GPU Usage in ATLAS

Charles Leggett

ATLAS ML Workshop
November 15 2019

Why The Renewed Interest In Accelerators?

- In the next generation of supercomputers we see extensive use of accelerator technologies
  - Oak Ridge: **Summit** (2018)
    - 4608 IBM AC922 nodes *w/* 2x Power9 CPU
    - 3x **NVIDIA** Volta V100 + NVLink / CPU
  - LBL: NERSC-9 "**Perlmutter**" (2020)
    - AMD EPYC "Milan" x86 only nodes + mixed CPU / "next gen" **NVidia** GPU
  - Oak Ridge: **Frontier** (2021)
    - 1.5 exaflop
    - AMD EPYC CPU + 4x **AMD** "Instinct" GPU
  - Commercial clouds:
    - Brainwave / Azure FPGA
    - Google Cloud TPU
  - LLNL: **Sierra** (2018)
    - 4320 IBM AC922 nodes *w/* 2x Power9 CPU
    - 2x **NVIDIA** Volta V100 + NVLink / CPU
  - Argonne: **Aurora A21** (2021)
    - Intel Xeon CPU + **Intel** X^{e}/gen12 GPU + Optane
  - Tsukuba: **Cygnus** (2020)
    - 2x Intel Xeon 6162+ 4x **NVidia** V100 GPU
    - 2x CPU + 4x GPU + 2x Intel Stratix FPGA
  - Japan: **Fugaku** (2021)
    - manycore ARM A64fx (48+2)
    - integrated "SVE" 512 bit GPU-like accelerator
- In order to meet the HL-LHC computing requirements, we need to use all available computing resources, or cut back physics projections
  - **US funding agencies have indicated that we will not be able to get allocations if our code does not make use of accelerator hardware**

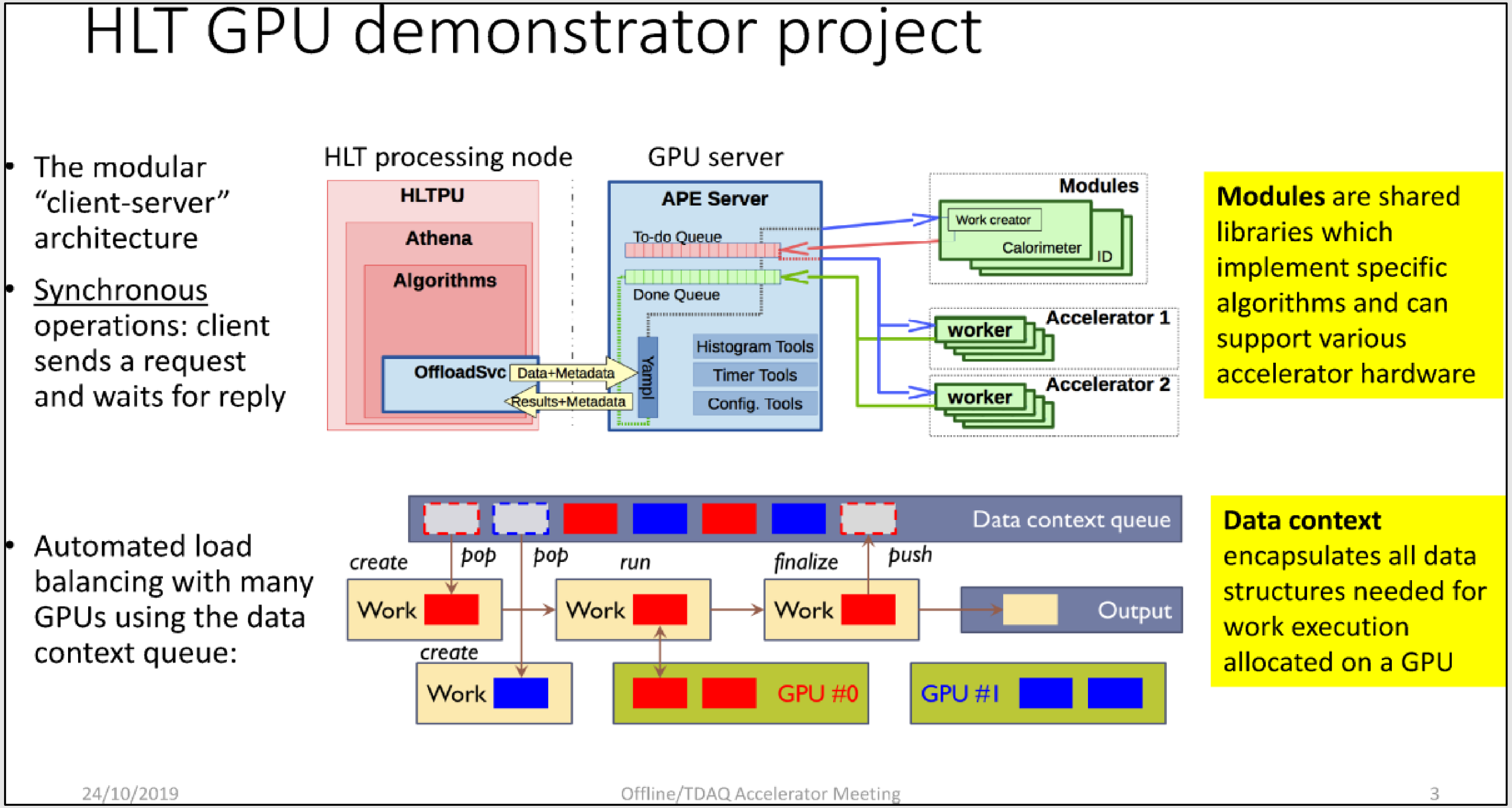
Past Usage

- Using GPUs accelerators in ATLAS is nothing new
  - they’ve been around for more than 2 decades
  - a trivial search on inSPIRE of “ATLAS” and “GPU” results in 25 papers, going back to 2011.
    - Tracking
    - Trigger
    - generic Algorithmic acceleration
    - ML acceleration
    - Event Generation
    - Visualization (doesn’t really count)

- Many other HEP experiments have dallied with GPUs
  - some have chosen to use them, some not

Trigger Studies

- Started investigations in 2010 for using GPUs in HLT farm for Inner Detector
  - significant amount of offline/HLT code ported to GPUs or developed from scratch



HLT GPU demonstrator project

- The modular “client-server” architecture
- Synchronous operations: client sends a request and waits for reply
- Automated load balancing with many GPUs using the data context queue:

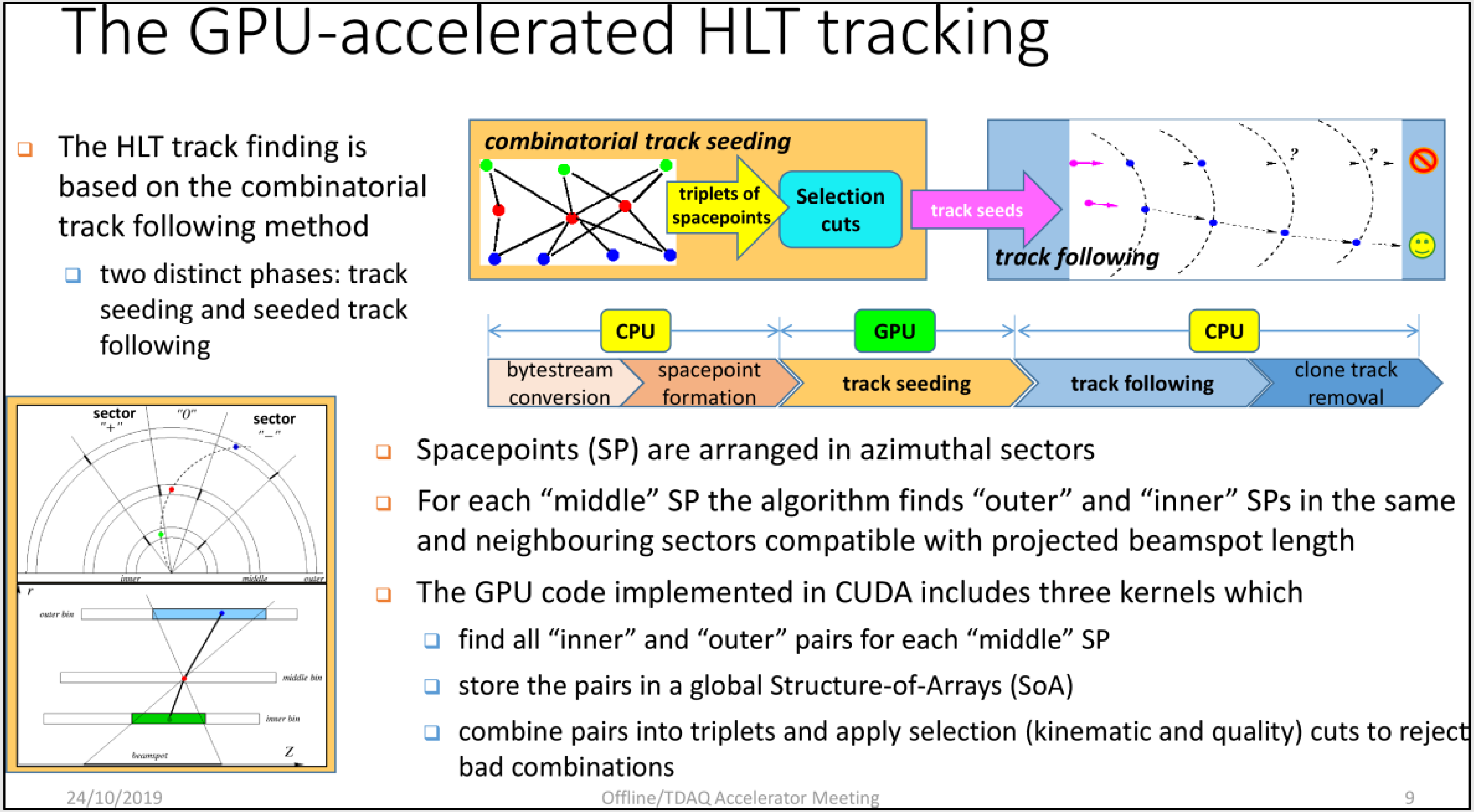
Modules are shared libraries which implement specific algorithms and can support various accelerator hardware

Data context encapsulates all data structures needed for work execution allocated on a GPU

24/10/2019 Offline/TDAQ Accelerator Meeting 3

- Client-server model (APE) made less important by advances in CUDA such as concurrent streams, and nvidia-cuda-mps-control server to share a GPU between several processes

from Dmitry Emeliyanov



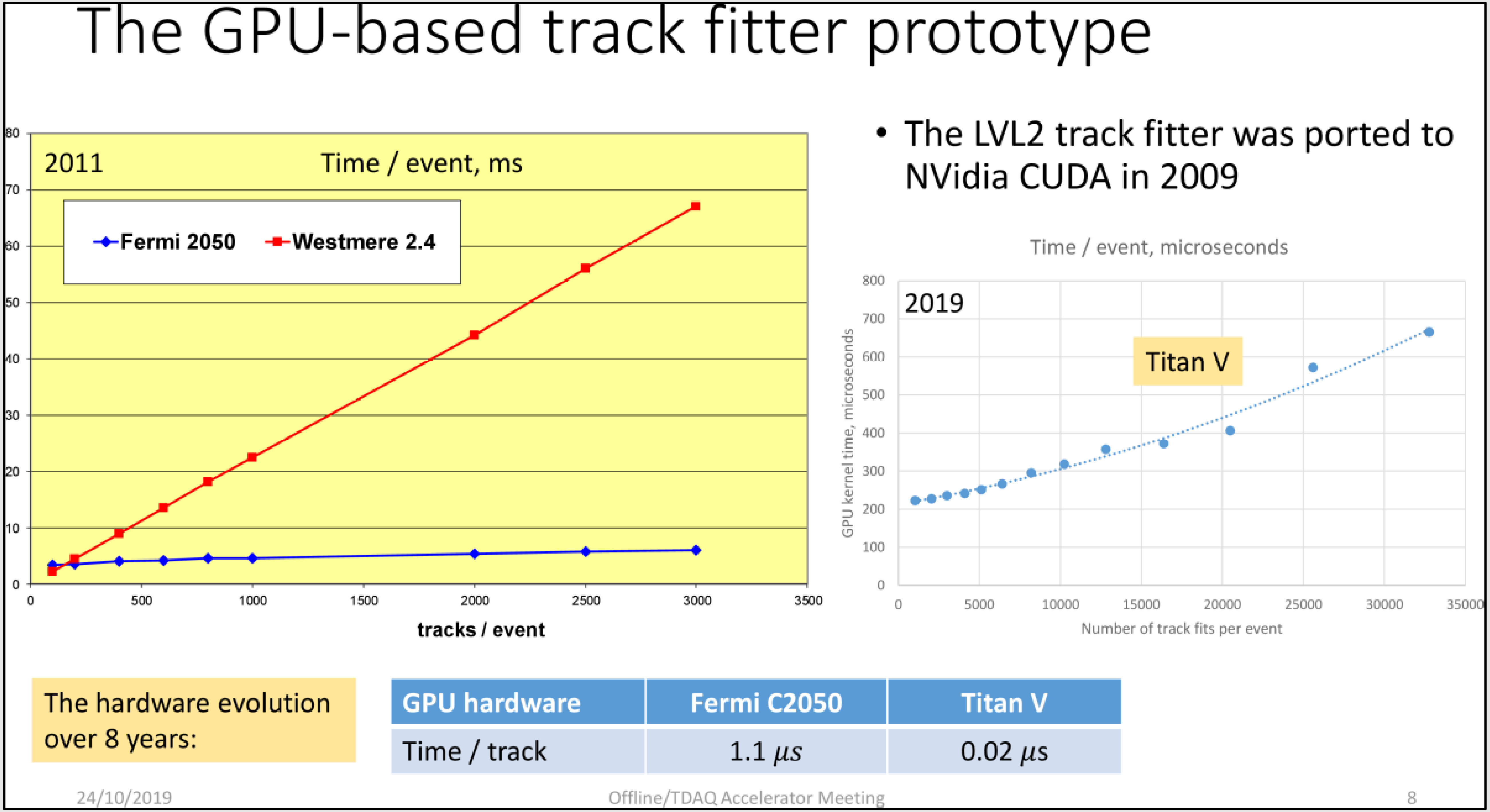
The GPU-accelerated HLT tracking

- The HLT track finding is based on the combinatorial track following method
  - two distinct phases: track seeding and seeded track following

combinatorial track seeding
triplets of spacepoints
Selection cuts
track seeds
track following
CPU
GPU
CPU
bytestream conversion
spacepoint formation
track seeding
track following
clone track removal
sector "+"
"0"
sector "–"
inner
middle
outer
r
outer bin
middle bin
inner bin
beamspot
Z

- Spacepoints (SP) are arranged in azimuthal sectors
- For each "middle" SP the algorithm finds "outer" and "inner" SPs in the same and neighbouring sectors compatible with projected beamspot length
- The GPU code implemented in CUDA includes three kernels which
  - find all "inner" and "outer" pairs for each "middle" SP
  - store the pairs in a global Structure-of-Arrays (SoA)
  - combine pairs into triplets and apply selection (kinematic and quality) cuts to reject bad combinations

Hardware Evolution



The GPU-based track fitter prototype

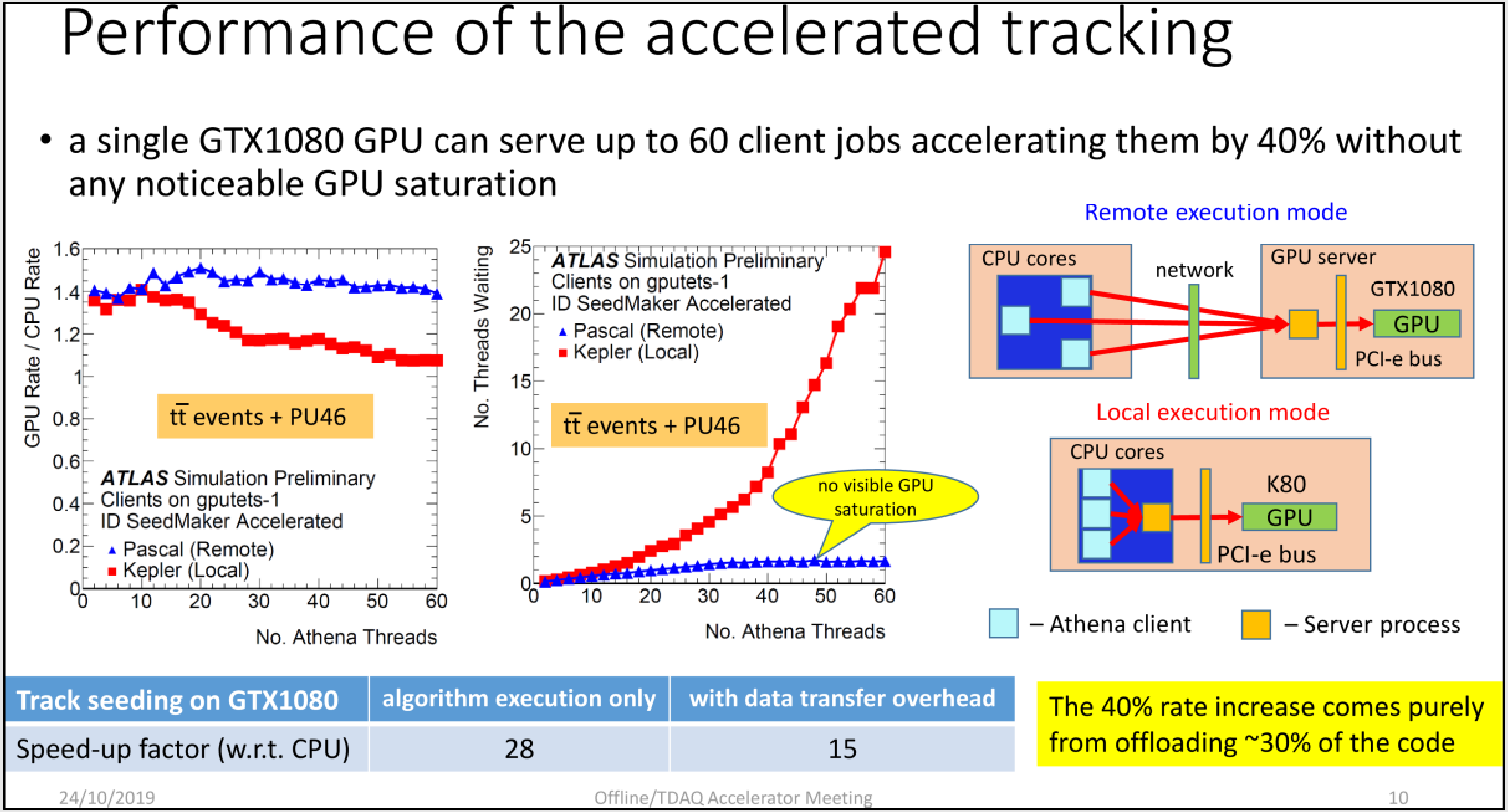
- The LVL2 track fitter was ported to NVidia CUDA in 2009

Time / event, microseconds
2019
Titan V
GPU kernel time, microseconds
0
100
200
300
400
500
600
700
800
0
5000
10000
15000
20000
25000
30000
35000
Number of track fits per event

The hardware evolution over 8 years:

GPU hardware	Fermi C2050	Titan V
Time / track	1.1 μs	0.02 μs

24/10/2019 Offline/TDAQ Accelerator Meeting 8



Performance of the accelerated tracking

- a single GTX1080 GPU can serve up to 60 client jobs accelerating them by 40% without any noticeable GPU saturation

Track seeding on GTX1080	algorithm execution only	with data transfer overhead
Speed-up factor (w.r.t. CPU)	28	15

The 40% rate increase comes purely from offloading ~30% of the code

Why Isn’t The HLT Using GPUs Now?

- Ultimately, the HLT decided not to use GPUs for Run 3

- Intermediate data required too much memory
  - 2MB input data → 50MB hits/doublets → 0.1MB output tracks
  - not enough fast, on chip memory (shared mem is very slow)
- Reject-accept nature of pattern recognition algorithms doesn’t map well onto GPUs
  - multiple levels of reduction (x2400) from cuts in eg, seed making don’t leave much work for GPU
  - memory latency can’t be hidden, performance bound to memory bandwidth
- Very large computational overheads from data conversion
  - need to convert C++ style objects to SofA that can be consumed by GPU
- Cost
  - what to do with big GPU farm when not taking data?

- HLT is currently re-evaluating GPU usage, looking at end-to-end solutions (hits in, tracks out), and modifications to EDM

Current GPU Demonstrator Projects

- Several “demonstrator” efforts are ongoing.
  - small, self contained
  - easy to run in different environments

- Being used as platforms to evaluate various GPU techonologies and issues
  - Languages
  - Ease of coding / conversion
  - GPU hardware comparison
  - Data structures
  - Portability to different platforms

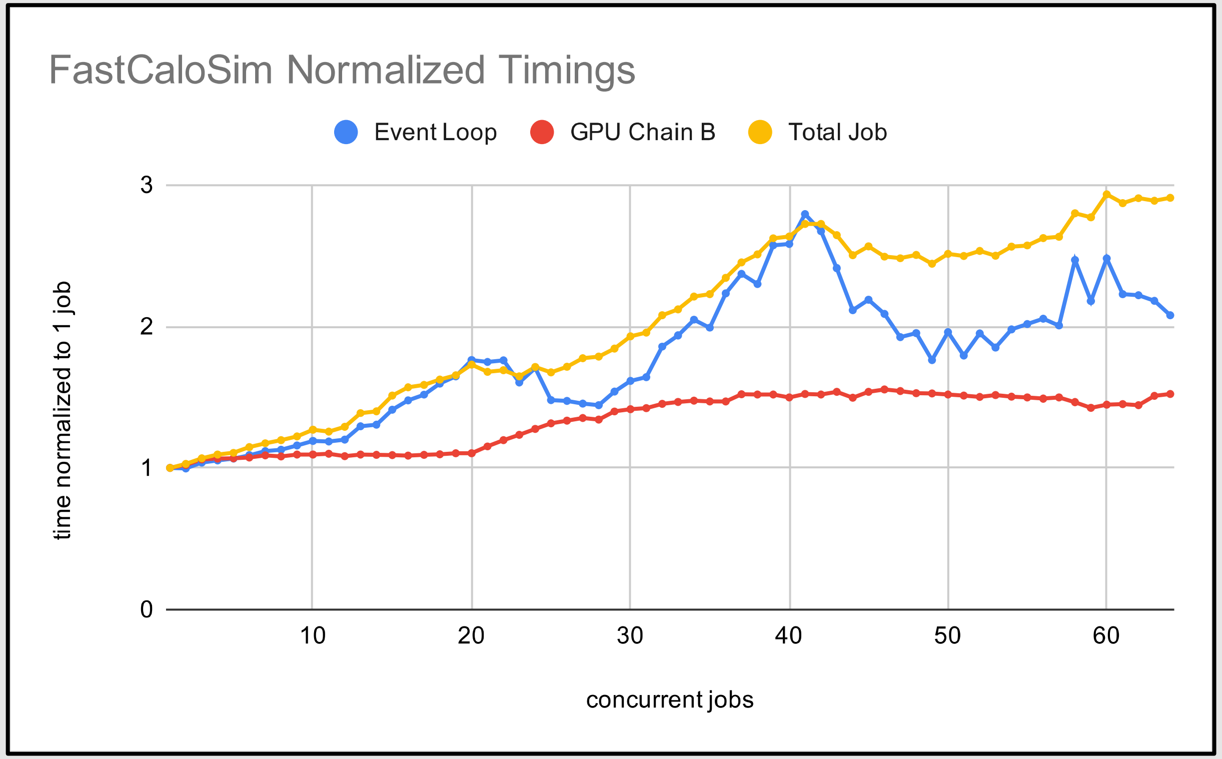
FastCaloSim

- Meifeng Lin @BNL

- Parametrized calorimeter simulation
- Aprox 6 months old
- Kernels written in CUDA
  - port to SyCL now underway

- Significant speedup over CPU (8x speedup for full event loop for simple hit simulation)

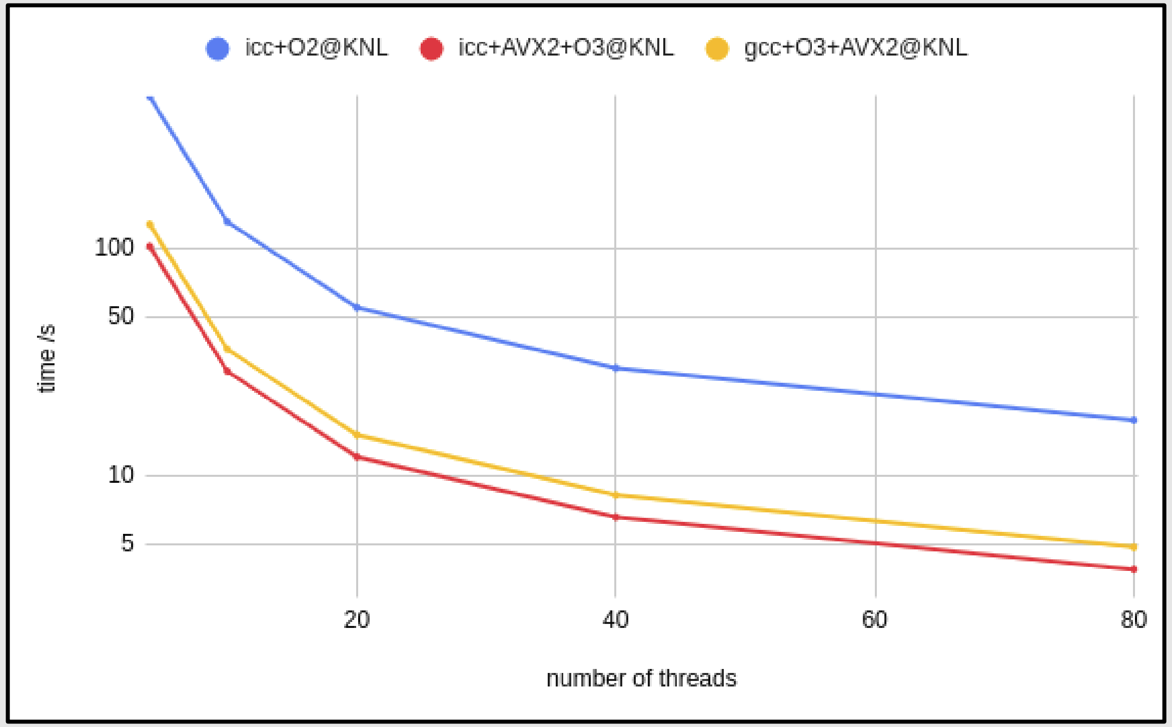
- Hampered by small work sizes - GPU is not well utilized
  - many concurrent CPU jobs can utilize the same GPU



ACTS

- “A Common Tracking Software”
- Heather Gray, Xiaocong Ai @LBL

- OpenMP hackathon to parallelize seed finder (originally targeting KNL on Cori)
- Extended to use OpenMP offloading directives to target GPU



Event Generation

- Very old (2012) Madgraph code that uses GPUs
  - parallelizing HEGET, gVEGAS, gBASES
  - GPU programs were not fully integrated
  - no CLI integration, event generation

- Current effort (Walter Hopkins @ ANL)
  - updated GPU code (CUDA4 → CUDA9)
  - updated gVEGAS
    - gVEGAS could also be used by Sherpa and other generators

- Next steps
  - update gSPRING and gBASES
  - benchmark performance

Doublet Seeding with Numpy+Numba (+GPU)

- Paolo Calafiura + Sean Conlon @LBL

- Project to parallelize track seeding with Numba
- Using fixed size arrays, parallelize inner loop
- Significant improvements in performance:
  - original for loop: 153s
  - 2-D Numpy array method: >200s
  - Numpy+Numba w/ parallel inner loop: 6s

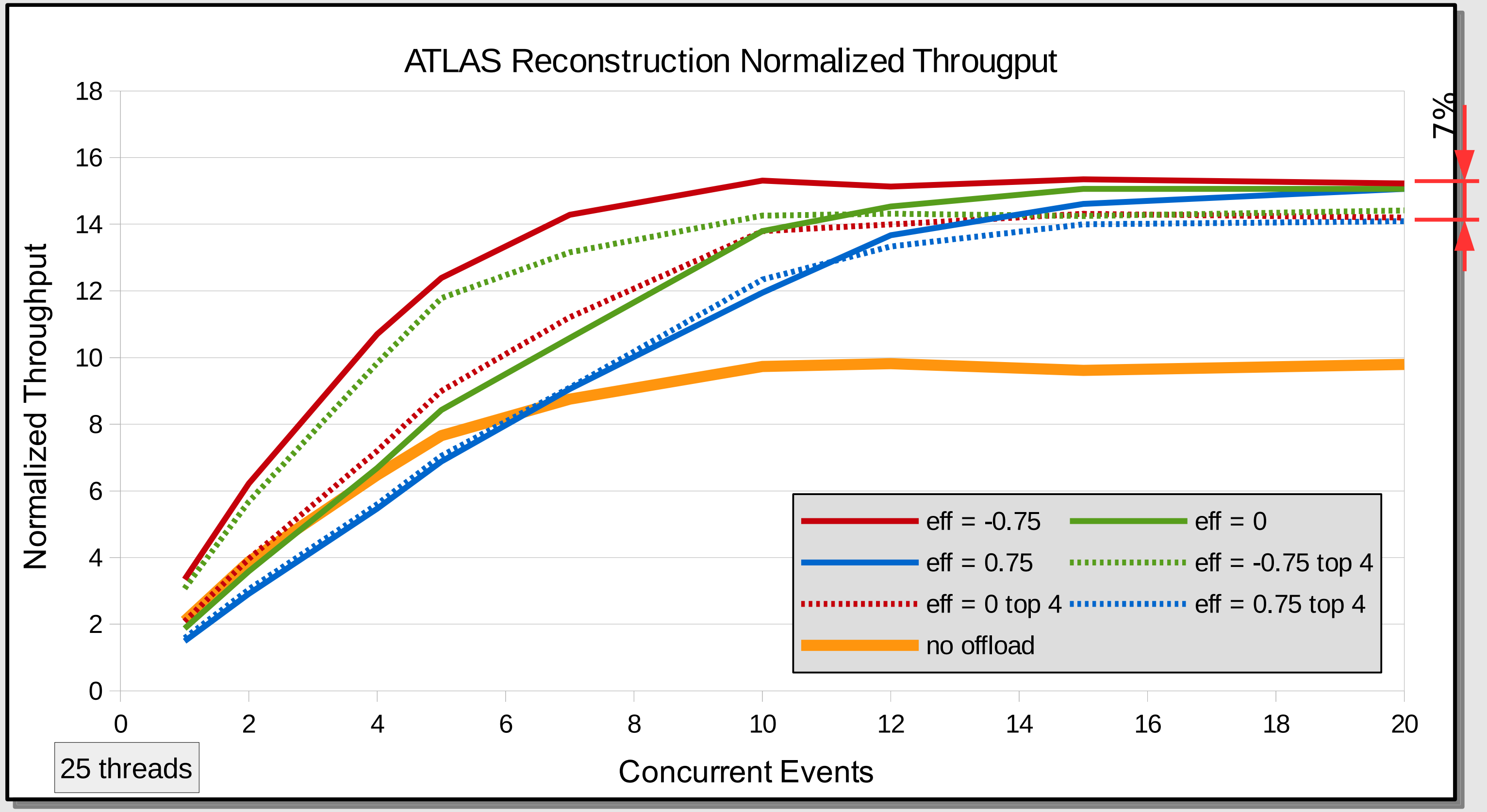
- Next step is to use CUDA backend to Numba
  - projected runtime: ms??

  - still some investigation needed for best data structures to use
    - CuDF or Numba CUDA + CuPy

- Select an inner hit
- Look at all the layers in the current and adjacent phi slices
- Apply cuts to the layers and phi slices
- Look at all the hits in the layers and phi slices that remain
- Apply cuts to this set of outer hits until the best candidates remain
- Finally, add these doublets to the set of input doublets

Framework Integration

- If we had Algorithms in AthenaMT workflows (reconstruction, simulation) that could be offloaded, how would we do it?
- AthenaMT AvalancheScheduler maintains a separate queue of Algorithms that it considers “IO Bound”
  - originally intended to be used for Algorithms that read/write data to disk
  - synchronous accelerator offloading behaves (in theory) in the same way
    - hardware thread is freed to do other work while GPU processes data
    - latency is hidden by scheduling more work (other Algorithms or concurrent events)
- Simulated reconstruction (q431)
  - oversubscribe with many more software threads than hardware
  - don’t need to offload many Algs to see significant gains
- Synchronous offloading has issues
  - may be addressed by NVidia soon



Asynchronous Offloading

- Asychronous offloading avoids the shortcomings of synchronous offloading
  - developed by Attila Krasznahorkay
  - fewer inefficiencies than synchronous offloading (for CUDA)
  - not possible in all languages
- Requires modifications to Gaudi / Scheduler to split `Algorithm::execute` into two steps
  - 1st part executed on GPU
  - 2nd part on CPU after GPU is done
    - triggered by callback, allows scheduler to continue dataflow processing
- Provides some transparent primitives to allow better GPU interaction with the AuxStore

Portability: The Challenge

- We see all three flavours of ice cream in the next the DOE HPCs
  - **A21**: Intel, **Perlmutter**: NVidia , **Frontier**: AMD
- Each accelerator vendor has its own flavour of programming tools to target their GPU
  - NVidia: CUDA
  - Intel: SYCL / OneAPI / dpcpp
  - AMD: hip
- Our software needs to run for the next 10+ years
  - we cannot afford to re-write for each hardware platform
- And what comes out in 5 years?
  - we may see more “exotic” architectures (TPU, FPGA, ASIC)
- What happens if a vendor significantly modifies their programming environment?
  - eg AMD recently dropped SPIR support for hip/ROCm

Size of Work

- Significant impedance mismatch:
  - V100 has 160,000 threads. AMD RX Vega / 5700 have similar. Intel Gen12 ???
  - Our loops are much less wide than that (10k or so)
    - may need to gang data between events to increase GPU workload - significant refactor
- Likely that we will need to be able to schedule and execute **concurrent kernels** on GPU
  - not well supported by portability layers
    - some explicit support (eg CUDA streams),
      - » even existing CUDA implementations have major performance drawbacks
      - » hidden synchronization points with some memory access
    - this may (is promised to) change in the coming year
  - we are often memory limited on GPUs
    - executing concurrent kernels exacerbates this problem

Working Towards A Solution

- We need to find a portable solution that works for all accelerator platforms
  - portability is more important than performance

- Projects worth exploring
  - Kokkos
  - Raja
  - SYCL / OpenCL
  - OpenMP / OpenACC
  - hip
  - Alpaka / cupla

- Machine learning tools
  - pytorch, tensorflow, etc
  - requires major paradigm shift to use pervasively
  - does map much better onto accelerators
  - hardware back-ends already there

- While some of these look good on paper, it is very important to understand how they map onto our workflows and framework
  - something which works well for the online may not map onto the offline

- We need to encourage ($$$) vendors to provide portability solutions
- We may need to develop these solutions ourselves if the vendors can’t deliver

General Issues

- Small problem size
  - inability to use all available threads on GPU
  - → may need to accumulate data over many events
- Data conversion
  - takes longer to convert data to form useable by GPU than to perform the calculation on GPU
  - → rewrite EDM
- Amdahl's law
  - in many workflows, parts that can be offloaded only account for small fraction of total time
  - → find non-traditional ways of doing calculations that map better onto GPUs
- Verification
  - code paths on GPU and CPU will never be the same
  - → verification of physics results will be required

Conclusions

- Lots of different ongoing efforts with different scopes

- Using GPUs *effectively* in HEP (especially for ATLAS) is non-trivial
  - ineffective usage can still lead to significant gains in throughput
  - our workloads tend to be too small. May need to gang data between events to “fill up” GPU, or implement concurrent kernel execution

- Portability is a big problem
  - major changes by hardware and software vendors in the next ~18 months
  - we should forge ahead with all our current studies while we evaluate different portability solutions

- We should not be afraid of paradigm shifts when looking for ways to use GPUs *fin*