GPU-based tracking with

Xiaocong Ai for the Acts team

Xiaocong.ai@desy.de

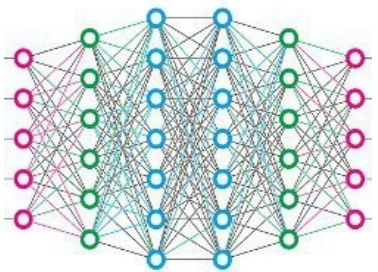
HSF WLCG workshop, Nov 24, 2020

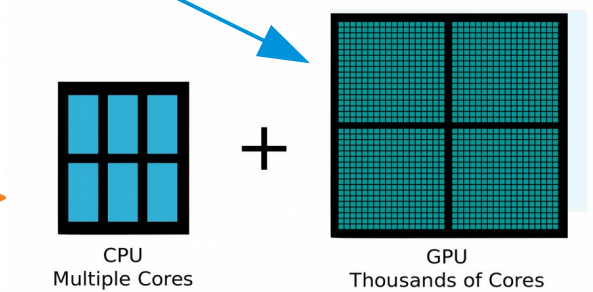
project

- A modern open-source HEP tracking toolkit
- A R&D platform to explore new techniques, parallelization and acceleration realized via plugins:

acts / Plugins /

- Autodiff — Covered in B. Huth's talk
- Cuda
- DD4hep
- Digitization
- Identification
- Json
- Legacy
- Onnx
- Sycl
- TGeo





About

Experiment-independent toolkit for (charged) particle track reconstruction in (high energy) physics experiments implemented in modern C++

acts.readthedocs.io

particle-track-reconstruction · reconstruction · physics-experiment · simulation

Readme

MPL-2.0 License

Releases 68

v3.0.0 Latest
15 hours ago

https://github.com/acts-project/acts

The charged track reconstruction

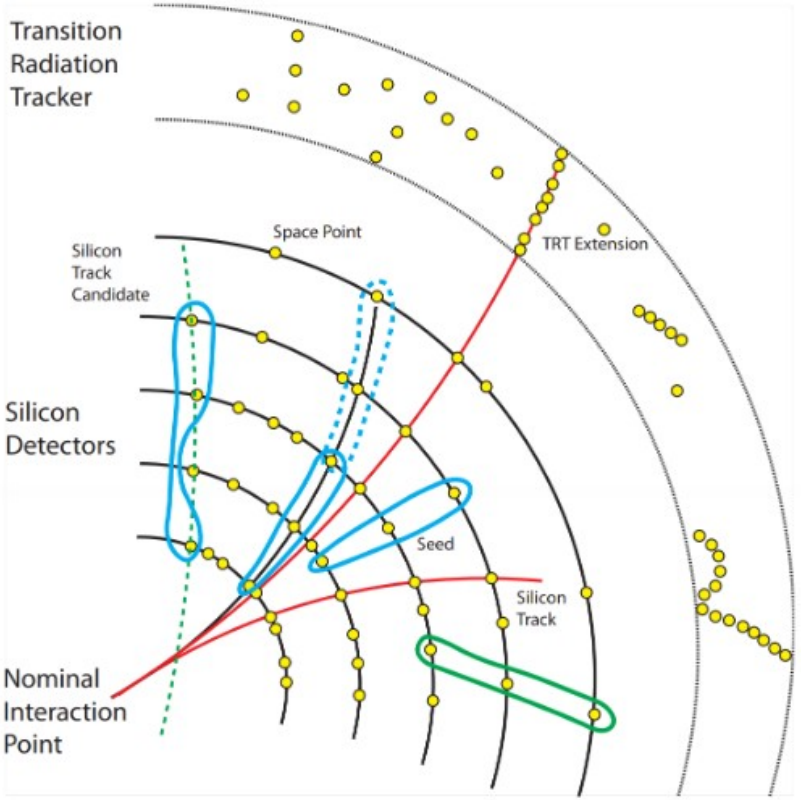
Input data preparation
(e.g. space point formation)

Track finding
(e.g. Seed finding + Combinatorial Kalman Filter)

Could be merged into track finding

Track fitting
(e.g. Kalman Filter)

Vertex reconstruction



The Acts seed finder

Flat data structure, fine-grained parallelism

Input 3D Space points

Groups building based on phi and z coordinates

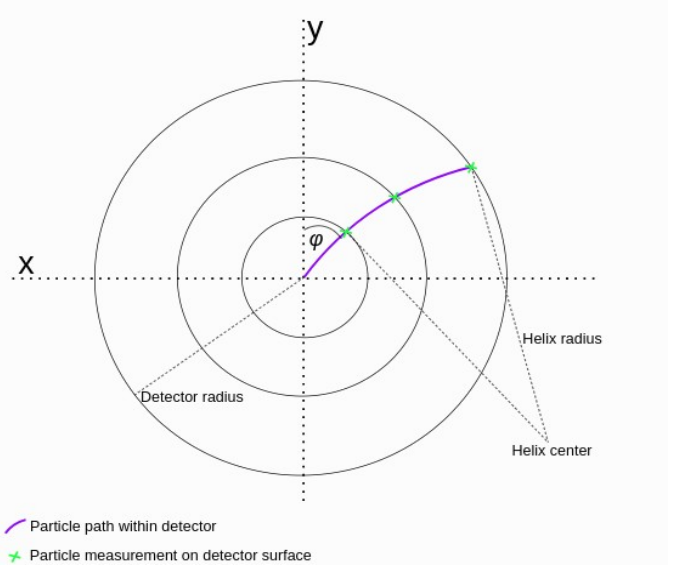
Duplet finding

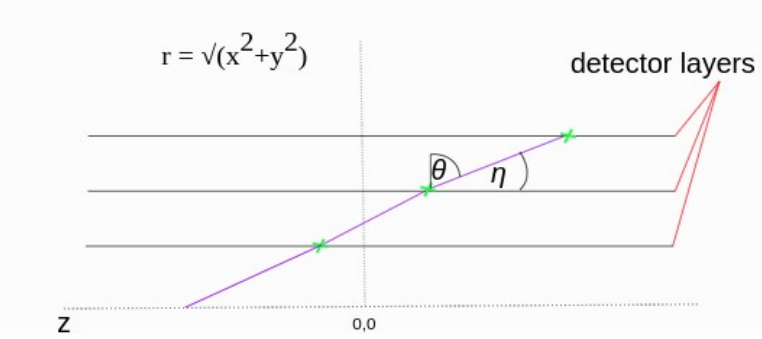
Triplet finding

Triplet filtering

Highly parallelizable

partially parallelizable





The GPU-based Acts seedfinder

- Implemented with both Cuda and Sycl/oneAPI

A. Krasznahorkay, A.Czirkos,
B. Yeo, C. Leggett et al.

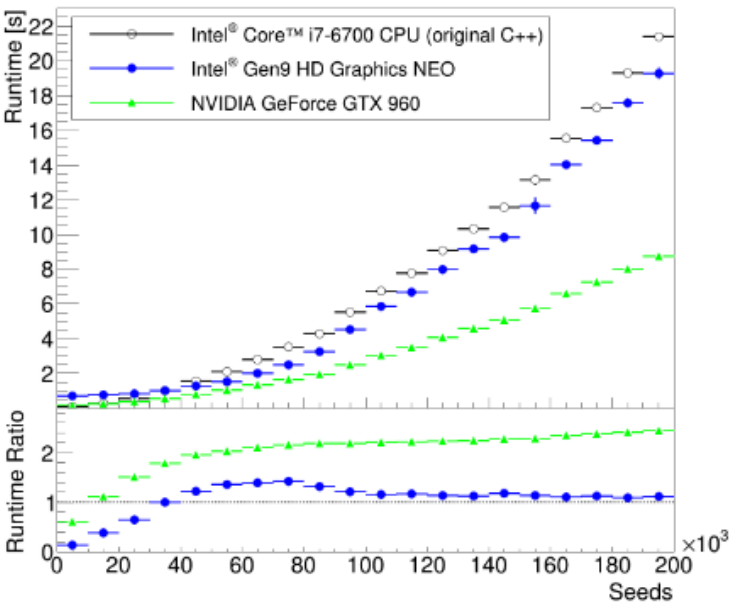
```
template <int SPType>
__global__ void findDublets(std::size_t nMiddleSPs,
                            const Details::SpacePoint* middleSPs,
                            std::size_t nOtherSPs,
                            const Details::SpacePoint* otherSPs,
                            float deltaRMin, float deltaRMax, float cotThetaMax,
                            float collisionRegionMin, float collisionRegionMax,
                            unsigned int* dubletCounts, std::size_t* dublets) {
  // Figure out which dublet the kernel operates on.
  const std::size_t middleIndex = blockIdx.x * blockDim.x + threadIdx.x;
  const std::size_t otherIndex = blockIdx.y * blockDim.y + threadIdx.y;

  // If we're outside of bounds, stop here.
  if ((middleIndex >= nMiddleSPs) || (otherIndex >= nOtherSPs)) {
    return;
  }

  // Calculate variables used in the compatibility check.
  const float deltaR = getDeltaR<SPType>(middleSPs[middleIndex].radius,
                                         otherSPs[otherIndex].radius);
  const float cotTheta = getCotTheta<SPType>(middleSPs[middleIndex].z,
                                             otherSPs[otherIndex].z, deltaR);
  const float zOrigin =
      middleSPs[middleIndex].z - middleSPs[middleIndex].radius * cotTheta;
```

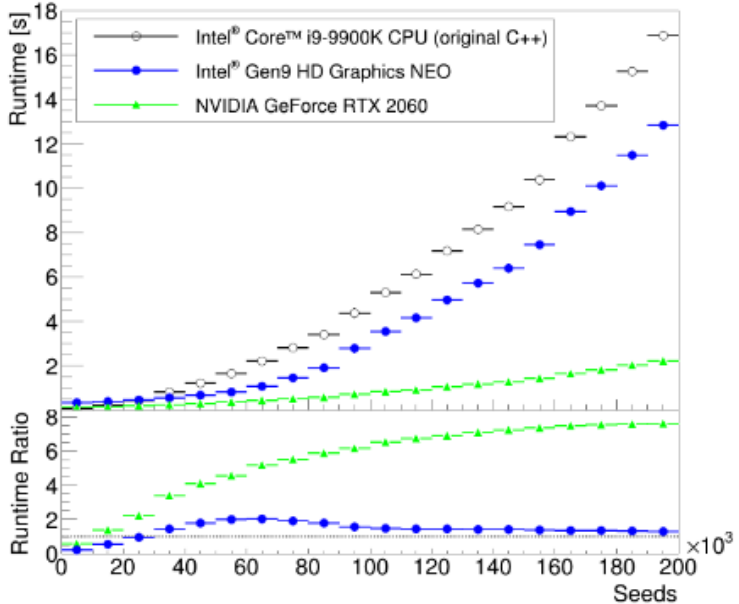
```
template <typename external_spacepoint_t>
class Seedfinder {
 public:
  Seedfinder(
      Acts::SeedfinderConfig<external_spacepoint_t> config,
      const Acts::Sycl::DeviceExperimentCuts& cuts,
      Acts::Sycl::QueueWrapper wrappedQueue = Acts::Sycl::QueueWrapper());

  ~Seedfinder() = default;
  Seedfinder() = delete;
  Seedfinder(const Seedfinder<external_spacepoint_t>&) = delete;
  Seedfinder<external_spacepoint_t>& operator=(
      const Seedfinder<external_spacepoint_t>&) = delete;

  /// Create all seeds from the space points in the three iterators.
  /// Can be used to parallelize the seed creation
  /// @param bottom group of space points to be used as innermost SP in a seed.
  /// @param middle group of space points to be used as middle SP in a seed.
  /// @param top group of space points to be used as outermost SP in a seed.
  /// Ranges must return pointers.
  /// Ranges must be separate objects for each parallel call.
  /// @return vector in which all found seeds for this group are stored.
  template <typename sp_range_t>
  std::vector<Seed<external_spacepoint_t>> createSeedsForGroup(
      sp_range_t bottomSPs, sp_range_t middleSPs, sp_range_t topSPs) const;
```

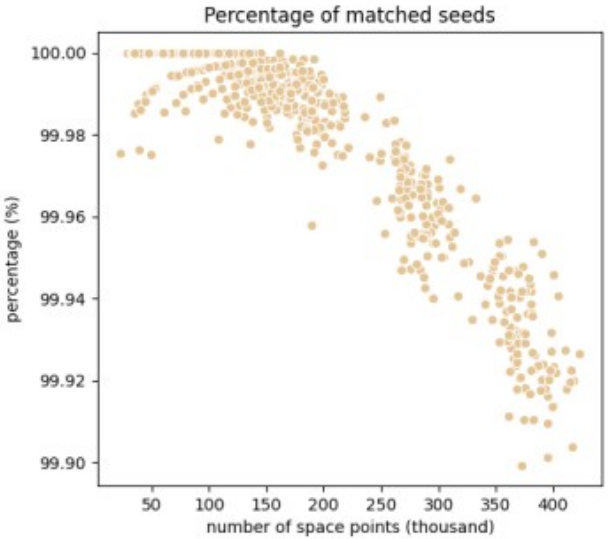
Acts seed finder performance on GPU

- Physics performance on GPU validated
- Speed gain is achieved using GPU, but very limited if the GPU is an integrated graphics

A single thread on CPU as reference test







CPU performance remains competitive if with multi-cores

Acts Kalman Filter

A track parameters updating process based on matrix operations

Track parameter is transported using Runge-Kutta-Nyström method integration with adaptive step size:

$$\mathbf{C}_k^{k-1} = \mathbf{F}_{k-1}\mathbf{C}_{k-1}\mathbf{F}_{k-1}^{\mathrm{T}} + \mathbf{Q}_{k-1}.$$

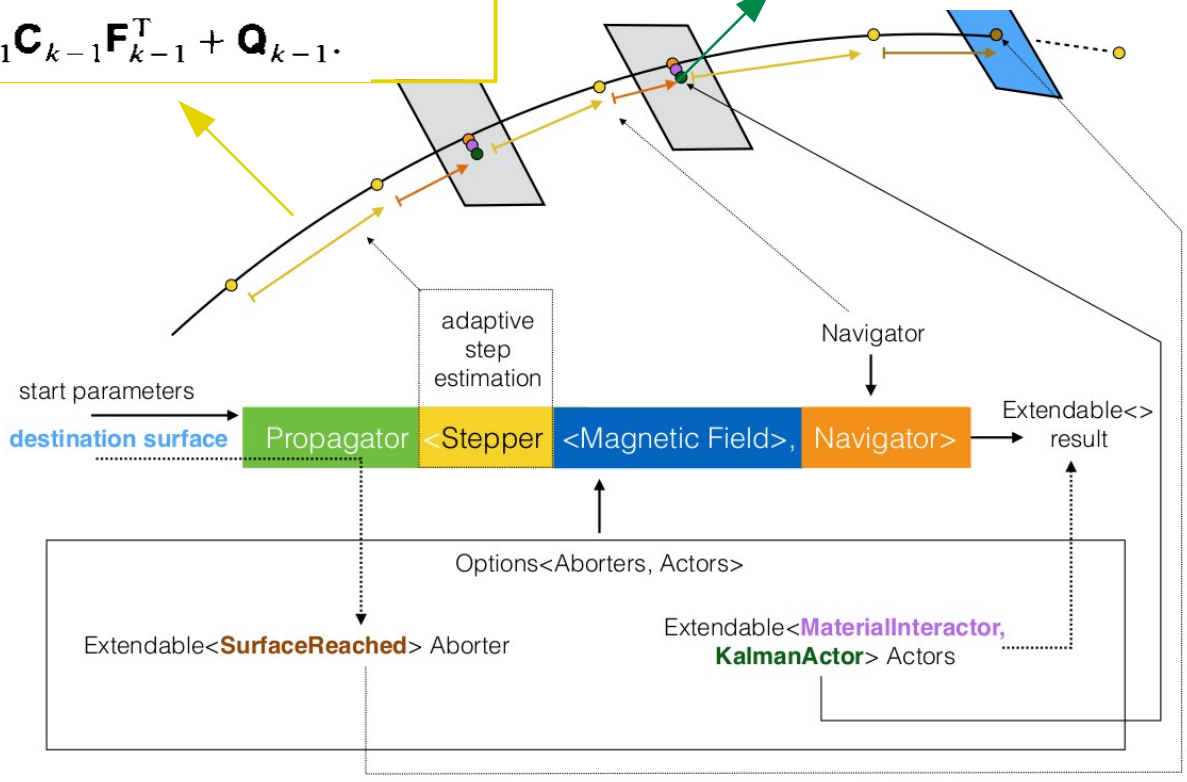
Track parameter is 'filtered' and 'smoothed' using measurements:

$$\mathbf{K}_k = \mathbf{C}_k^{k-1}\mathbf{H}_k^{\mathrm{T}}\left(\mathbf{V}_k + \mathbf{H}_k\mathbf{C}_k^{k-1}\mathbf{H}_k^{\mathrm{T}}\right)^{-1}$$

$$\mathbf{C}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{C}_k^{k-1}$$

$$\mathbf{A}_k = \mathbf{C}_k\mathbf{F}_k^{\mathrm{T}}\left(\mathbf{C}_{k+1}^{k}\right)^{-1}$$

$$\mathbf{C}_k^n = \mathbf{C}_k + \mathbf{A}_k\left(\mathbf{C}_{k+1}^n - \mathbf{C}_{k+1}^k\right)\mathbf{A}_k^{\mathrm{T}}$$



'Global' tracking data:
detector geometry
and magnetic field

Track fitting parallelization strategy

X. Ai., G. Mania, N. Styles, H. Gray, M. Kuhn et al.

```
__global__ void __launch_bounds__(256, 2) fitKernelThreadPerTrack(
    KalmanFitterType *kFitter, Acts::PixelSourceLink *sourcelinks,
    BoundState *startStates, Acts::LineSurface *targetSurfaces,
    FitOptionsType *fitOptions, TSType *fittedStates,
    Acts::BoundParameters<Acts::LineSurface> *fitPars, bool *fitStatus,
    const Acts::Surface *surfacePtrs, int nSurfaces, int nTracks, int offset) {
  // In case of 1D grid and 1D block, the threadId = blockDim.x*blockIdx.x +
  // threadIdx.x + offset
  // @note This might have problem if the number of threads is smaller than the
  // number of tracks!!!
  int threadId =
      blockDim.x * blockDim.y * (gridDim.x * blockIdx.y + blockIdx.x) +
      blockDim.x * threadIdx.y + threadIdx.x + offset;
```

One thread per track:

- heavy workload for one thread

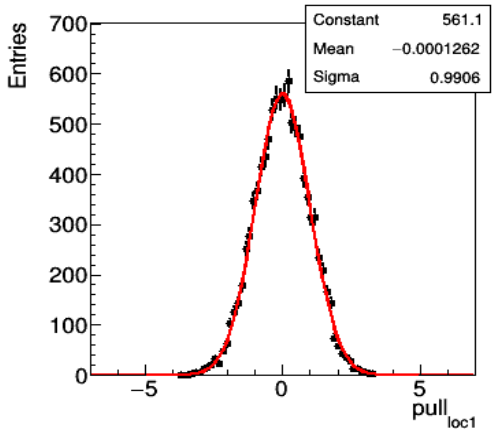
```
__global__ void __launch_bounds__(256, 2) fitKernelBlockPerTrack(
    KalmanFitterType *kFitter, Acts::PixelSourceLink *sourcelinks,
    BoundState *startStates, Acts::LineSurface *targetSurfaces,
    FitOptionsType *fitOptions, TSType *fittedStates,
    Acts::BoundParameters<Acts::LineSurface> *fitPars, bool *fitStatus,
    const Acts::Surface *surfacePtrs, int nSurfaces, int nTracks, int offset) {
  // @note This will have problem if the number of blocks is smaller than the
  // number of tracks!!!
  int blockId = gridDim.x * blockIdx.y + blockIdx.x + offset;
```

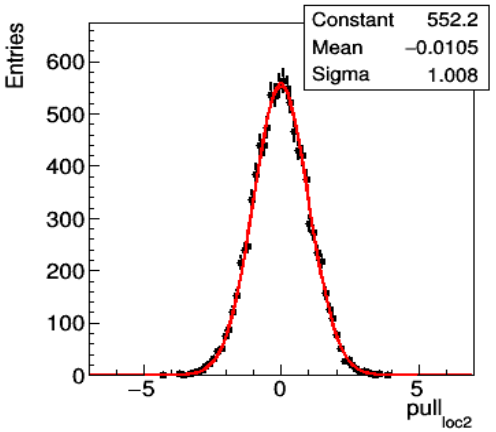
One block with 64 threads per track:

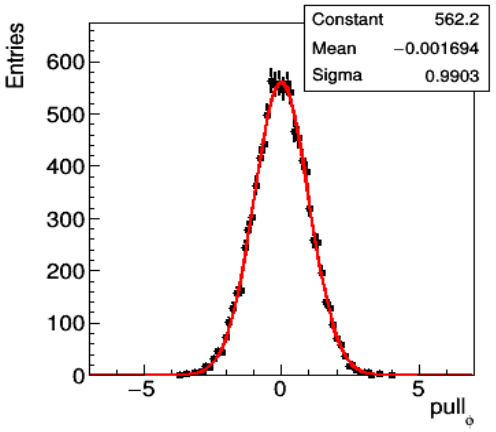
- Main thread only for trivial execution
- 64 threads simultaneously for large size matrix multiplication

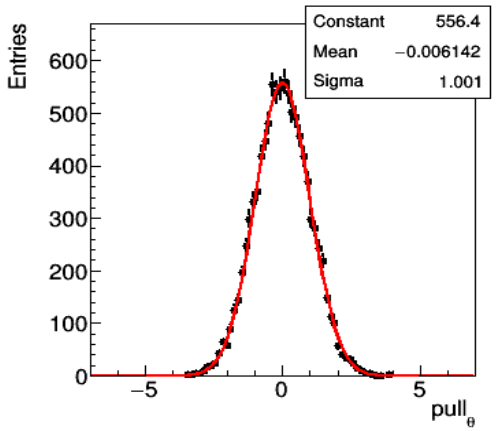
Fitting results validation

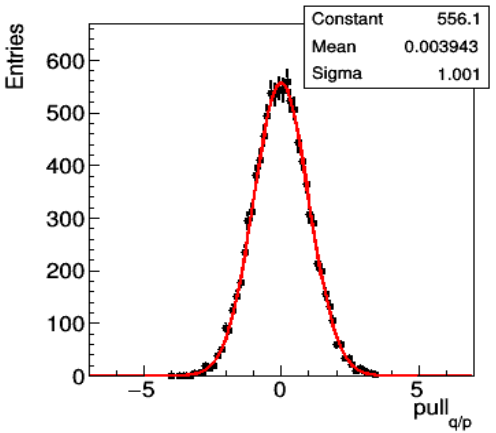
Perigee parameter pulls has healthy normal distributions

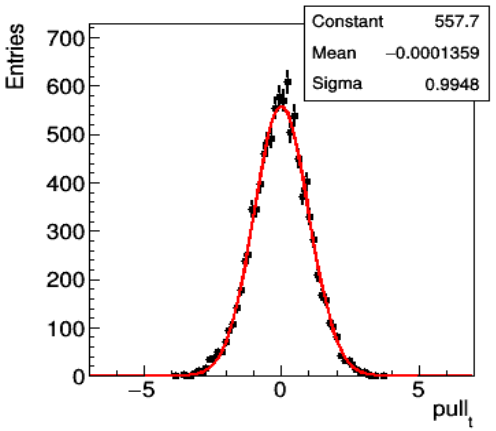








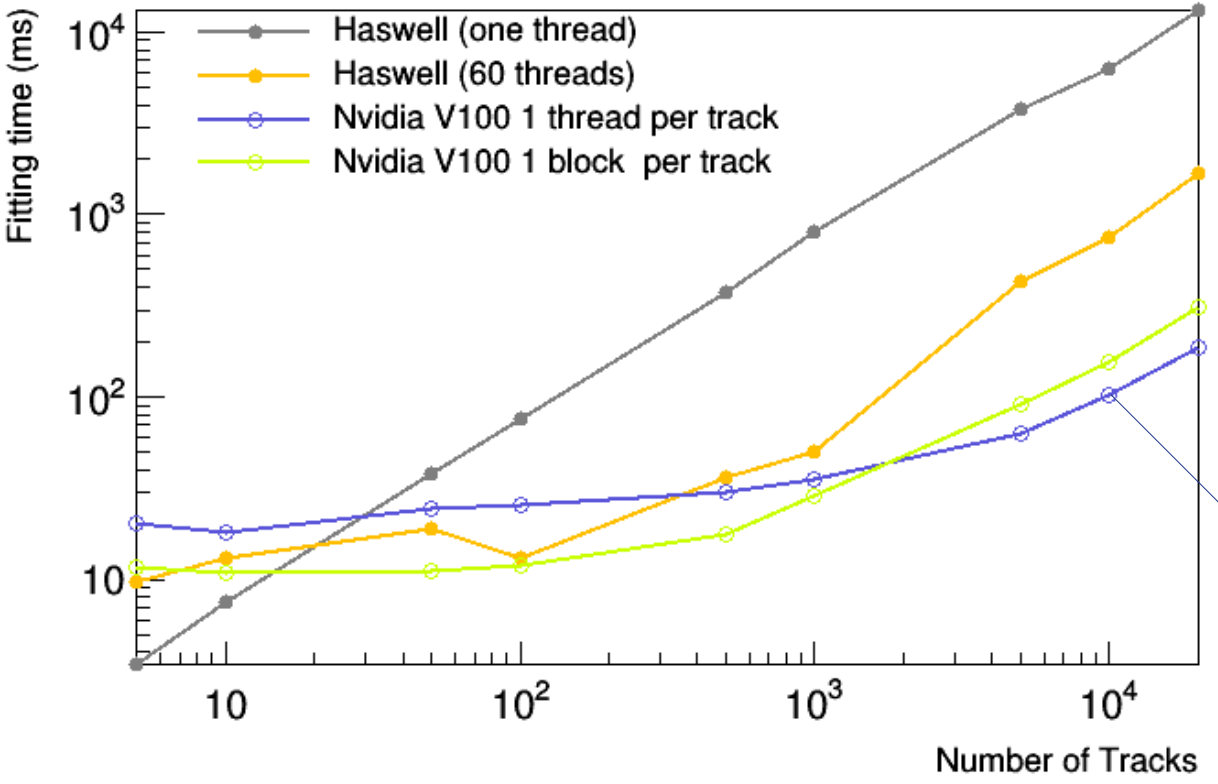




10k single muons with telescope-like geometry (10 plane surfaces) in constant B field

Fitting timing performance

- On-going optimizations of matrix operations on GPU
  - Had to implement a customized version of the Eigen matrix inverse() method



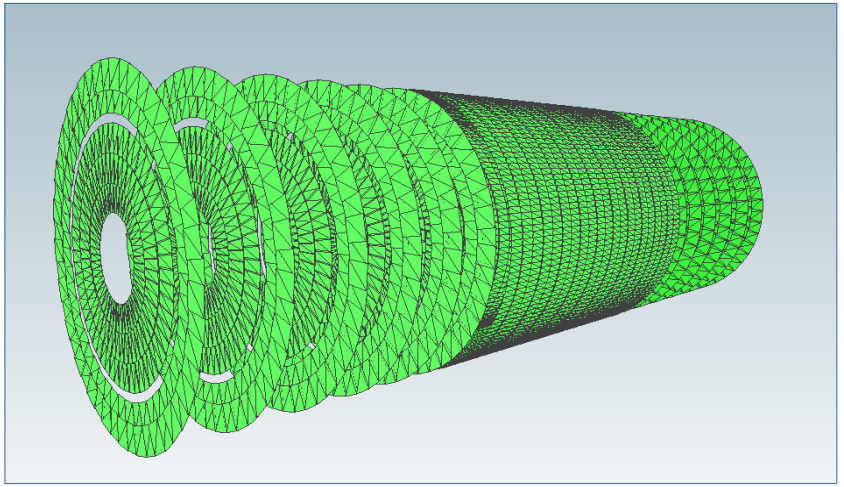
For better performance, use just one thread per track when number of tracks >2k

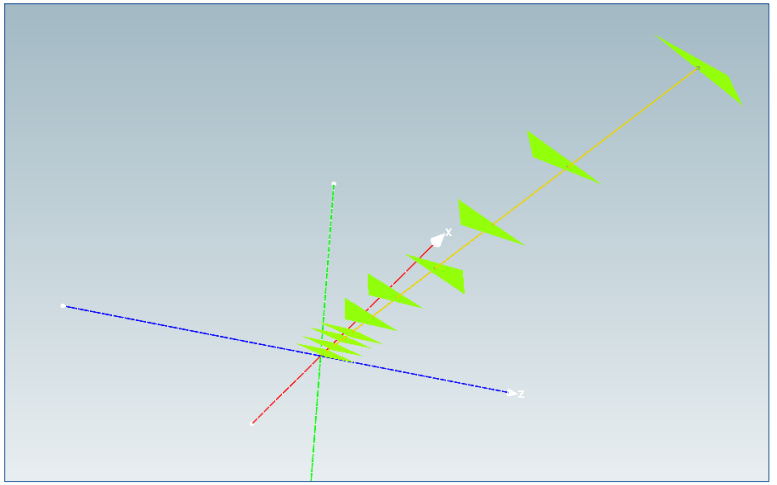
Kernel execution time: 67ms

GPU-based geometry navigation

X. Ai., A. Salzburger et al.

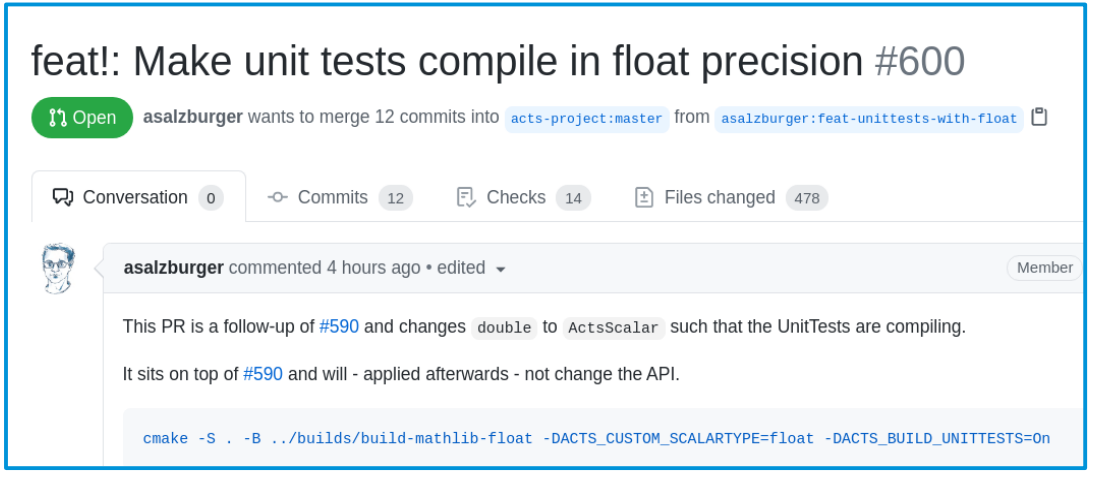
- GPU is not polymorphism-friendly
  - Common representation of geometry objects using flat data structure is desirable
- Ray-tracing inspired geometry navigation is implemented:
  - Surfaces are meshed using polyhedron, e.g. triangles
  - On-going investigation to minimize data transfer overhead





Summary

- Prototype of GPU-based tracking tools with Acts are available
  - Seed finder, Kalman Fitter, geometry navigator
- ‘One code for all’ is difficult (though GPU is now more generic-purposed)
  - Speed-up with GPU might be gained at the cost of code customizability, portability and performance loss on CPU
- But working towards the solutions
  - e.g. work almost done in Acts to use customized scalar precision (e.g float or double)
- More R&D is necessary to understand all the possibilities and limitations
  - Welcome to join the efforts if you are interested!

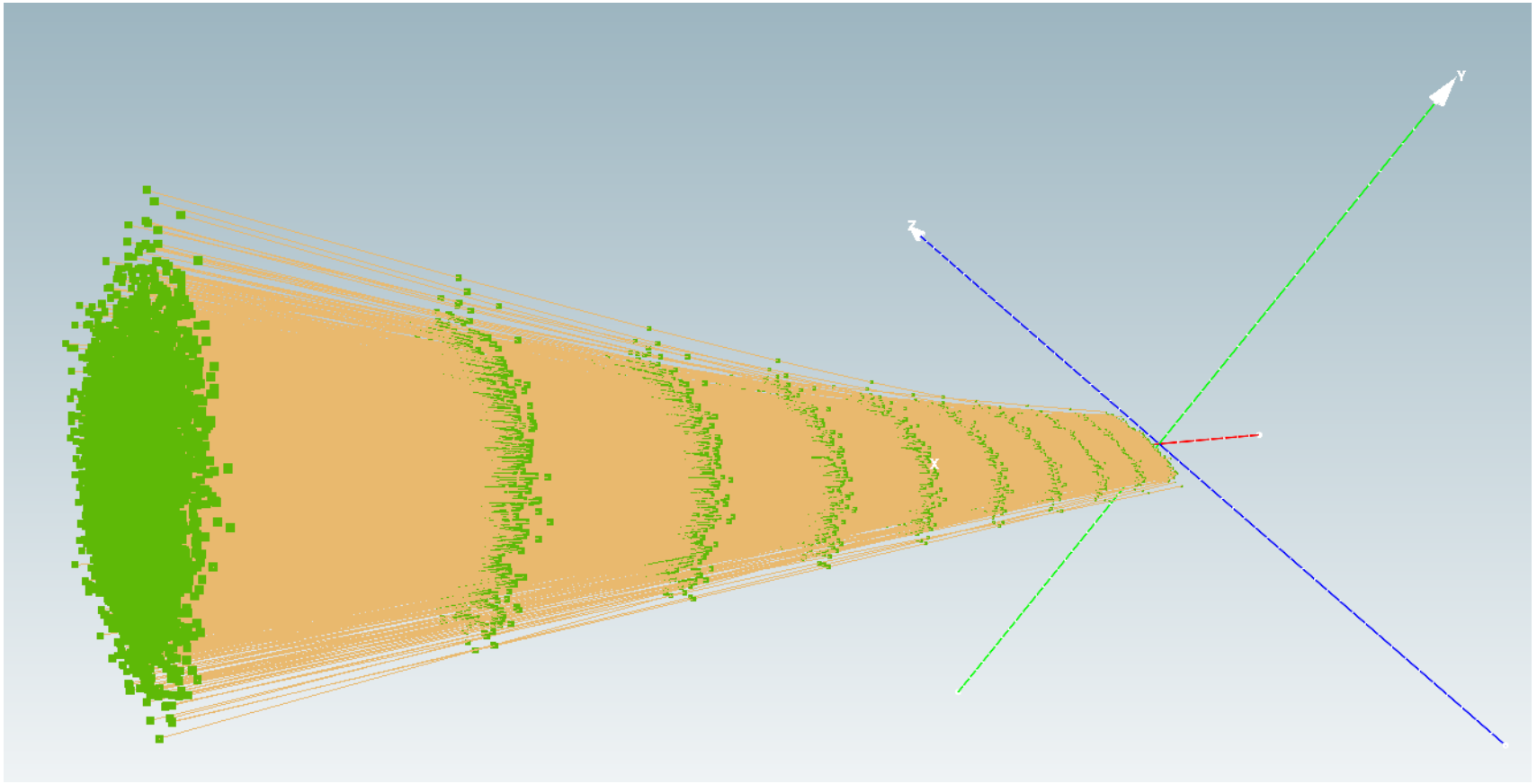


backup

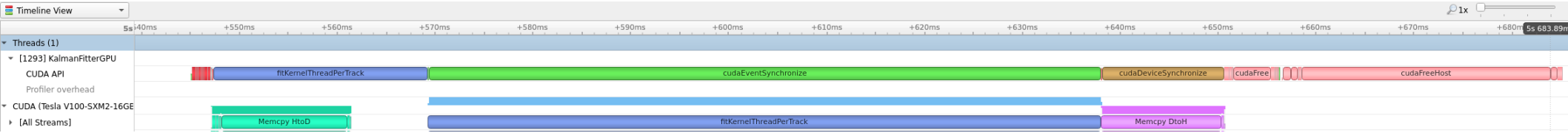
Computing environment

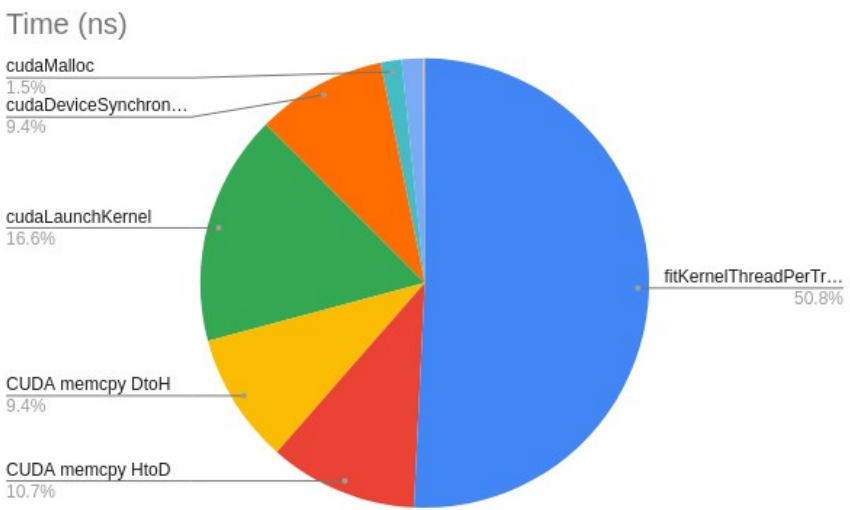
- Cori Haswell node: 2 Intel Xeon E5-2698v3 (2.3GHz, 16 cores, 32 threads, 40MB Cache)
- Intel Core i7-6700: 3.4GHz, 4 cores, 8 threads, 34.1GB/s, 8MB Cache
  - Intel HD Graphics 530: 350MHz, 24 execution units, 364.8 GFLOPS (FP32)
- Intel Core i9-9900K: 3.6GHz, 8 cores, 16 threads, 41.6GB/s, 16MB Cache
  - Intel UHD Graphics 630: 350 MHz, 24 execution units, 403.2 GFLOPS (FP32)
- NVIDIA GeForce GTX 960: 1127MHz, 1024 cuda cores, 2GB GDDR5 at 112GB/s, 2.413 TFLOPS (FP32)
- NVIDIA GeForce RTX 2060: 1365MHz, 1920 cuda cores, 6GB GDDR6 at 336GB/s, 6.4TFLOPS (FP32)
- NVIDIA Tesla V100: 1455MHz, 5120 cuda cores, 16GB HBM2 at 900GB/s, 15TFLOPS (FP32)

Fitting test



KF fitting profiling





10k tracks on V100

Multi-threaded event processing in ACTS

- Multiple events are processed concurrently with many threads through Intel TBB library
- An AlgorithmContext is used to support contextual condition data
  - Geometry/Calibration/Magnetic field

```
// execute the parallel event loop
tbb::task_scheduler_init init(m_cfg.numThreads);
tbb::parallel_for(
    tbb::blocked_range<size_t>(eventsRange.first, eventsRange.second),
    [&](const tbb::blocked_range<size_t>& r) {
      std::vector<Duration> localClocksAlgorithms(names.size(),
                                                  Duration::zero());

      for (size_t event = r.begin(); event != r.end(); ++event) {
        // Use per-event store
        WhiteBoard eventStore(Acts::getDefaultLogger(
            "EventStore#" + std::to_string(event), m_cfg.logLevel));
        // If we ever wanted to run algorithms in parallel, this needs to be
        // changed to Algorithm context copies
        AlgorithmContext context(0, event, eventStore);
        size_t ialgo = 0;
```

```
size_t algorithmNumber;            ///< Unique algorithm identifier
size_t eventNumber;                ///< Unique event identifier
WhiteBoard& eventStore;            ///< Per-event data store
Acts::GeometryContext geoContext;  ///< Per-event geometry context
Acts::MagneticFieldContext
    magFieldContext;                   ///< Per-event magnetic Field context
Acts::CalibrationContext calibContext;  ///< Per-event calbiration context
```