ALICE HLT TPC Tracking on GPUs

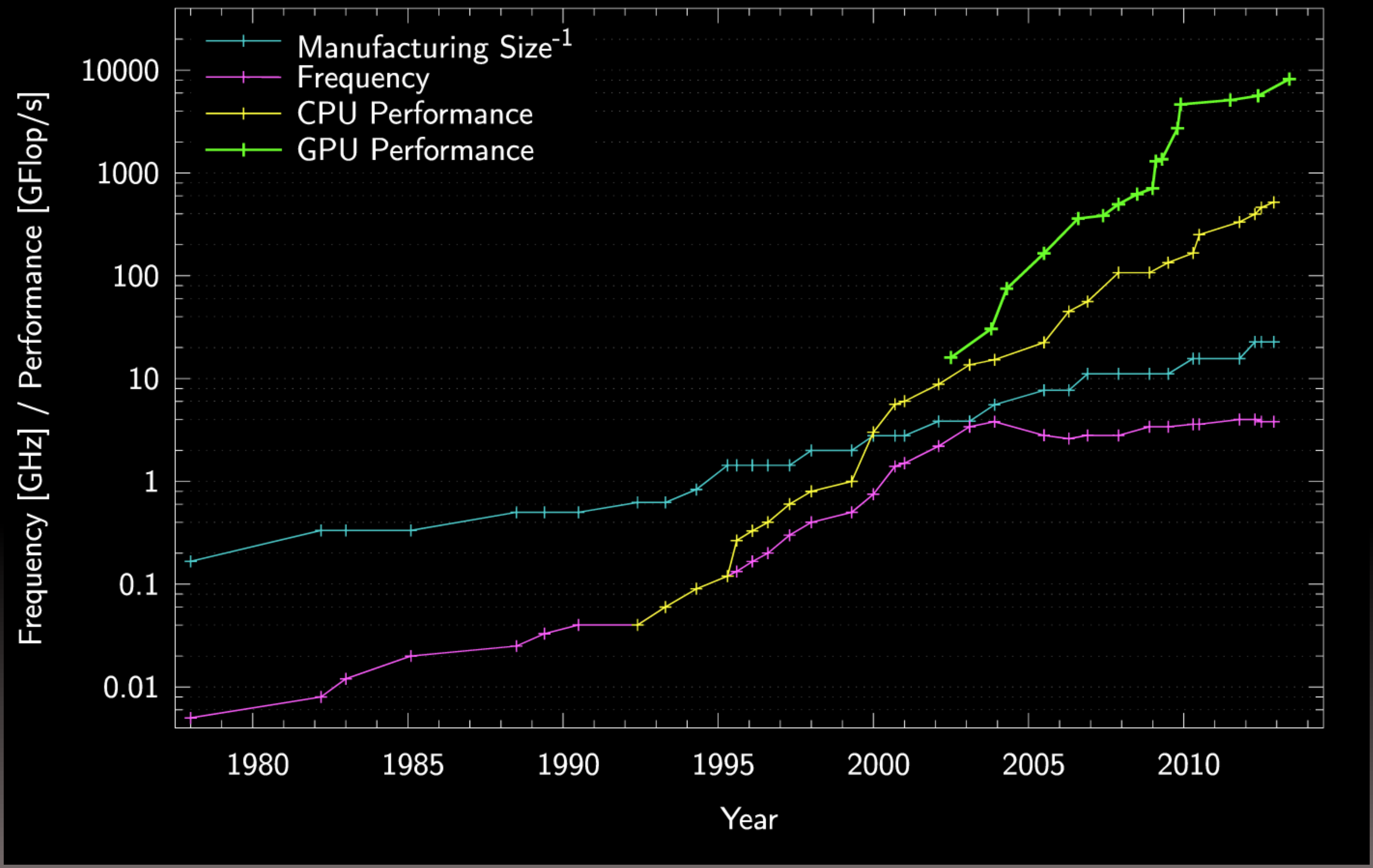
I: GPUs
II: Integration of GPUs in ALICE Framework
III: GPU-based track reconstruction in ALICE
IV: ALICE CPU / GPU Tracker Comparison
V: ATLAS
VI: CMS
VII: LHCb

David Rohr
CERN – 5.9.2014

FIAS Frankfurt Institute for Advanced Studies

GPUS

Performance Development



Why GPUs

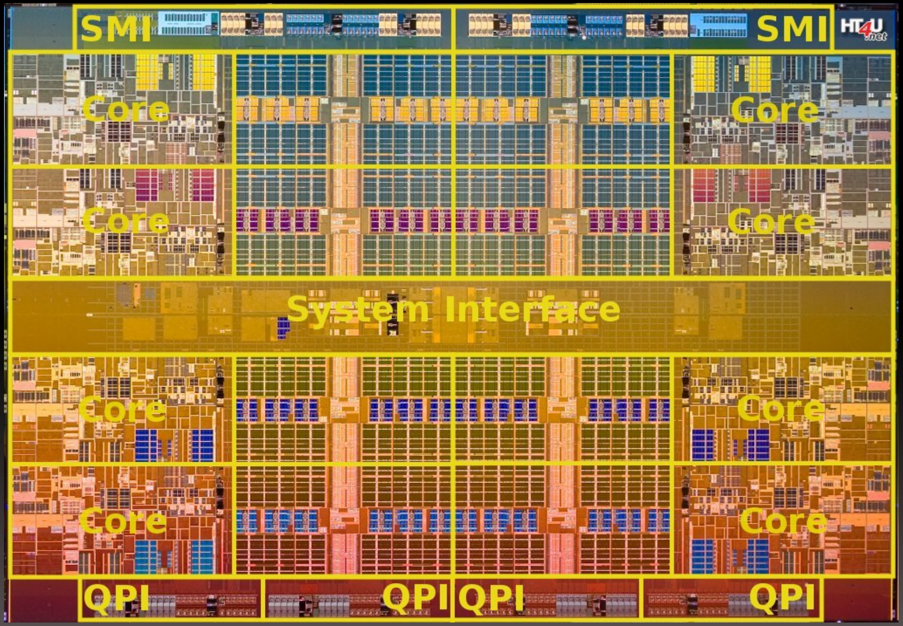
- CPUs are designed for fast execution of serial programs.
  - Clocks have reached a physical limit.

    → Vendors use parallelization to increase performance.
- GPUs are designed for parallel execution in the first place.
  - The „only“ limit for GPU performance is heat dissipation.
  - GPU clocks are usually lower than they could be.
    - This saves power
    - Hence more hardware can be powered in parallel

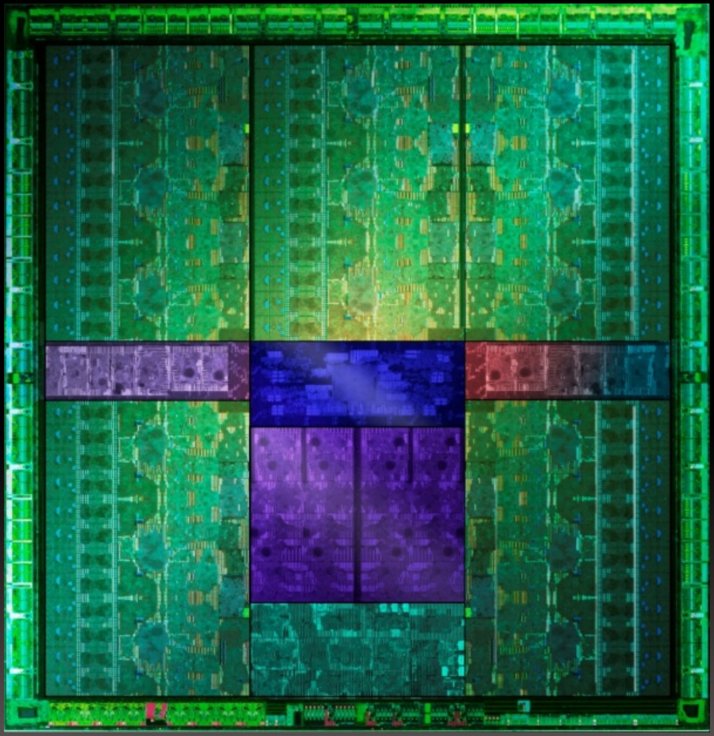
    → Better overall performance

Why GPUs

- GPUs use their silicon for Aus
- CPUs use their silican mainly for caches, branch prediction, etc.



Intel Nehalem



NVIDIA Kepler

Why GPUs

- Some number os the hardware:

Hardware	Cores	Clock Rates	ALUs (Single Precision)
Nehalem	4-6	2-3.6 GHz	48
Sandy Bridge	4-8	2-3.6 GHz	128
Ivy-Bridge	4-12	2-3.6 GHz	196
Magny-Cours	6-12	1.8-2.4 GHz	96
Interlagos	8-16	2-2.6 GHz	128
NVIDIA GTX285	32	1476	240
NVIDIA GTX580	16	1544	512
NVIDIA Kepler	16	1006	2688
AMD Cypress	20	850	1600
AMD Cayman	24	880	1536
AMD Graphics Core Next	32	950	2048

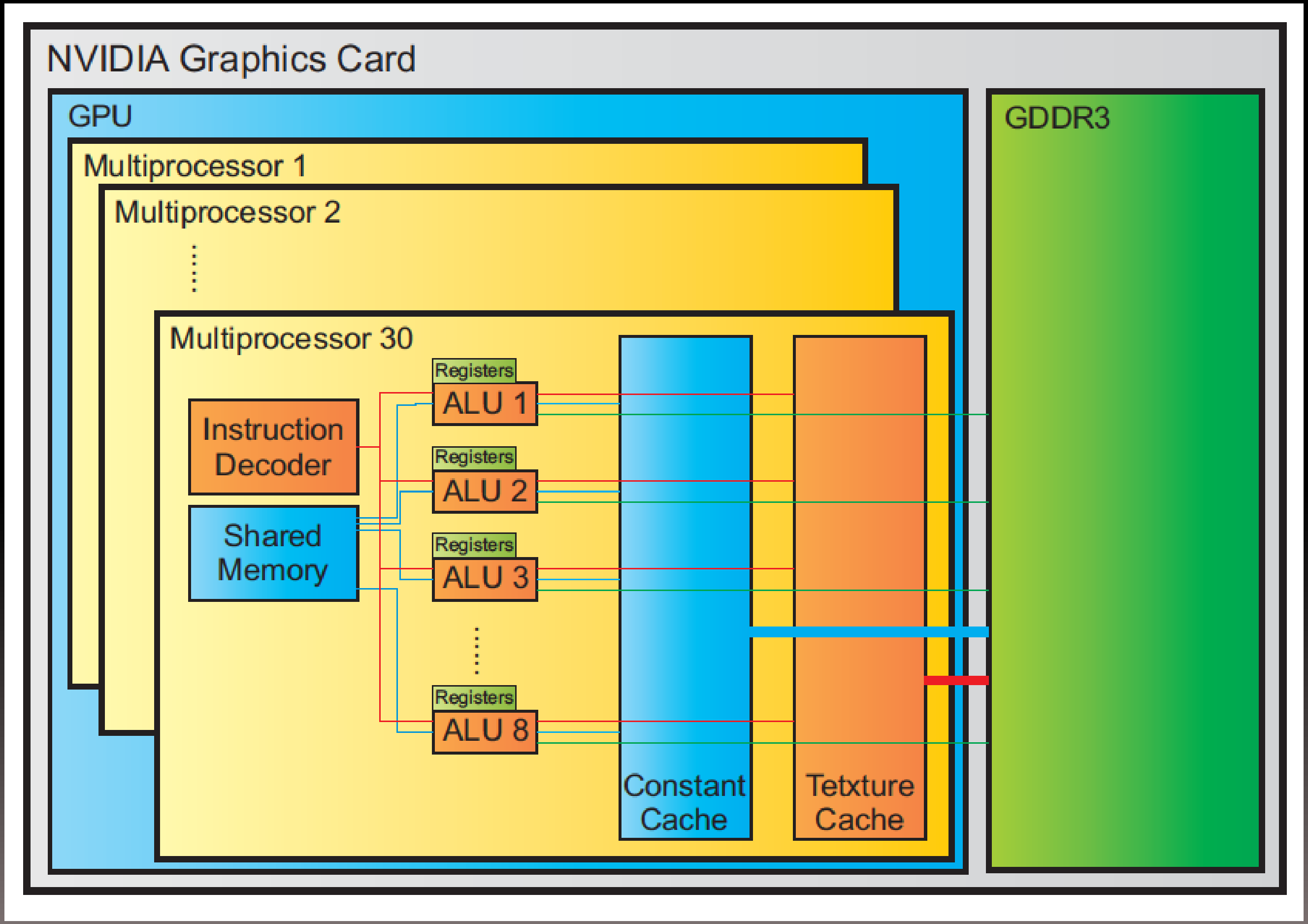
Why GPUs

- Some performance numbers:

Hardware	Peak Perf. (Single)	Peak Perf. (Double)	Mem. Bandwidth
Nehalem, 4 Cores, 3 GHz	96 Gflop/s	48 Gflop/s	38 GB/s
Sandy Bridge, 8 Cores, 3 GHz	384 Gflop/s	192 Gflop/s	51 GB/s
Haswell, 4Cores, 3 GHz	384 Gflop/s	192 Gflop/s	58 GB/s
Magny-Cours, 12 Cores, 2 GHz	192 Gflop/s	96 Gflop/s	42 GB/s
Interlagos, 16 Cores, 2.4 GHz	307 Gflop/s	154 Gflop/s	51 GB/s
NVIDIA GTX285	714 Gflop/s	89 Gflop/s	159 GB/s
NVIDIA GTX580	1581 Gflop/s	198 Gflop/s	192 GB/s
NVIDIA Kepler	3950 Gflop/s	1310 Gflop/s	250 GB/s
AMD Cypress	2720 Gflop/s	544 Gflop/s	154 GB/s
AMD Cayman	2703 Gflop/s	675 Gflop/s	176 GB/s
AMD Graphics Core Next	3789 Gflop/s	947 Gflop/s	264 GB/s

Introduction

NVIDIA GTX280 GPU



Challenges

- Keeping GPU utlization high
  - Hide DMA transfer times, make use of vector units.
- Many frameworks work on a per-event basis
  - One event might contain too less data to exploit GPU parallelism
- Offline compute centers use heterogeneous hardware
  - Need to be vendor-independent
- Event reconstruction / analysis consists of many tasks
  - GPU must be shared among these tasks
- Large effort to maintain multiple variants of the source code
  - One should use a common source code where possible
- Huge effort to port all code to GPU
  - One should find computational hotspots, and port only those

INTEGRATION OF GPUS IN ALICE FRAMEWORK

Integration

- AliRoot bases on C++.
  → GPU kernel language must support C++.
- In 2010 (Start of the project), CUDA was the only such language.
- Today, there are C++ kernel language extensions for OpenCL by AMD.
  - AMD is pushing to get this into the next OpenCL standard.
  - Unfortunately, it did not make it in the 2.0 Specs.
- Can run on CPU / Xeon Phi (C++, OpenMP), NVIDIA GPU (CUDA), AMD GPU (OpenCL)
- TPC track finding responsible for 50% of compute resources
  - We run only TPC track finding on GPU
  - Optionally, we could also run track fit (10% of compute resources)

Integration

- GPU and CPU tracker (CUDA and OpenCL) share a 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);
  }
}
```

gpu_wrapper.cpp:

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

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

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

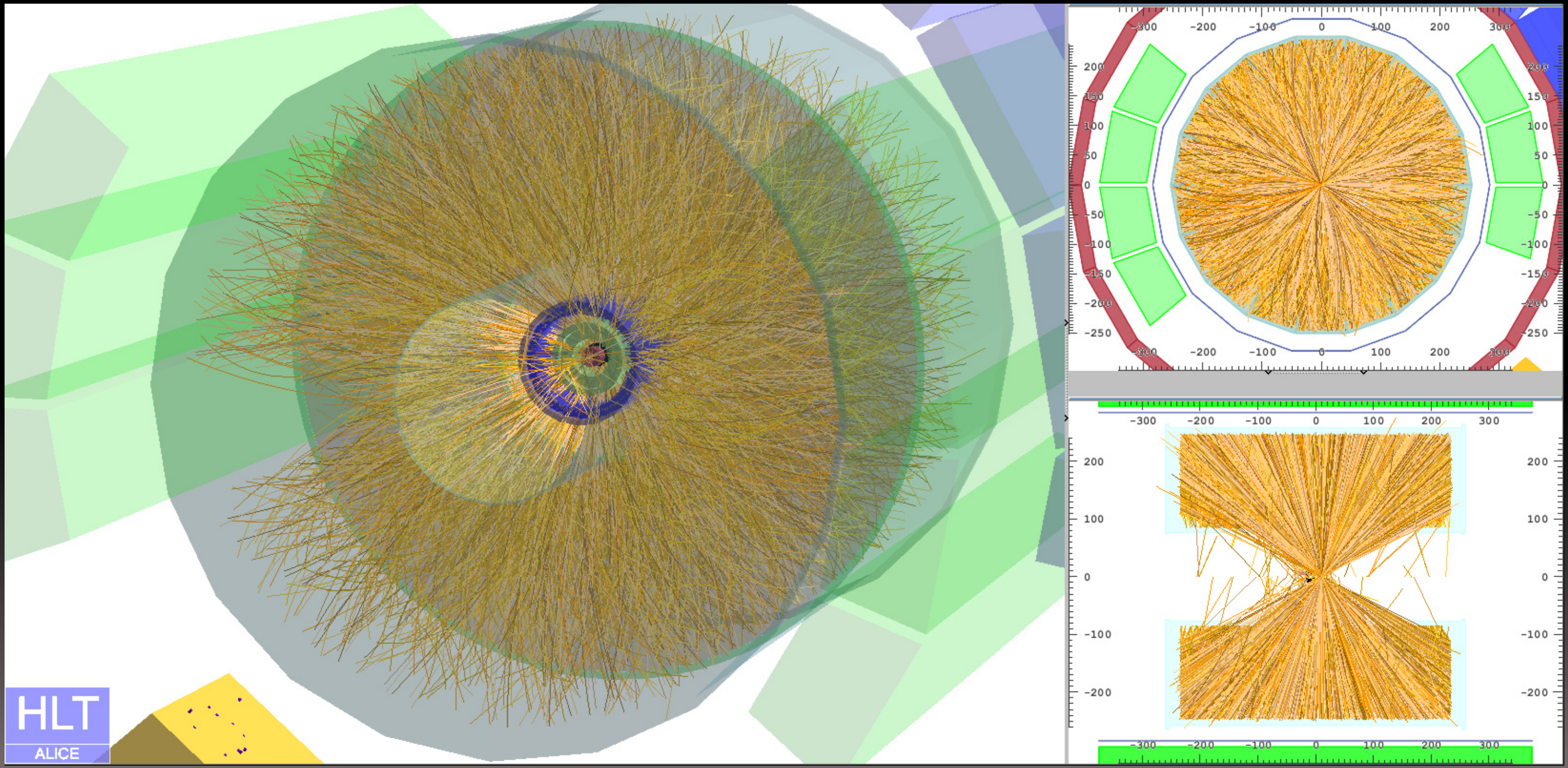
Integration

- The GPU Tracker is accessed via a virtual interface. The actual implementation is contained in a dedicated library (cagpu), which links against the CUDA runtime.
- AliRoot opens cagpu with dlopen, this creates a clear separation between AliRoot and CUDA.
- The same AliRoot binaries can be used on compute nodes with GPU and without GPU.
- This scheme is easily adoptable to other programming APIs, such as OpenCL.

GPU-BASED TRACK-FINDING IN ALICE

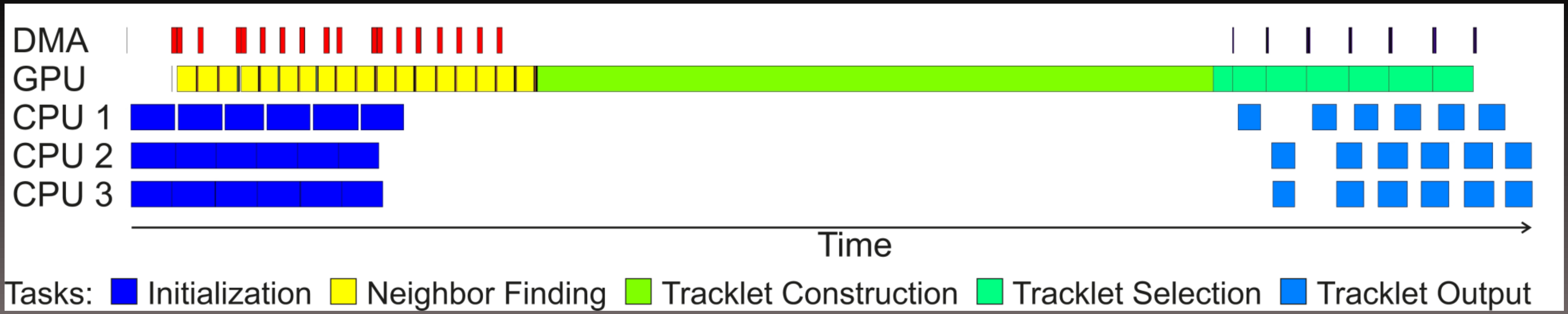
Introduction

Screenshot of ALICE Online-Event-Display
during first physics-fill with active GPU Tracker



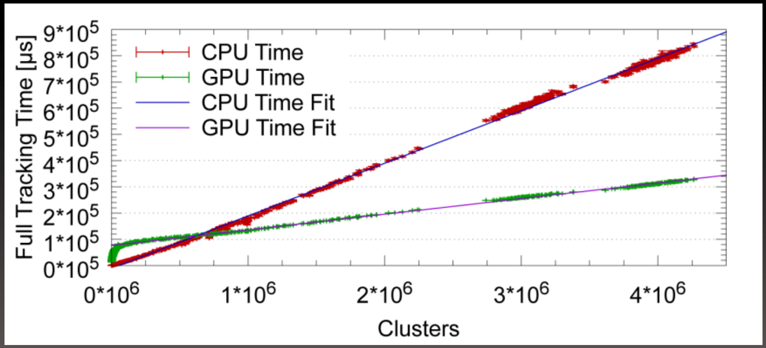
GPU Tracker Performance

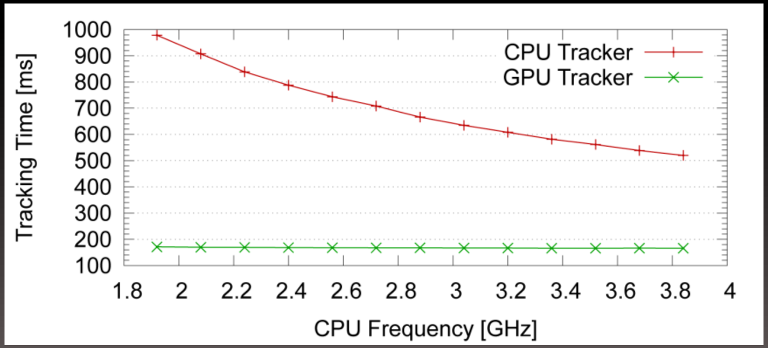
- For good performance the GPU tracker pipelines the slices such that initialization on CPU, GPU tracking, and DMA transfer can overlap.
- Multiple CPU cores are required to feed the GPU with sufficient inut data.



GPU Tracker Performance

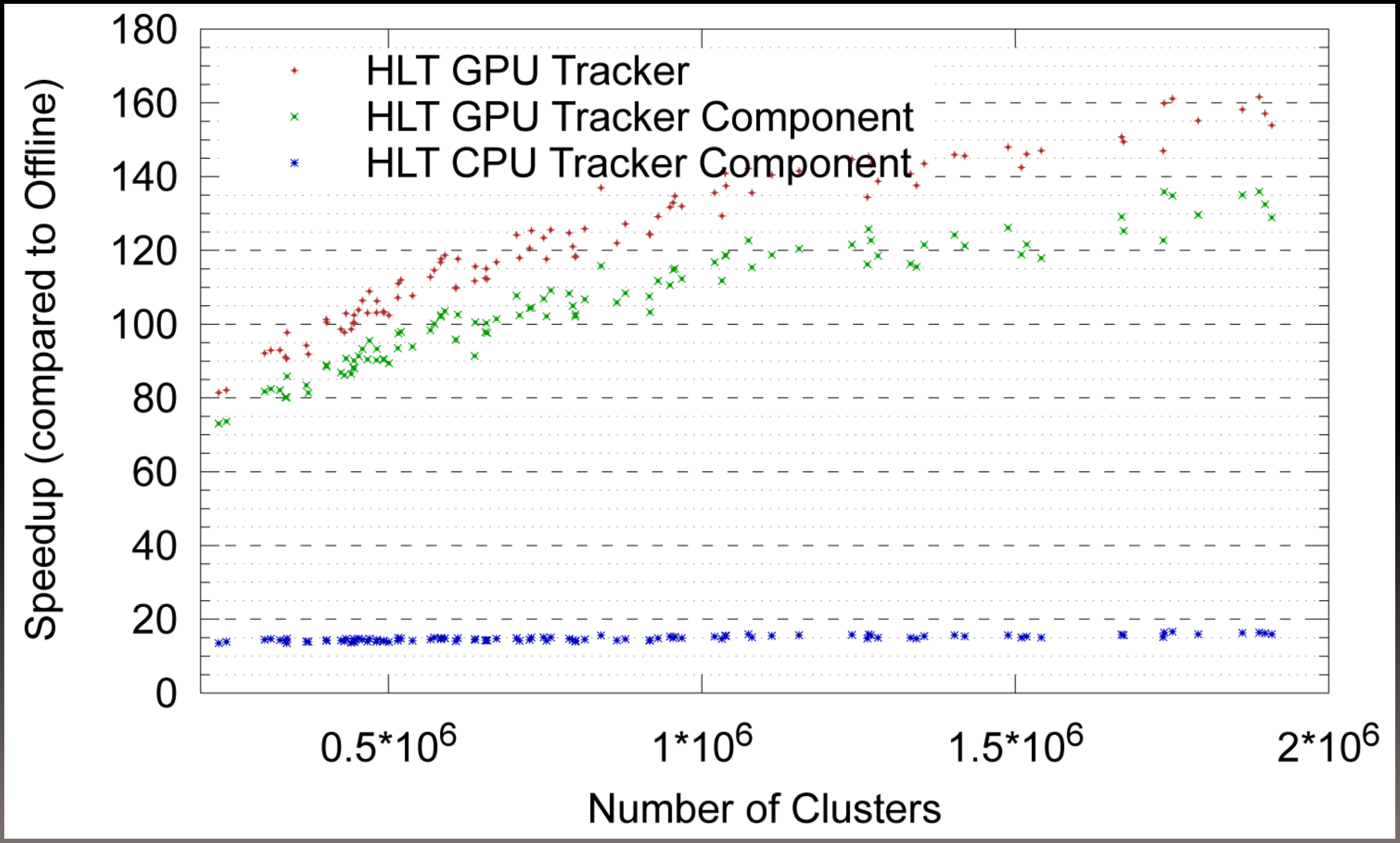
- Tracking time depends linearly on input data size.
- GPU tracking time independent from CPU performance (if initialization is fast enough).





GPU Tracker Performance

- Speedup of HLT GPU tracker v.s.offline and CPU Tracker (four CPU cores used each)



CPU / GPU Tracker Comparison

CPU / GPU Tracker Comparison

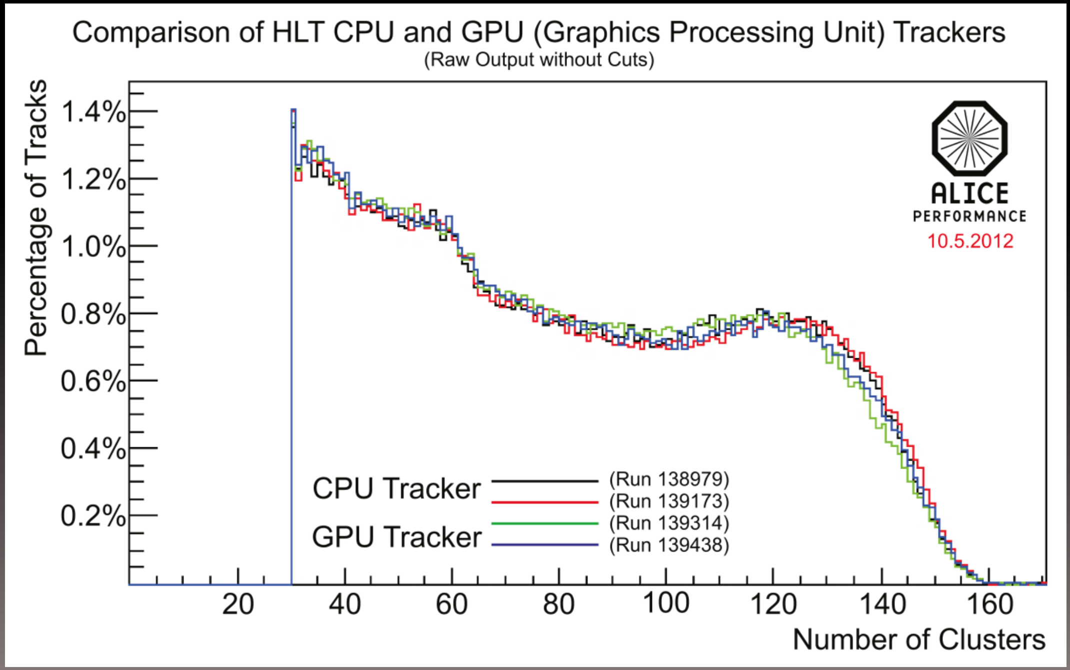
- Comparison of GPU and CPU Tracker during 2010 run
  - No significant variations in physically observables.
  - Only the number of clusters per track statistics shows a variation.

CPU / GPU Tracker Consistency

- Inconsistencies during November 2010 run
  - Cluster to track assignment differs.
    (Differences caused by concurrent track-finding)

    SOLVED
  - Non-associative floating point arithmetics

    NEGLIGIBLE



Usage of the GPU Tracker

- The GPU tracker was deployed and commisioned in the ALICE HLT farm in fall 2010.
  - 64 GPU enabled compute nodes eqiped with NVIDIA Fermi GPUs have been installed.
  - Some bottlenecks in the framework had to be solved, before the GPU tracker could run at full rate.
  - GPU tracker ran throughout the entire year 2012 without incident.
- The upgraded ALICE HLT farm after LS1 bases on the GPU tracker, with more recent GPUs.
  - We employ 180 AMD S9000 GPUs

Results on current hardware

- GPU tracking time on exemplary PbPb event.
  - NVIDIA Fermi (current version) 174 ms
  - NVIDIA GTX780 (Kepler) 155 ms
  - NVIDIA Titan (Kepler) 146 ms
  - AMD FirePro 160 ms
- Current Generation GPUs (Kepler / GCN) offer new features.
  - We assume approx. 20% performance gain by adapting the tracker
- In the future, we might run into CPU limitations.
  - Current design with CPU-based initialization and outpat phase should be reevaluated.

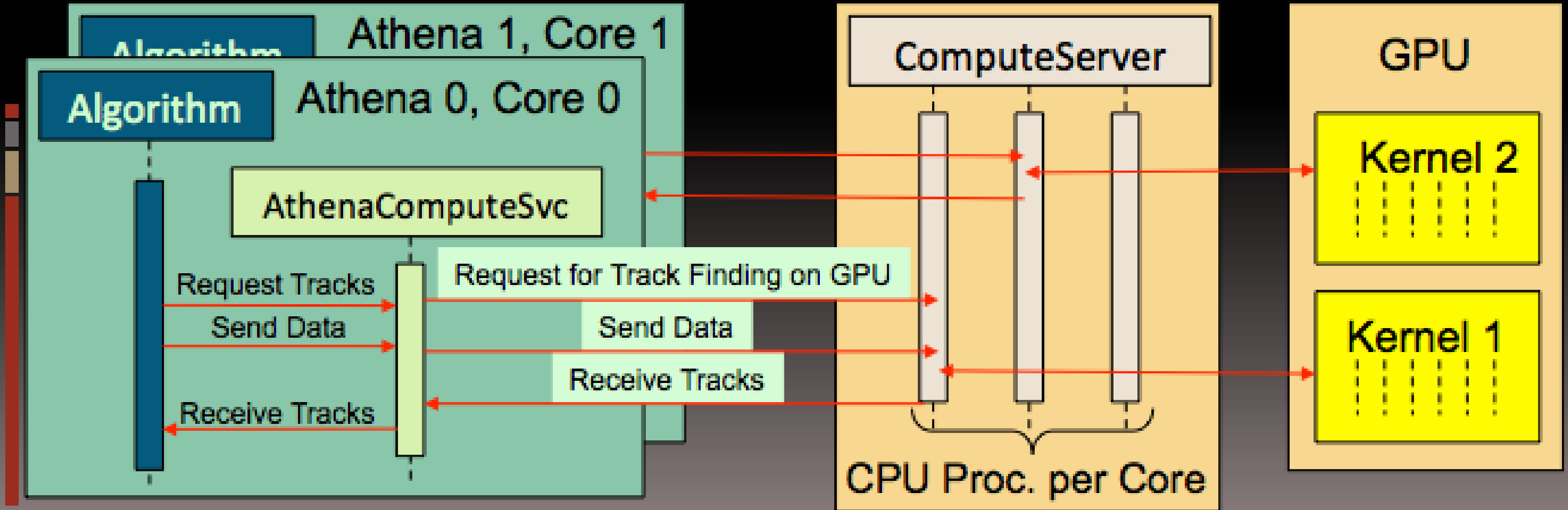
Summary

- Threefold performance increase of GPU tracker compared to all CPUs of a node, tenfold increase in a reasonable HLT scenario.
- GPU tracker performance is independent from CPU and depends linearly on data size.
- Results of GPU and CPU tracker match almost completely. Only 0.00024% of the clusters differ due to non-associative floating-point arithmetic.
- Common source code ensures great maintainability, separation from libAliHLTTPC makes a common binary work on all nodes – with and without GPU.
- GPU tracker has been employed sucessfully in the recent PbPb runs and is employed in the new HLT cluster after the shutdown.

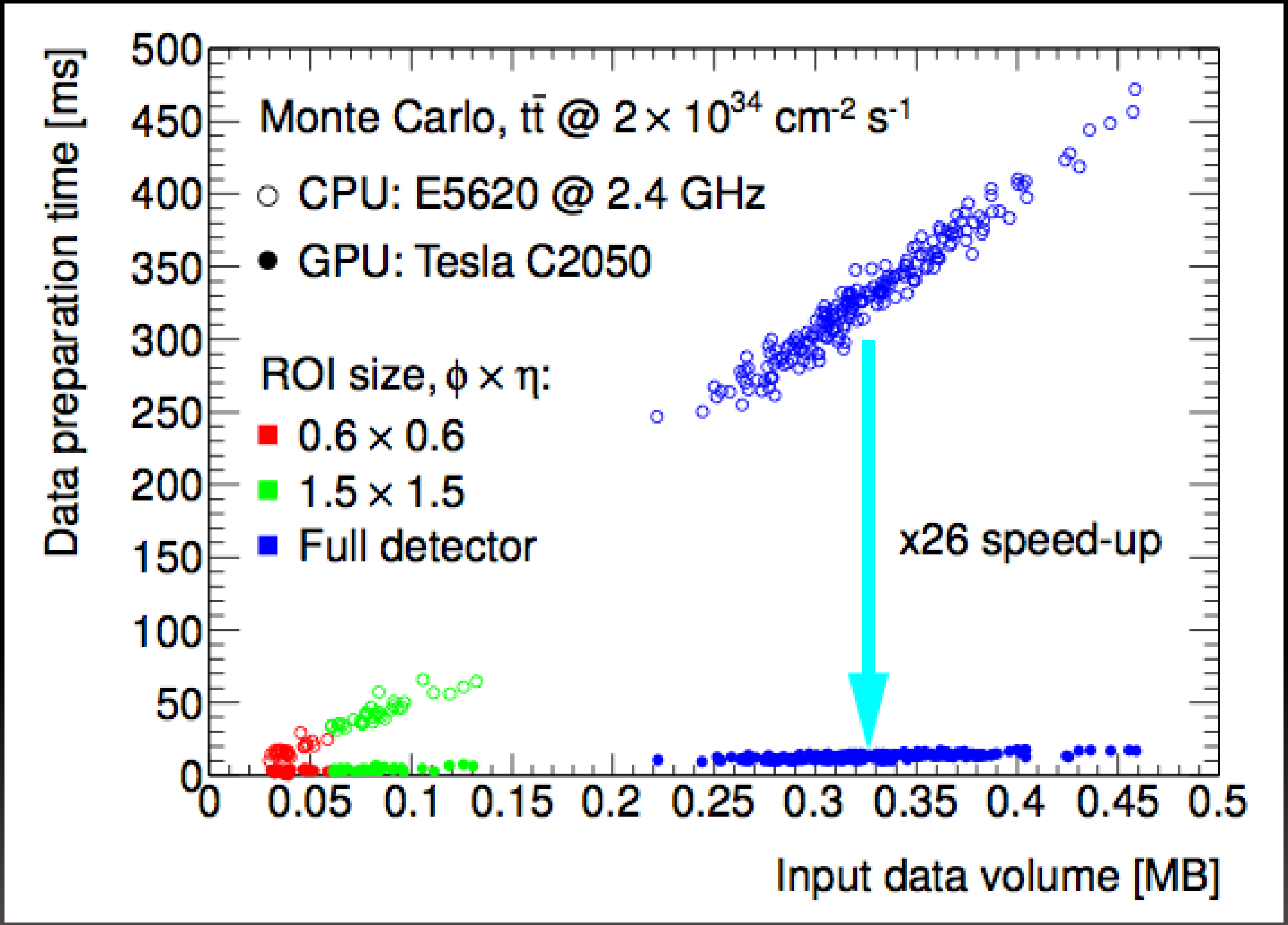
ATLAS

Client-Server Architecture

- Client-server architecture allows GPU resources to be shared amongst multiple trigger instances
- Data transfer is done over shared memory segment
  - Also used as CUDA host buffer
- Minimizes integration surface in trigger software - only POSIX required
- Allows for GPU memory resources (e.g. hardware maps) to be shared

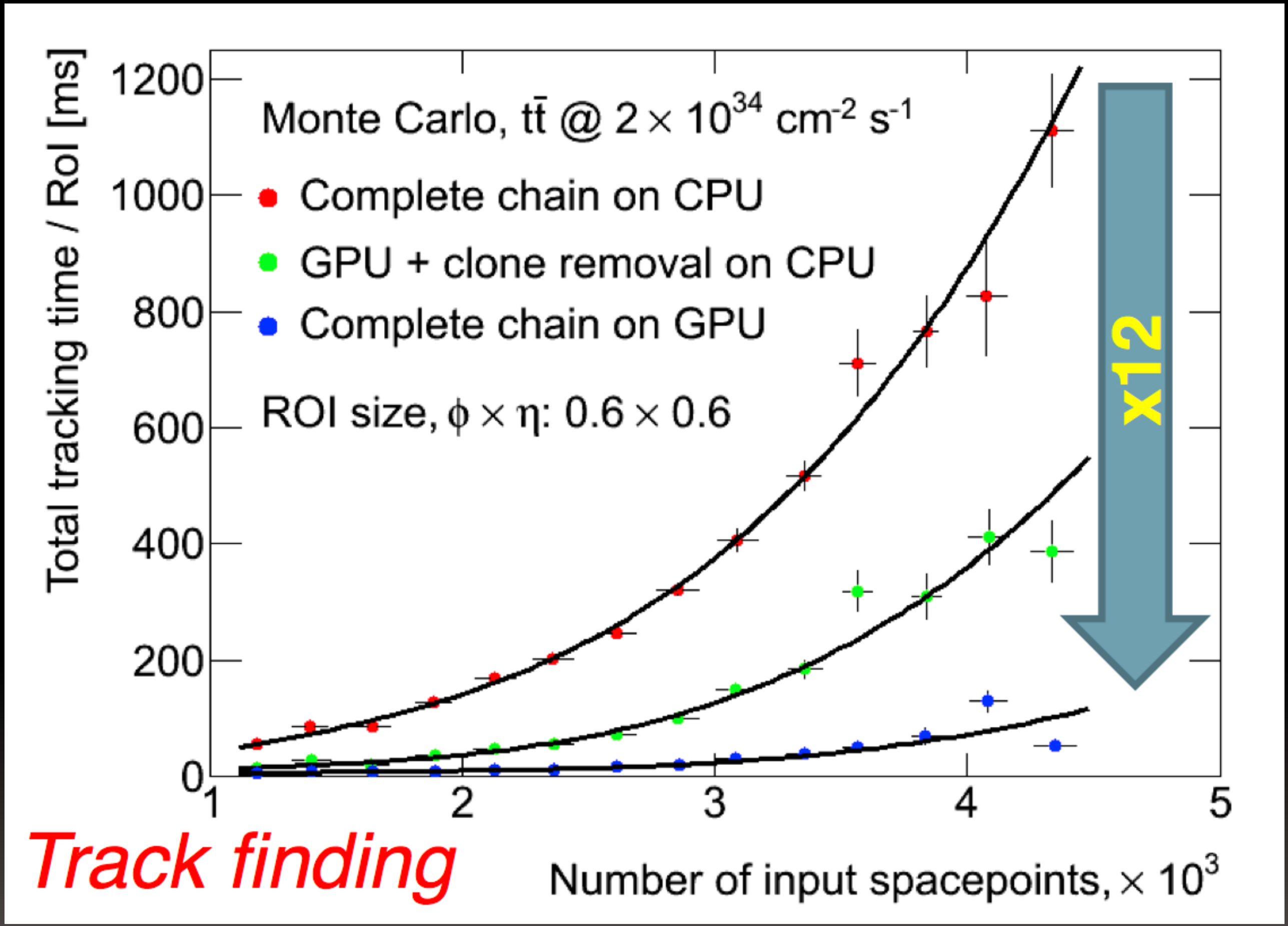


Data Preparation Results



Bytestream decoding and clustering show a **26x** speed-up against single-threaded CPU

Tracking Results



Track formation and clone removal show a **12x** speed-up against single-threaded CPU

OpenCL Studies

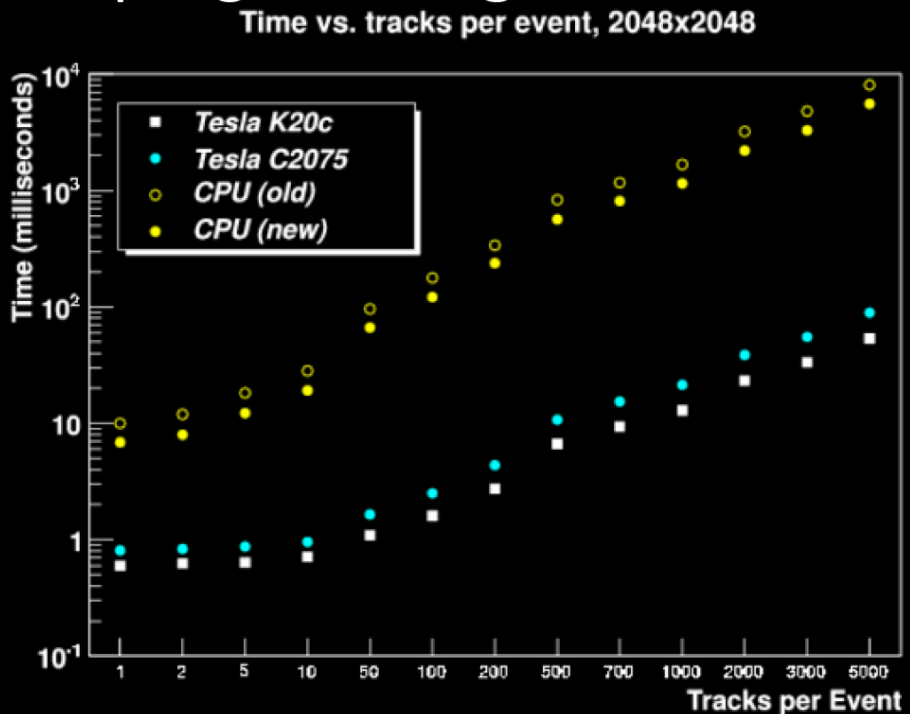
- The CUDA implementation has been ported to OpenCL
- Initial performance comparisons show encouraging results on GPU, ~15% performance loss

Platform	C2050 (CUDA)	C2050 (OpenCL)
Pixel Processing	3.2 ms	3.9 ms
SCT Processing	3.6 ms	4.0 ms
Total Processing	6.8 ms	7.9 ms

CMS

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!*

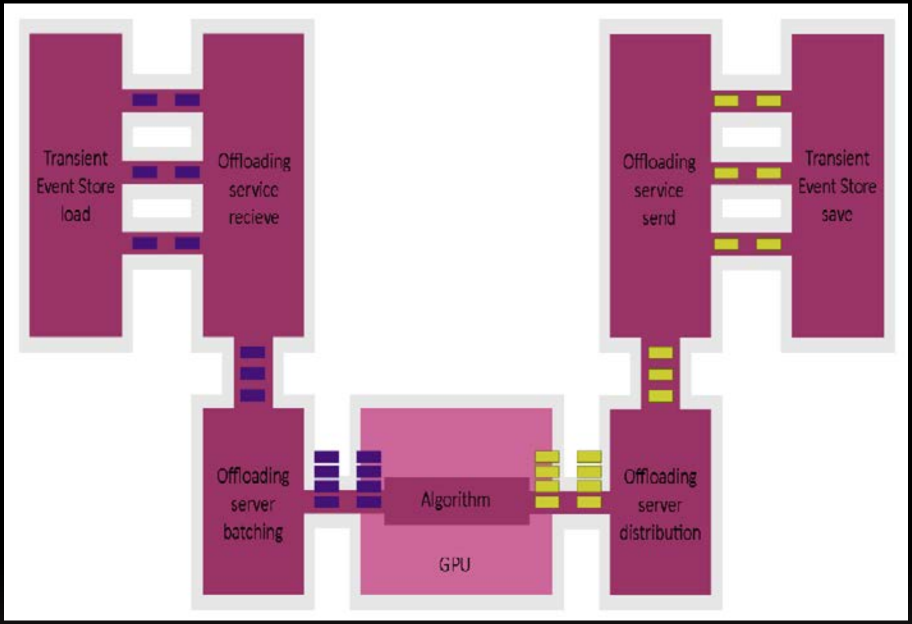
- Also a candidate for investigating with Xeon Phi

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

LHCB

LHCb – GPU Manager
Gaudi tool to offload algorithms

- 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



Manycore on LHCb

- LHCb tracking
  - FastVELO
    - Local method (Track Forwarding), 2x over sequential version Currently expanding into ST tracks
  - VELO Pixel (LS2 upgrade)
    - Local method, 11x over sequential version
      - Working on improvement over Physics RE
    - Hough transform implementation ongoing
  - Vertexing using graph-theory and techniques from DNA matching and social networking
- RICH
  - Prototyping Ray Tracing machinery on GPU
- All the efforts so far
  - https://lbonupgrade.cern.ch/manycore
  - **GPGPU opportunities at the LHCb trigger –** LHCb-PUB-2014-034

Summary / Challenges

- Keeping GPU utlization high
  - Hide DMA transfer times, make use of vector units.
- Many frameworks work on a per-event basis
  - One event might contain too less data to exploit GPU parallelism
- Offline compute centers use heterogeneous hardware
  - Need to be vendor-independent
- Event reconstruction / analysis consists of many tasks
  - GPU must be shared among these tasks
- Large effort to maintain multiple variants of the source code
  - One should use a common source code where possible
- Huge effort to port all code to GPU
  - One should find computational hotspots, and port only those
- Usually a speedup of around a factor 3, compared to multi-core CPU