R&D Line:
Accelerators and Parallelization

Georgiana Mania (DESY)

Computing Architectures in HEP

Main computational resource has been the CPU

High performance computing facilities now offer

- clusters of hundreds or even thousands of CPUs
- sockets and NUMA nodes with different inter-connectivity to answer different demands (high computational power vs high memory throughput)
- heterogenous resources – CPUs and GPUs
- low energy consumption resources – FPGAs
- containers and frameworks which make use of the hardware topology to ensure an efficient level of resource allocation and resource usage

Different level of parallelization are now achievable

Parallelization

Hardware parallelism

- instruction level (CPUs have now built-in instruction pipelines to speed up the execution)
- thread level (within the same CPU)
- process level (within the same node or different nodes of the cluster)
- CPU-GPU (by offloading a part of the computation to the accelerator)

Software parallelism

- data parallelism (same code is executed on different data)
    - vectorization (using wide registers)
    - multi-threads (eg OpenMP)
- task parallelism (each thread/process executes a different part of the code on the same/different data)

ACTS's Track Reconstruction

Step 1: Seed finding

- Combinatorial search in space

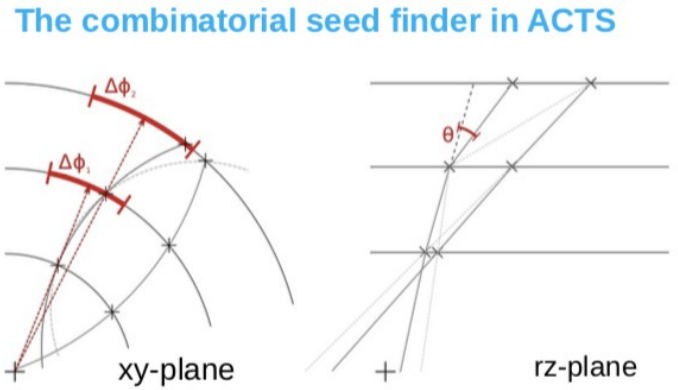


Image source: X. Ai

Step 2: Track following

- Combinatorial Kalman Filter for simultaneous track finding and fitting
  - Integrate the equation of motion of charged particles in a magnetic field
  - Propagate the track parameters and covariance matrix through the detector's geometry

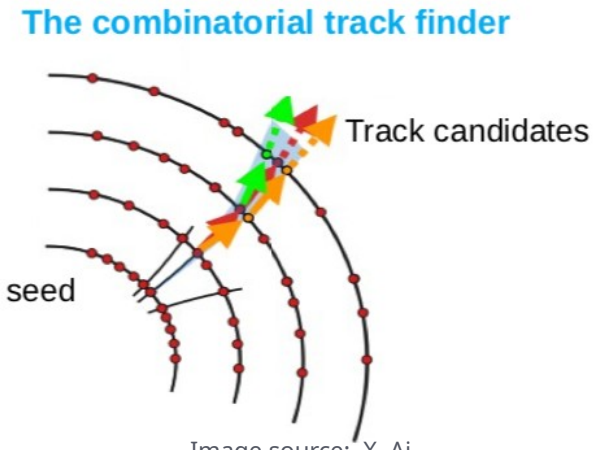


Image source: X. Ai

→ Track reconstruction requires compute-intensive algorithms!

ACTS offer the optimal environment to explore ALL the parallelization strategies into tackling the complexity!

Seed Finding

CPU time from the total reconstruction time

- 20% in Run 2
- estimated **50%** for <μ>=200

Computational characteristics

- combinatorial problem → multiplicity of the solutions
- problem space can be decomposed in independent sub-spaces which can be explored in parallel

} perfect candidate for massive parallelism

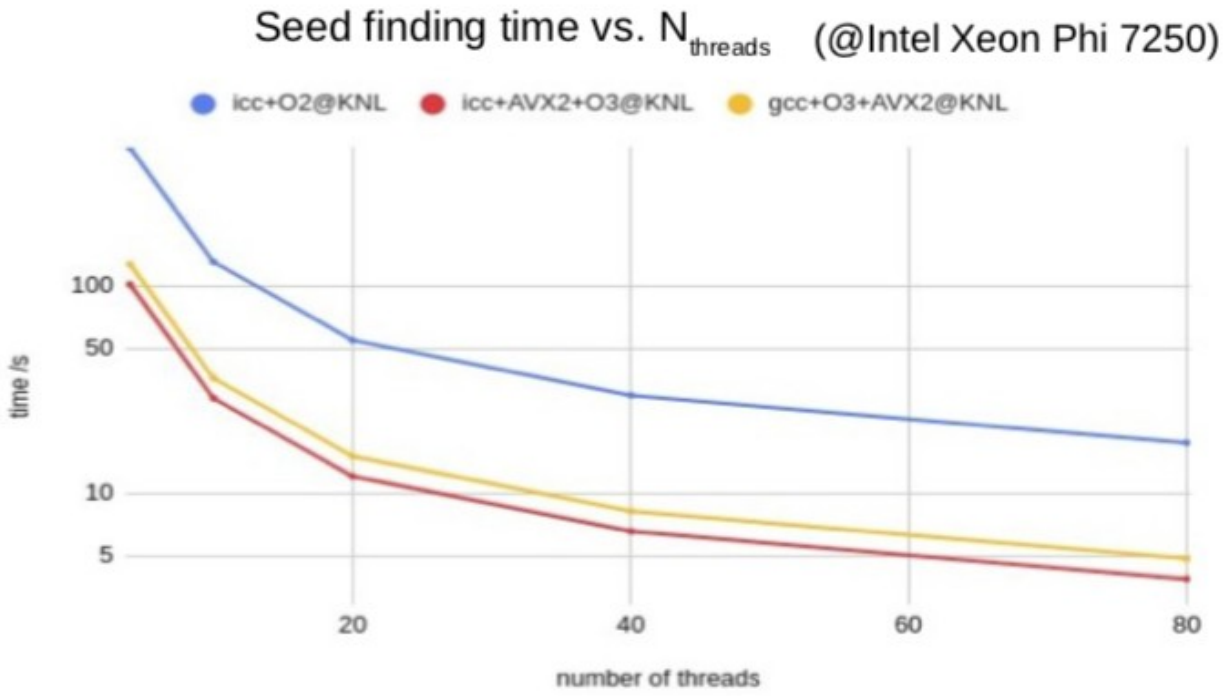
Recent ACTS experiments explored

- CPU thread-level parallelism
- GPU accelerated support using SYCL and CUDA

Seed Finding – CPU Parallelism

Multi-threaded implementation

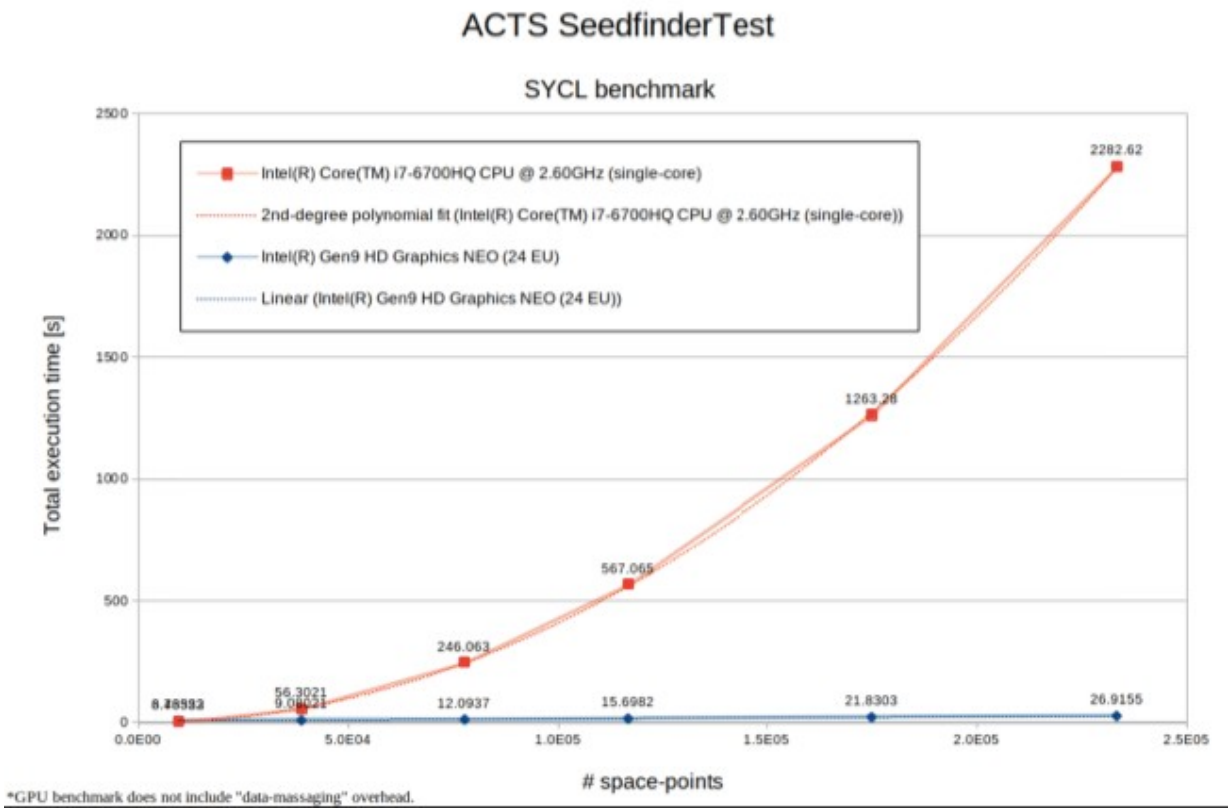
- explored by X. Ai, X. Ju, C. Leggett et. al
- uses **OpenMP** for thread-level parallelism
- shows a 20x speedup



Seed Finding – GPU Accelerated

SYCL implementation

- explored by V. Pascuzzi, C. Leggett et. al
- CPU and GPU results matched 100%
- great scalability was achieved when working with space points as arrays

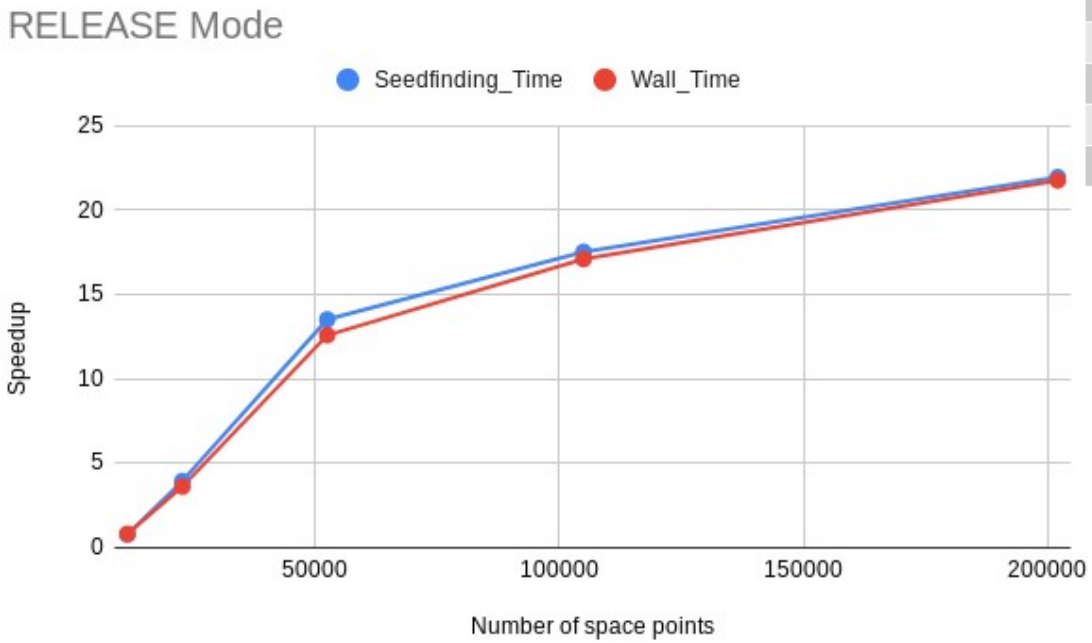


*GPU benchmark does not include "data-massaging" overhead.

Seed Finding – GPU Accelerated

CUDA implementation

- explored by B. Yeo, C. Leggett et. al
- proves great scalability and efficiency
- CPU and GPU results matched 100% for #space points < 200k when fast math mode is turned off in CUDA compiler
- in progress of promoting from “experimental” to “production ready”



# space points	# of seeds (CUDA)	# of seeds (CPU)	Seed matching ratio
10k	2690	2690	100%
20k	5512	5512	100%
50k	12805	12805	100%
100k	25572	25572	99.99%
200k	49302	49302	99.97%

Note:

The number of space points for HL-LHC is between 50k and 100k.

Track Finding & Fitting

ACTS examples use task-level parallelism through Intel TBB library support

```
258    // execute the parallel event loop
259    tbb::task_scheduler_init init(m_cfg.numThreads);
260    tbb::parallel_for(
261        tbb::blocked_range<size_t>(eventsRange.first, eventsRange.second),
262        [&](const tbb::blocked_range<size_t>& r) {
263          std::vector<Duration> localClocksAlgorithms(names.size(),
264                                                      Duration::zero());
265
266          for (size_t event = r.begin(); event != r.end(); ++event) {
```

Code snippet from Sequencer.cpp

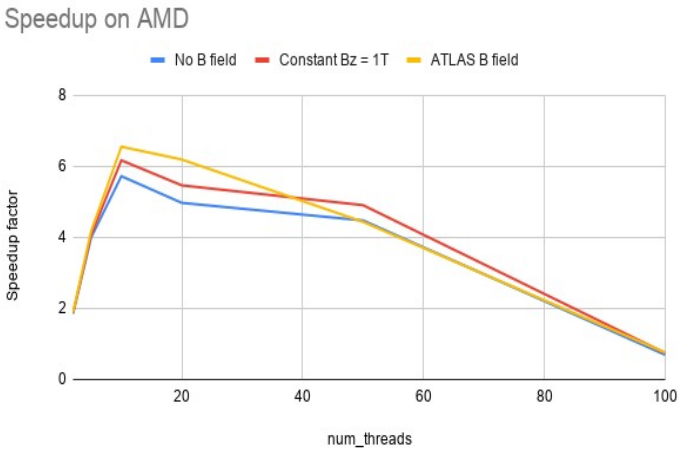
Recent ACTS track fitting examples explored

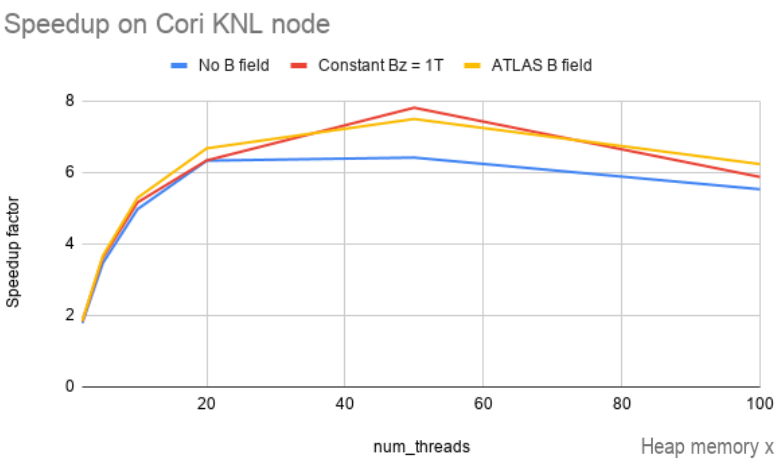
- Imbricated task-parallelism
- GPU offloading of the numerical integration

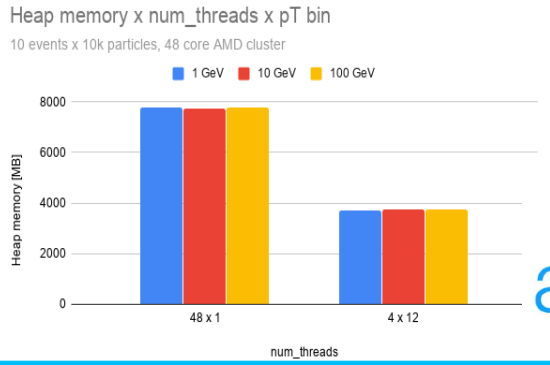
Track Fitting – CPU Parallelism

Multi-thread implementation

- explored by G. Mania, N. Styles, X. Ai et. al
- comes on top of the existing multi-threaded solution for event handling
- uses **Intel TBB** for imbricated task level parallelism (for track handling)
- shows good speedup and reduced memory footprint





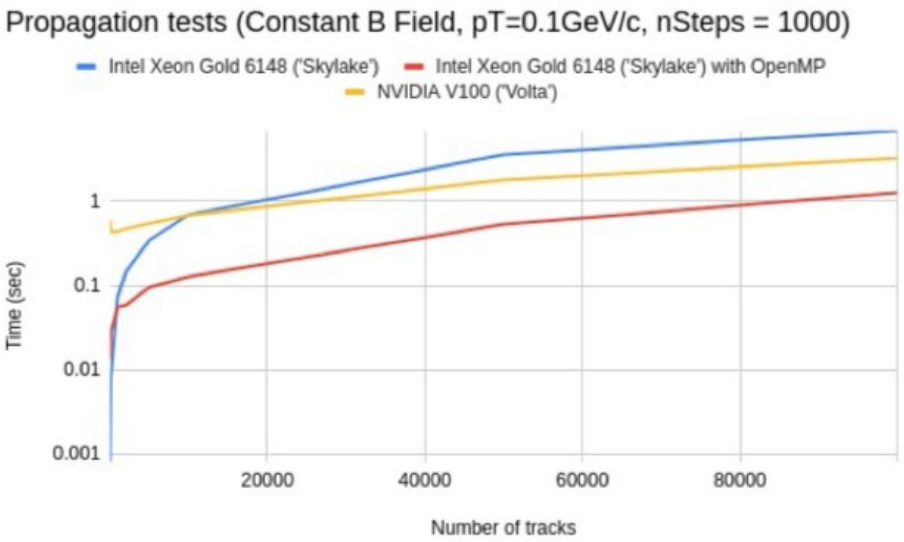


Track Fitting – GPU Accelerated

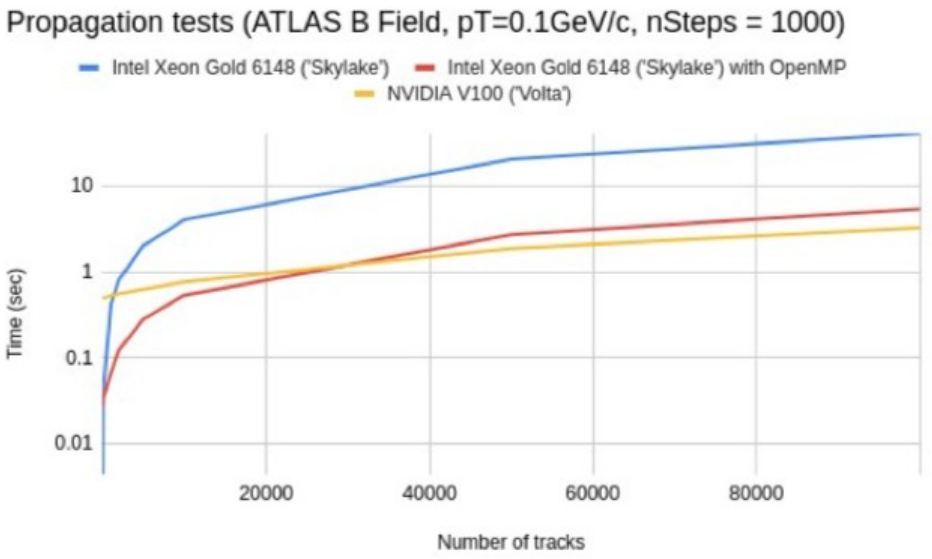
CUDA implementation of adaptive RKN4

- explored by X. Ai et. al
- shows comparable results to OpenMP CPU implementation

Constant B Field



ATLAS B Field



Ongoing efforts

Port the full functionality of the Kalman Filter on CUDA and measure performance gains

Explore parallel numerical algorithms with the goal to speed up the integration

Investigate how the hardware-awareness can be used to optimize the parallelization

Summary & Outlook

- The motivation for exploring parallel architectures lies in the need to efficiently make use of computational clusters
- Seed finding approaches based on OpenMP, SYCL and CUDA showed very promising results
- Track fitting implementation extended with a second level of parallelism showed potential great improvements in memory usage
- CUDA Stepper proves to be comparable with thread-parallel CPU code
- Ongoing research efforts are being done to improve ACTS's scalability

Thank you!

Backup

Clusters configurations:

- KNL node on NERSC Cori:

  1 socket Intel Xeon Phi 7250 @1.4GHz,68 cores x 4 hyper-threads

- GPU node on NERSC Cori:

  2 sockets x 20-core Intel Xeon Skylake @ 2.4 GHz
  8 NVIDIA Volta V100 GPUs x 5120 CUDA Cores, 16 GB memory

- AMD NUMA cluster @ UHH:

  4 sockets x 12 CPUs Opteron 6168 @1.9GHz → 48 cores