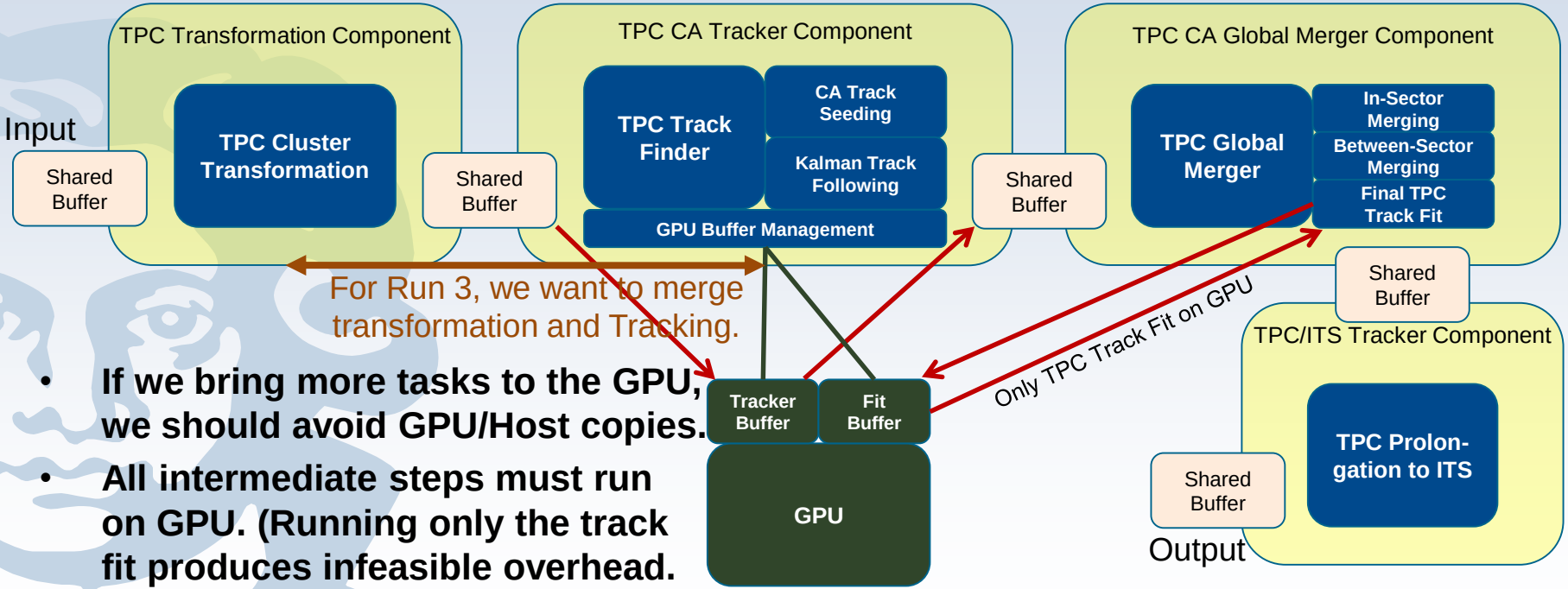
- **GPU Memory usage of TPC tracking is below 1 GB.**
- **GPUs in ALICE HLT have 6 GB.**

- **We can run multiple instances of the GPU tracker on multiple events in parallel (without further tuning).**
  - GPU parallelization also over events, on top of tracks / clusters.
  - Tracking time of 1 instance: **145 ms** (Full central PbPb).
  - Tracking time of 2 instances: **220 ms** (**110 ms** / event).
  - Speedup because of better GPU resource usage. Even a full central PbPb event can no longer utilize all ALUs of modern GPUs (this was different some years ago when we started to use GPUs in the HLT).

    → The speedup is much larger for smaller events.
  - **With this approach, the HLT can already process around 40.000.000 tracks / second (compute-wise).**

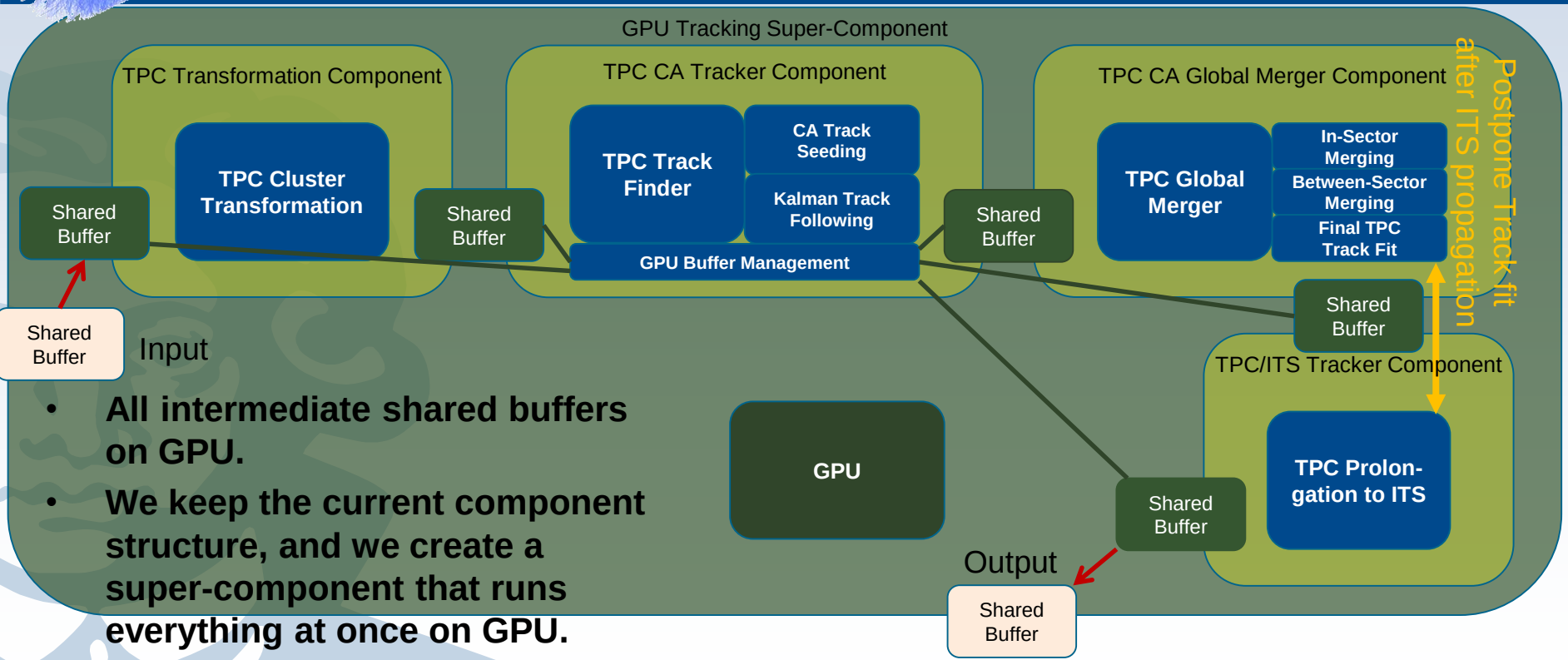
- **At very high rates, processing / sending all events individually might be inefficient.**
  - E.g. ALICE HLT framework is designed to operate at up to 6 kHz
  - It is better to combine multiple events, and process them jointly.
  - ALICE will inherently do this, due to time frames in continuous read out.

Current HLT TPC / ITS Tracking



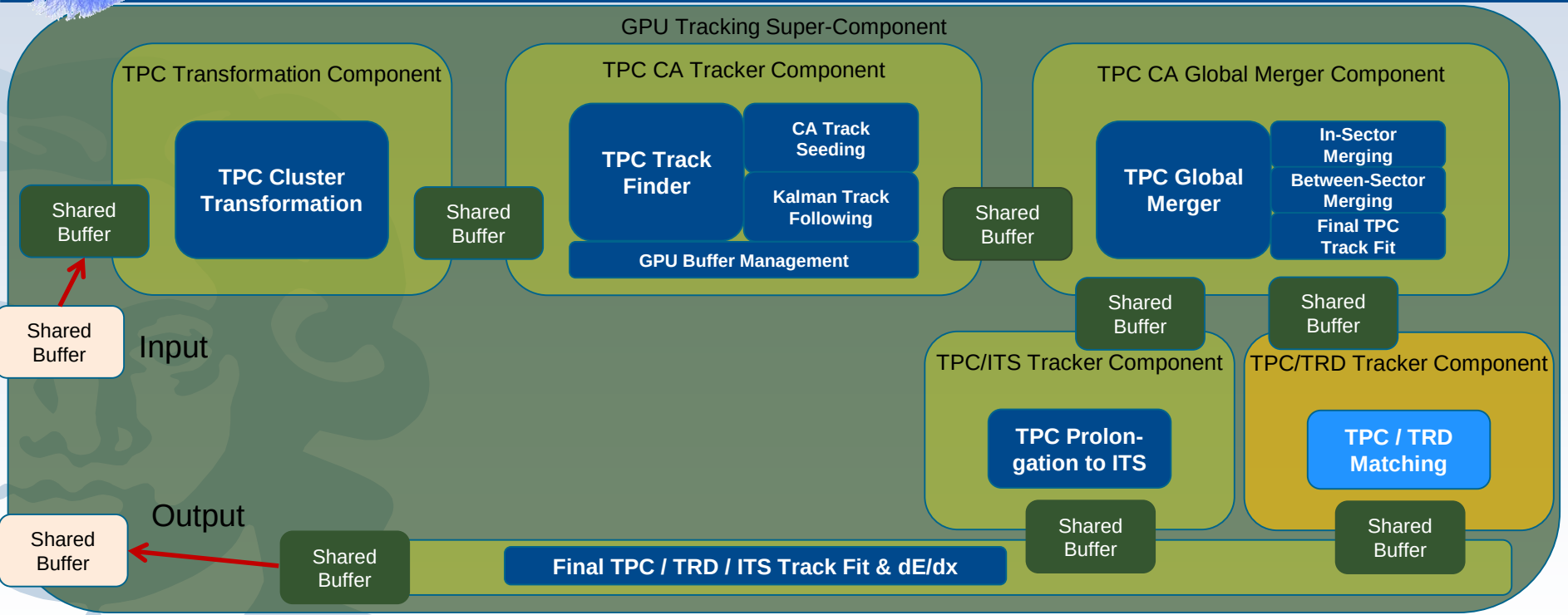
- **If we bring more tasks to the GPU, we should avoid GPU/Host copies.**
- **All intermediate steps must run on GPU. (Running only the track fit produces infeasible overhead.**
- **We have to evaluate whoch (consecutive) components can use GPU efficiently.**
  - The entire tracking chain seems a good candidate.

Next developments in tracking



- **All intermediate shared buffers on GPU.**
- **We keep the current component structure, and we create a super-component that runs everything at once on GPU.**

Next developments in tracking



- **TRD prolongation could run in parallel to ITS prolongation, final track fit afterward.**
- **We could add dE/dx to final track fit**

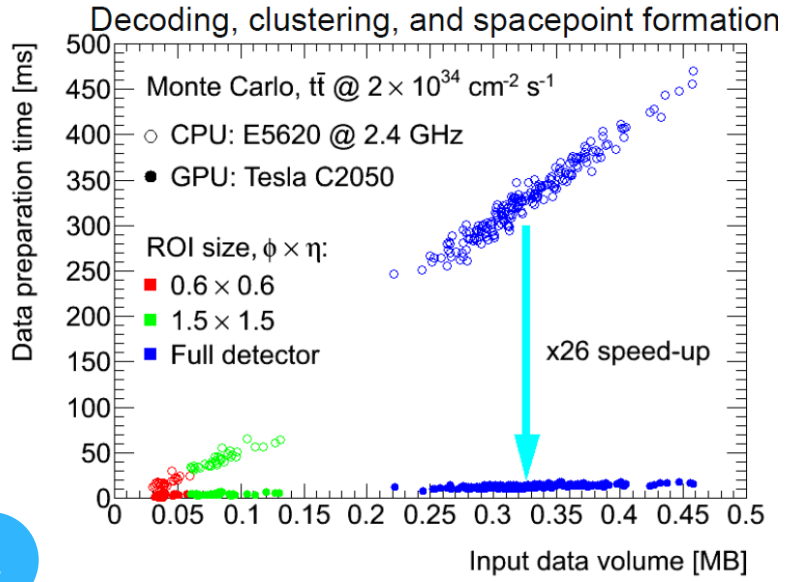
FIAS Frankfurt Institute for Advanced Studies

A BRIEF LOOK AT OTHER EXPERIMENTS

ATLAS Trigger GPU Prototypes

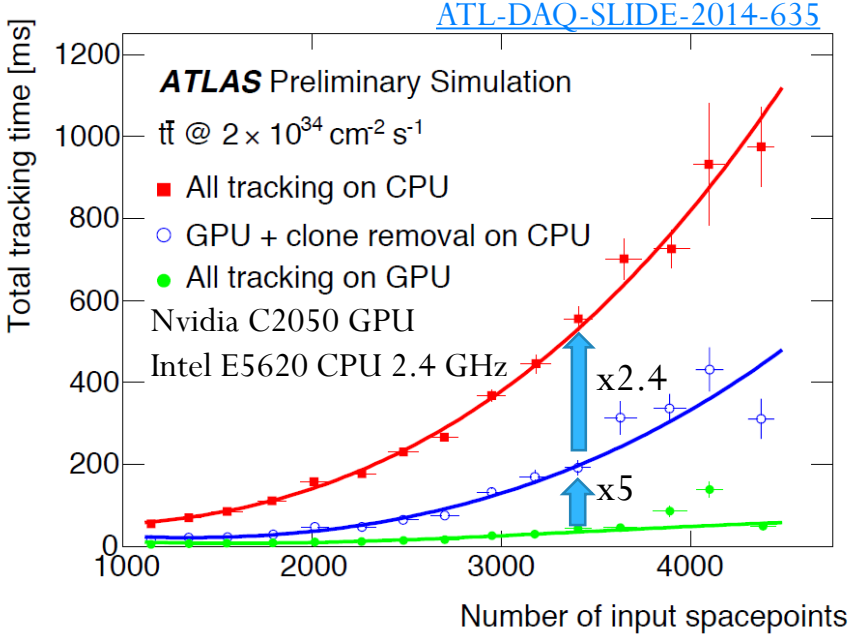
- **GPU prototypes developed to quantity benefits of GPGPU for ATLAS Trigger:**
  1) **ID prototype:** Complete Inner Detector (ID) trigger chain implemented; measurements made on C2050
  2) **Extended prototype:** ID & Muon Tracking, Calorimeter clustering – measurements on K80
- Goal is to assess the potential benefit in terms of event though-put per unit cost.
- **ID Prototype comprises:**
  - Data Preparation: Bytestream decoding, Clustering, Space-Point formation
  - Tracking: Seed-making, track extension, clone removal
- Average factor 12 speedup for whole tracking chain on C2050 GPU c.f. 1 E5620 CPU core
  - Speed-up greater for some algorithms: Factor 26 for data preparation

	Nvidia C2050	Nvidia K80
Cores	448	2x2496
Multiprocessors	14	2x13
Cores/SMX	32	192



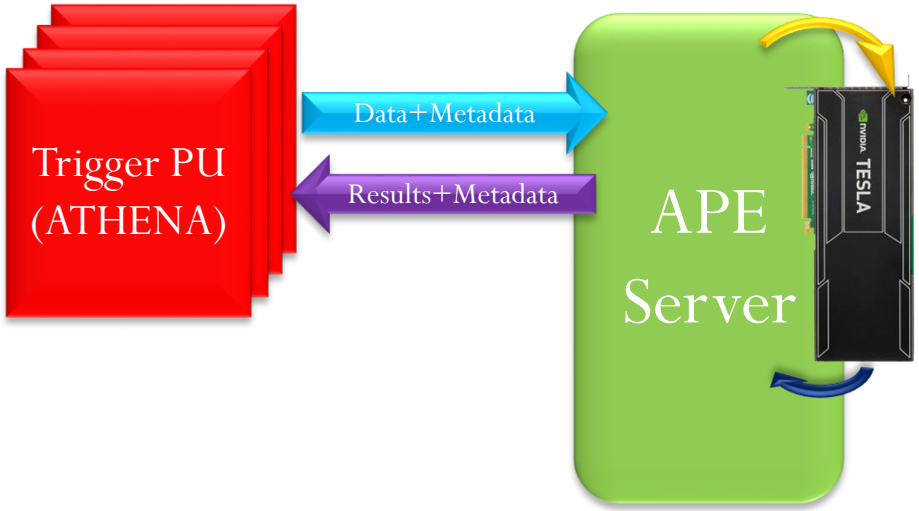
D Emeliyanov and J Howard 2012 J. Phys.: Conf. Ser. 396 012018

ATL-DAQ-SLIDE-2014-635



ATLAS Offloading Mechanism

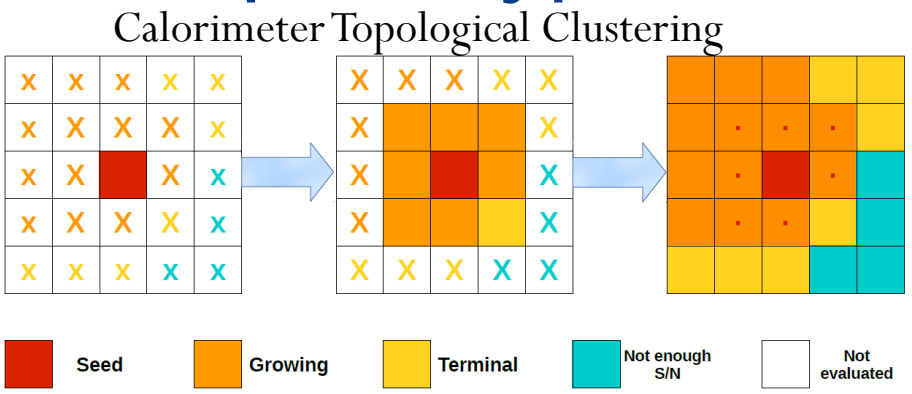
- A client-server approach is implemented to manage resources between multiple PU processes.
- PU prepares data to be processed and sends it to server
- Accelerator Process Extension (APE) Server manages offload requests and executes kernels on GPU(s)
- It sends results back to process that made the offload request



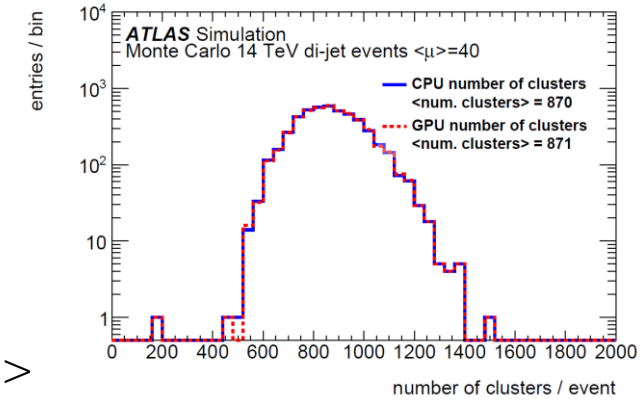
Server can support different hardware types (GPUs, Xeon-Phi, CPUs) and different configurations such as GPU/Phi mixtures and in-host off-host accelerators.

ATLAS Extended GPU prototype

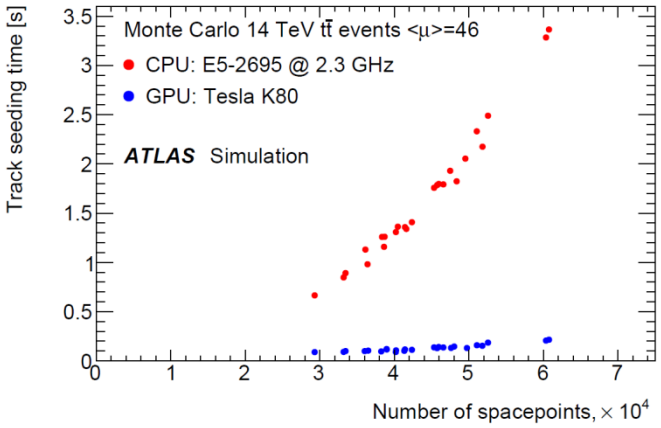
- **Extended prototype comprising:**
  - Inner Detector Tracking
  - Calorimeter Topological Clustering
  - Muon Tracking : Based on Hough Transform



- **Initial measurements:**
  - Comparison of GPU and CPU algos:
    - Reconstructed quantities e.g. no. Calo clusters =>



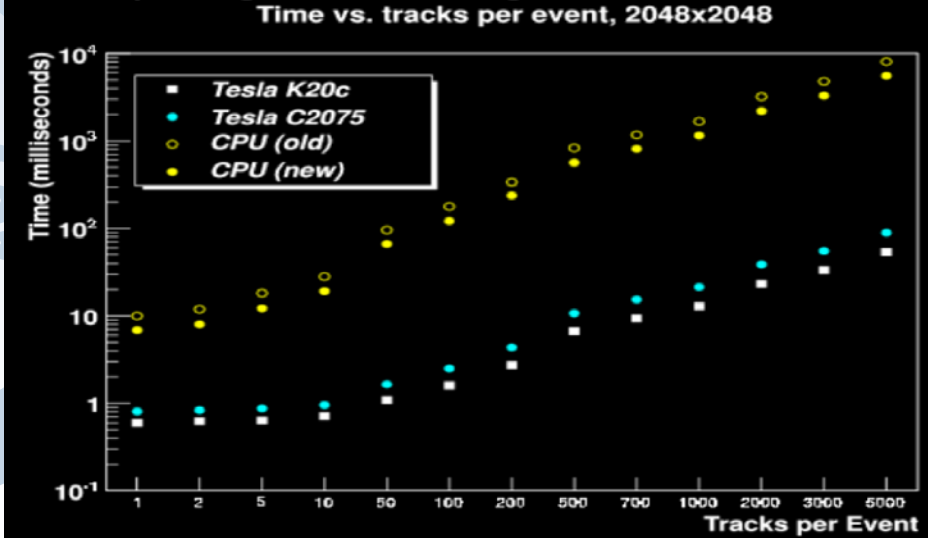
    - Execution times e.g. Track Seeding algorithm time =>



    - Throughput: events/s:
      - Initial results suggest factor of two increase in system throughput could be obtained by adding GPU
      - To be confirmed with ongoing measurements

CMS – GPU Implementation

- Hough transform is a **natural candidate** for GPU acceleration using general-purpose GPU programming with CUDA.



CPU implementation before (open) and after (filled) optimization (performed on Intel Core i7-3770)

GPU implementation on Tesla C2075 and K20c
–10-60x faster!

- Also a candidate for investigating with Xeon Phi

See **arXiv:1309.6275** for more on these implementations

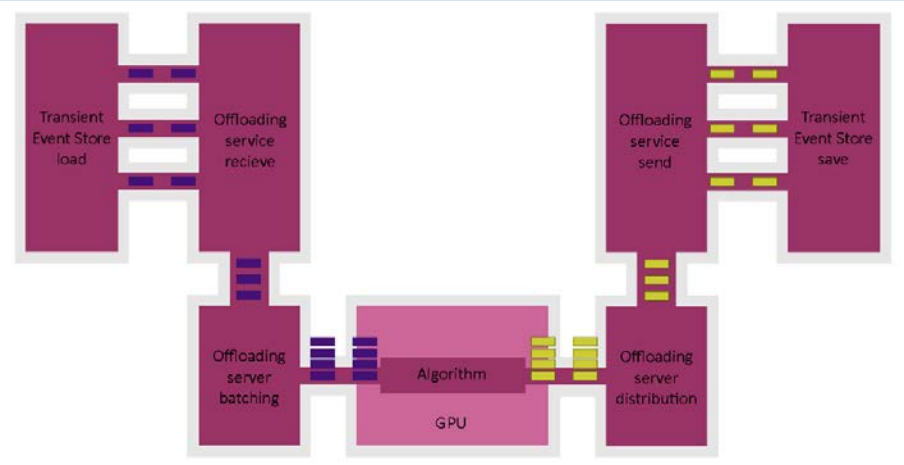
LHCb – GPU Manager Offload Tool

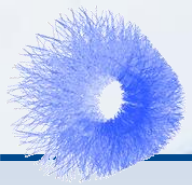
Socket client-server tranmission

Scheduler First-Come First-Served, gathers multiple events and ships them for concurrent processing

Some goodies

- Algorithm exceptions propagated to callers
- Centralized profiling, logging
- File input / output configurable
- Outside framework execution possible





QUESTIONS?