October 19 - 25, 2024
CHEP 2024
Conference on Computing in High Energy and Nuclear Physics

traccc: GPU track reconstruction library for HEP experiments

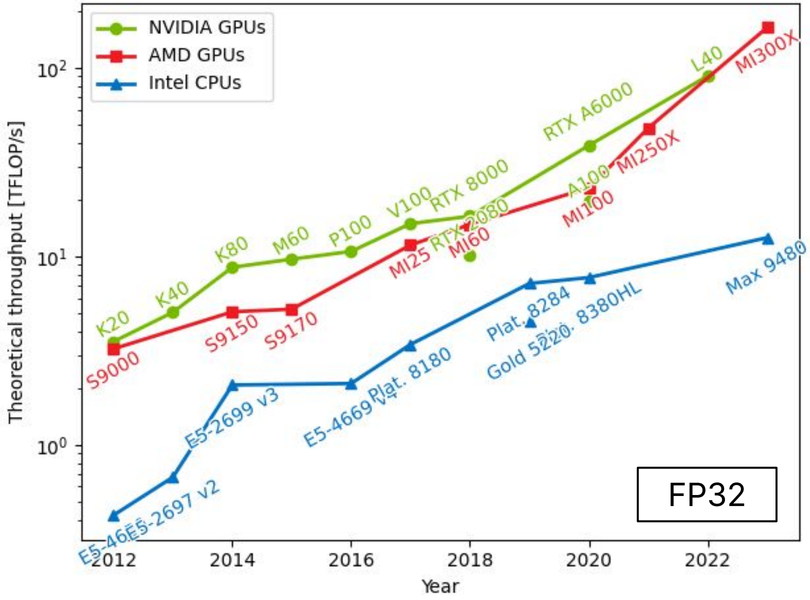
Heather Gray[1,2], Attila Krasznahorkay[3], Charles Leggett[2], Joana Niermann[4], Andreas Salzburger[4], Stephen Nicholas Swatman[4], **Beomki Yeo**[1,2]

[1] *University of California, Berkeley*
[2] *Lawrence Berkeley National Laboratory*
[3] *University of Massachusetts Amherst*
[4] *European Organization for Nuclear Research*

Motivation for GPU Track Reconstruction

- In upcoming HEP experiments, data analysis will be challenged by massive amount of data
- GPUs could transform our data processing capabilities as they can outperform CPUs for parallelizable algorithms with large data
- Track reconstruction is a good place to use GPUs as it is parallelizable and computationally heavy
  - Driven by A Common Tracking Software (ACTS) developers through **traccc** library

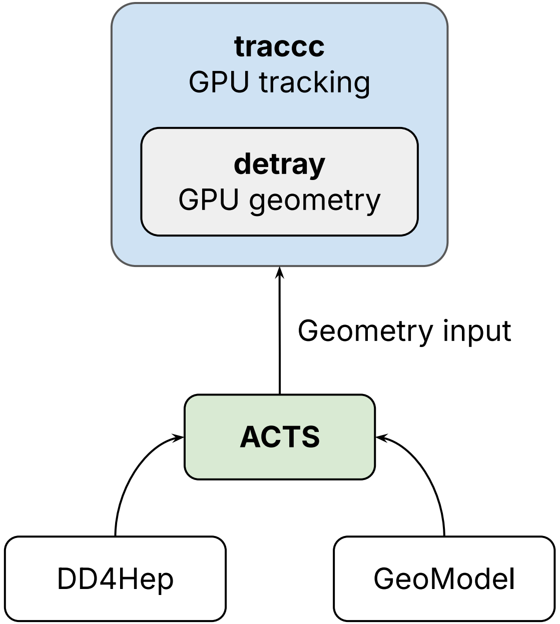
Theoretical throughput = # cores × FLOPS/cycle × cycles/s



Courtesy of S. Swatman

traccc Project Overview

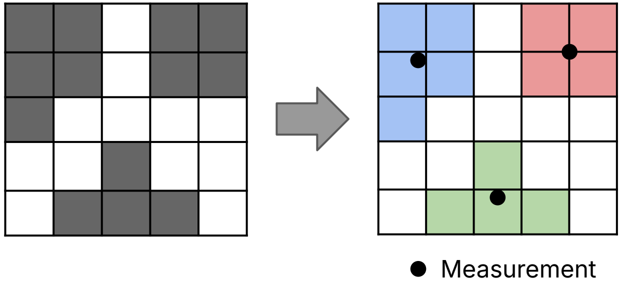
- traccc: GPU track reconstruction library
  - Main platforms: CPU, CUDA and SYCL
  - Supports both FP32 and FP64 precisions
  - R&D ongoing for ATLAS Phase II and CEPC
    (See the next talk by Y. Zhang for CEPC)
- detray: GPU geometry library used in traccc
  - Si-based detectors
  - Gaseous detectors with wire measurements
- ACTS: An experiment independent track reconstruction library
  - A bridge to general detector libraries (DD4Hep and GeoModel) for traccc



traccc Algorithms

1. Hit clusterization
   - Creating measurements from Si-based detector readouts
2. Seeding
   - Finding sub-patterns made of three measurements
3. Track finding (Combinatorial Kalman Filter)
   - Finding full patterns of all measurements
4. Track fitting (Kalman Filter)
   - Fitting the patterns to precise tracks

Hit clusterization



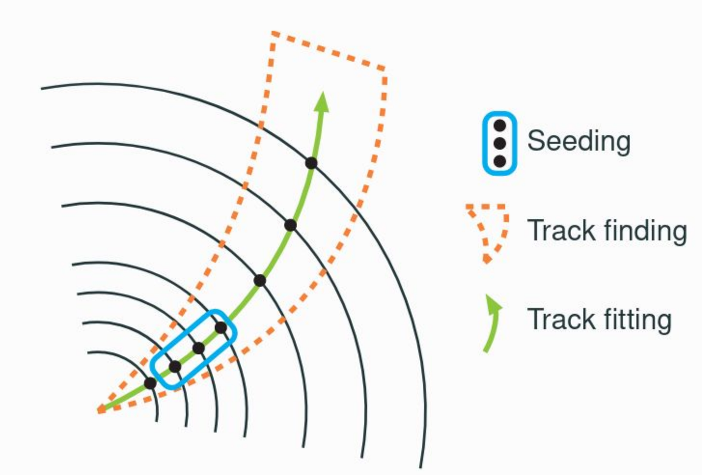
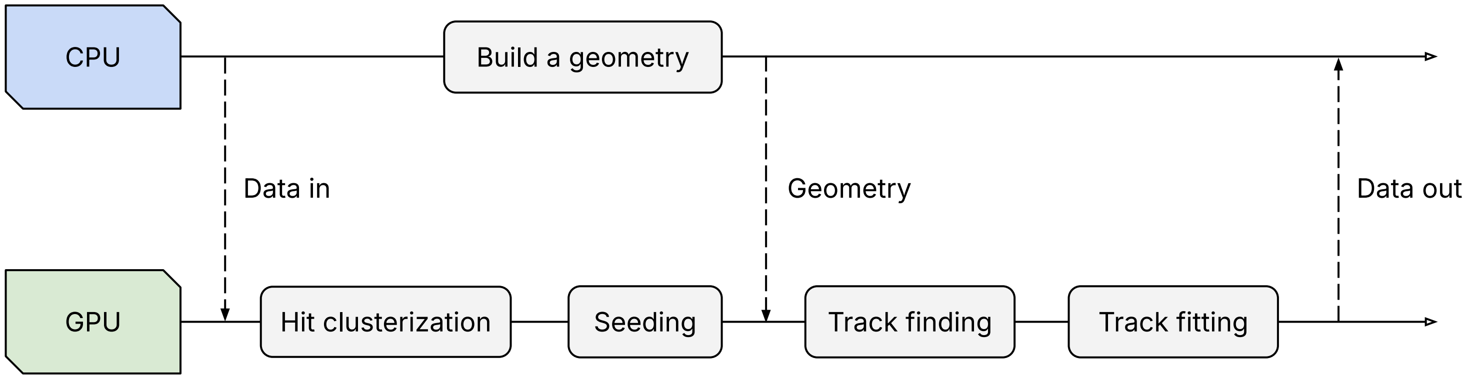


Image by P. Gessinger

traccc Workflow

- End-to-end analysis in the GPU
  - Minimize the memory transfer between CPU and GPU
- The geometry is built and transferred only once
  - Ongoing development for the alignment of detector modules

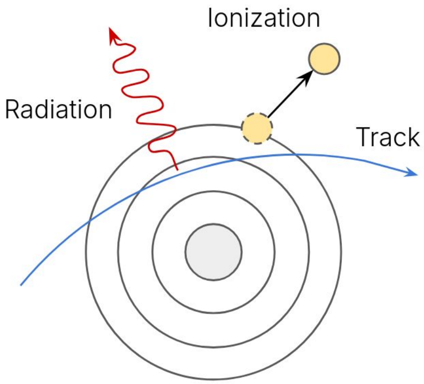


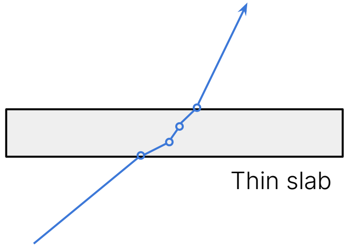
Track Propagation Model

- Track Propagation and covariance transport based on the adaptive fourth-order Runge-Kutta-Nyström method
  [L. Bugge, J. Myrheim (1981), E. Lund, et al (2009), E. Lund, et al (2009), B. Yeo, et al (2024)]

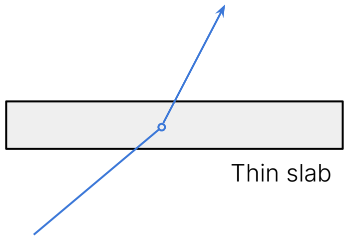
- Material interactions
  - Ionization and radiative energy loss
  - Multiple scattering

- Thin scatterer approximation to simplify the propagation inside thin detector planes
  [R. Frühwirth, A. Strandlie (2021)]





Volume-based geometry (e.g. Geant4)

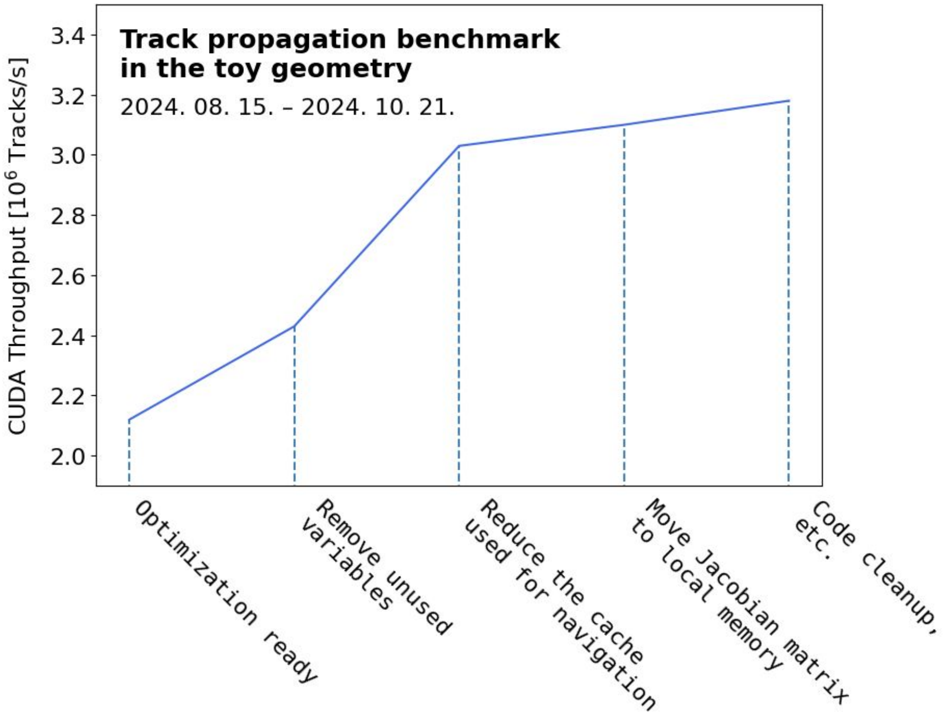


Thin scatterer approximation

Track Propagation Optimization

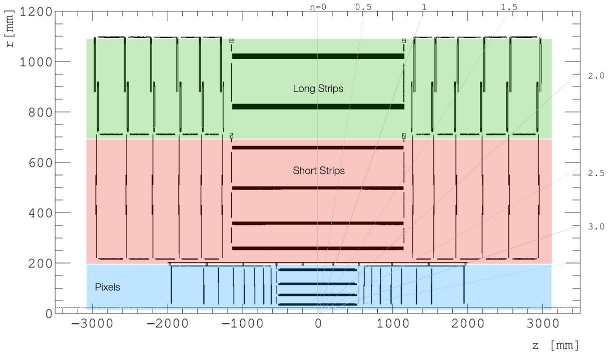
- Track propagation is the main bottleneck of the full chain, hence, prioritized for optimization
  - Move variables in the global memory to the local memory, if possible
  - Minimize the cache memory used for geometry navigation

Further optimization is planned to maximize the computing performance gain

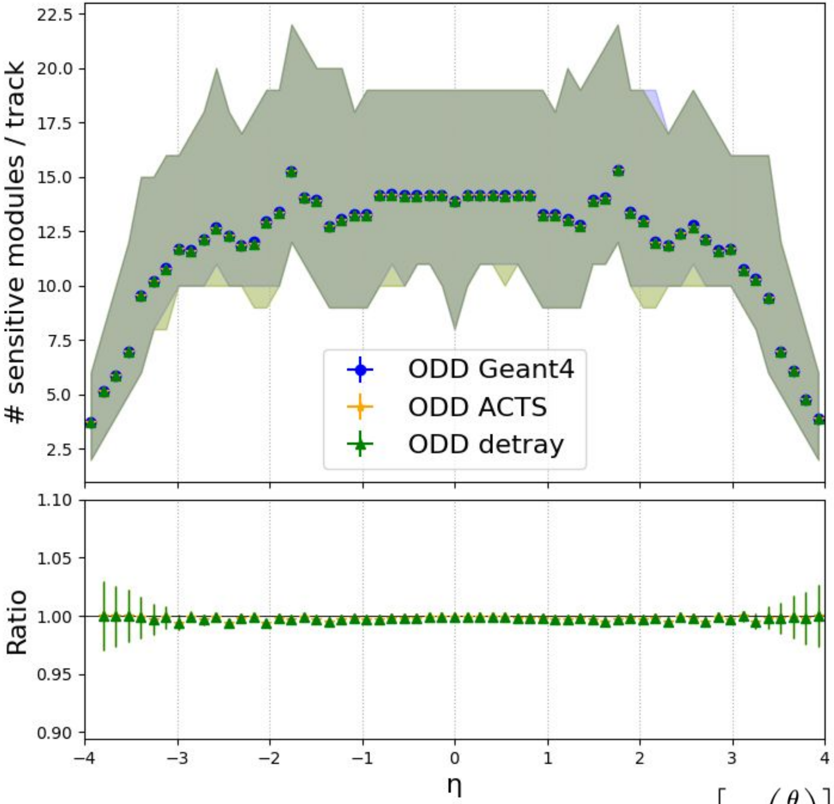


Geometry Validation against Geant4 and ACTS

- Compare the number of sensitive modules (readout modules) intersected by straight rays in Open Data Detector (ODD)
  - ODD is a generic detector developed for software R&D
  - Made of pixel and strip detectors



Geant4 hit recorded in the ODD



$$\eta \equiv -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$$

GPU Computation Validation against CPU

- GPU computation results can be different from CPU
  - Different standard in floating point calculation
  - The sequence of arithmetic executions may not be deterministic (atomic operations)

- Compared the track finding and fitting results between CPU and GPU
  - Used a ODD ttbar event with 140 pileup, simulated by Geant4
  - Ongoing study to understand the difference in FP32 track fitting

	Tolerated Diff.	Matching ratio (FP32)	Matching ratio (FP64)
Track finding pattern	-	94.7%	99.4 %
Track fitting parameter	0.01%	37.8%	91.8 %
	0.1%	52.4%	93.5 %
	1%	65.9%	95.3 %
	5%	75.4%	96.5 %

* Track finding pattern is considered to be matched when all measurements in the pattern are the same
* Tolerated Diff. is the tolerated relative difference between CPU and GPU track parameters

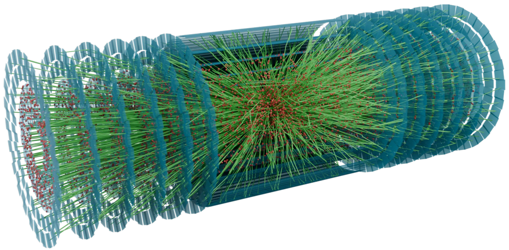
Physics Validation

- Track finding shows a good tracking efficiency for ttbar pileup (μ) event

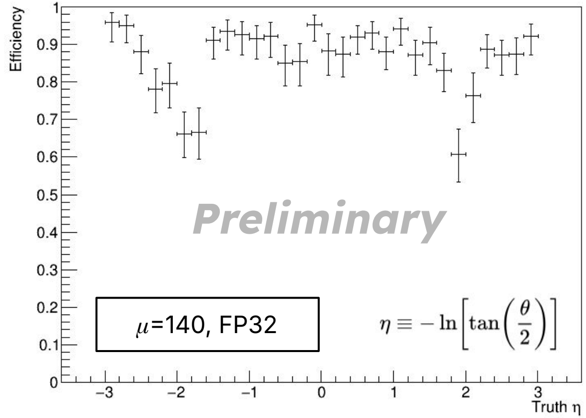
$$\text{Tracking efficiency} \\ = \frac{\text{The number of particles matching any of found tracks}}{\text{The number of particles}}$$

- Particle matching conditions (*Double matching*)
  - The number of measurements from the major particle is larger than a half of the number of the measurements of the found track
  - The number of measurements from the major particle is larger than a half of the number of measurements produced by the particle

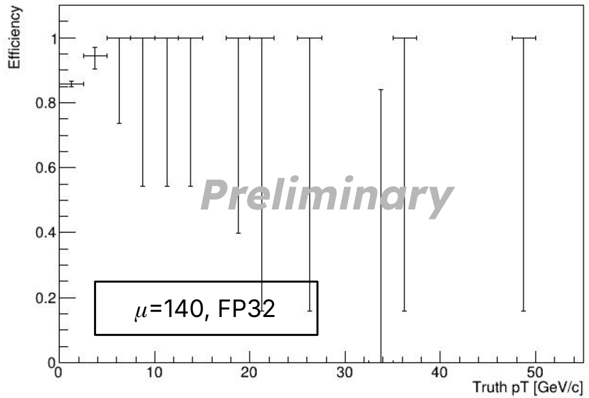
- Particle cuts:
  - Charged
  - p_T > 0.5 GeV/c
  - Vertex with $|z|$ < 500 mm and r < 200 mm



Tracking efficiency vs. η



Tracking efficiency vs. p_T



Computing Performance Benchmark Setup

- Used the Geant4 ODD data (μ in the range of [20, 300])
- Measure the event throughputs of both traccc CPU and CUDA implementations
  - CPU with multi-threading
  - CUDA with multiple streams where multiple pipelines running asynchronously

Benchmarked devices

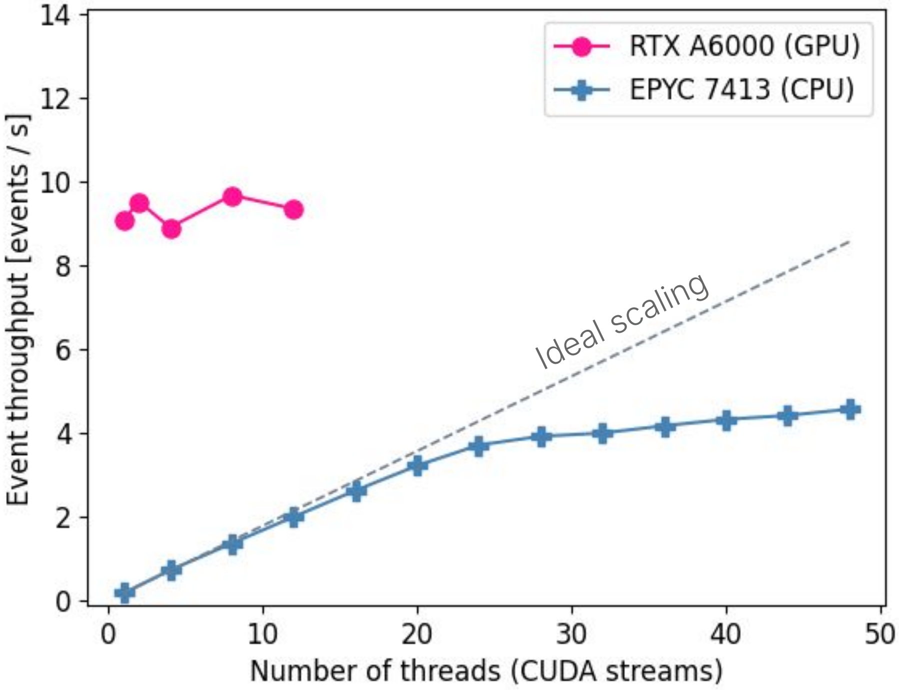
	Model	Cores	Mem.	TDP	FP64:FP32
GPU	RTX A6000	10752	48 GB	300 W	1:64
	RTX 2080 SUPER	3072	8 GB	250 W	1:32
	A30	3584	24 GB	165 W	1:2
CPU	AMD EPYC 7413	24	-	180 W	-

* TDP = Thermal Design Power (Maximum theoretical load)
* FP64:FP32 = Theoretical performance ratio between FP64 and FP32

Computing Performance Scaling

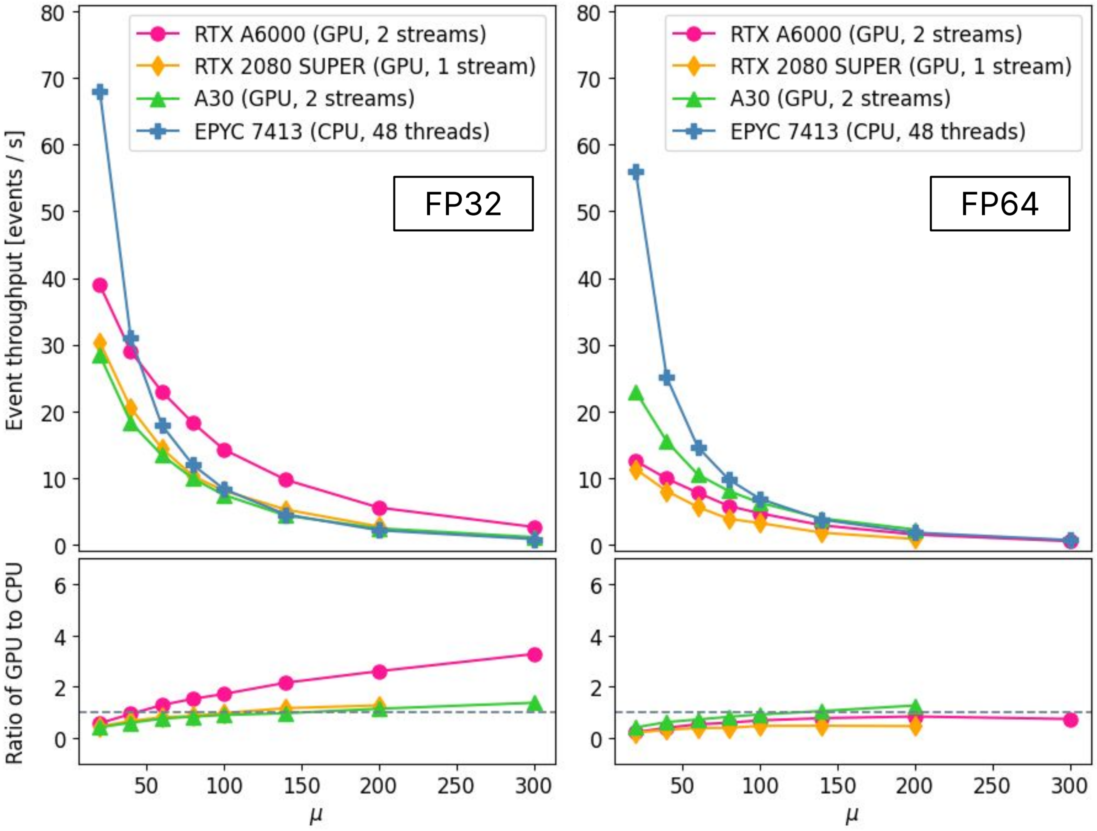
- CPU performance scales linearly until the number of its physical cores, i.e. 24
- GPU performance scales weakly and tends to fluctuate

μ=140, FP32



Event Throughput vs. Pileup

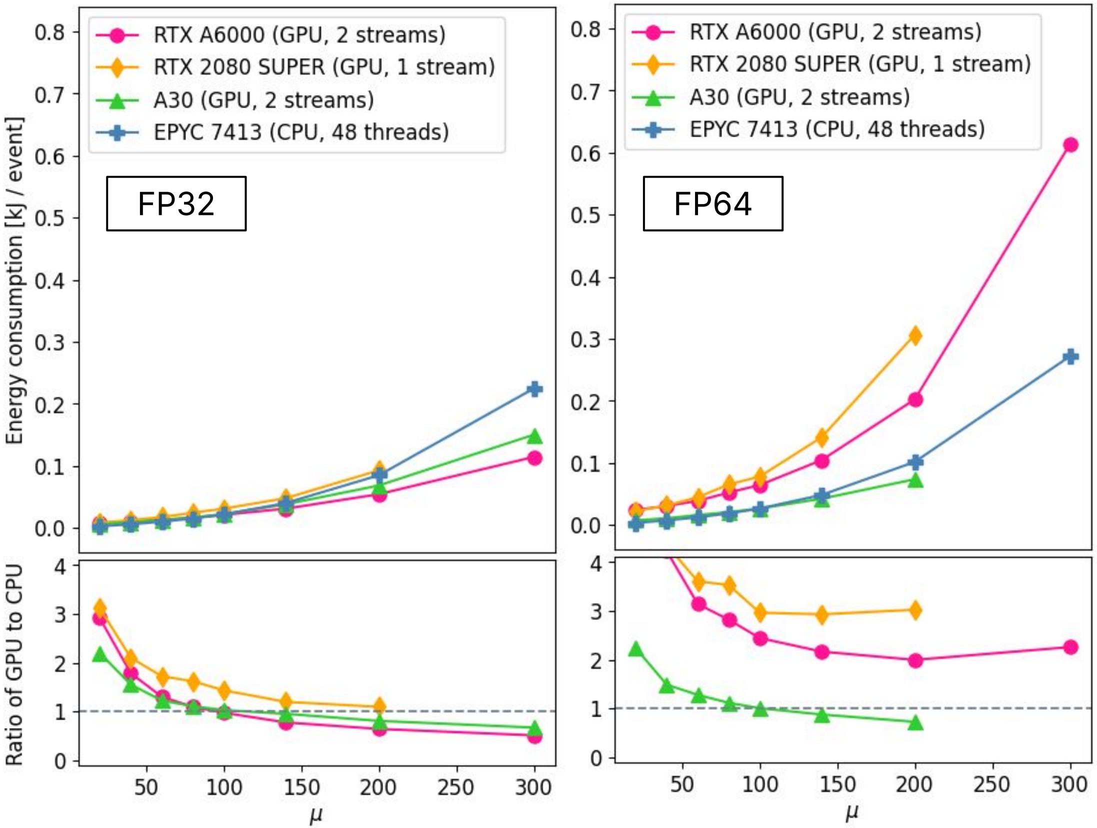
- The gain from GPU increases with larger data
- For FP64, A30 shows better performance than the CPU for large pileup events



Energy Consumption vs. Pileup

Energy consumption per event
= TDP / Event throughput

- Important metrics for sustainability and carbon neutrality
- For FP32, GPUs are more energy efficient with large pileup events
- For FP64, RTX models are less energy efficient



Summary and Prospects

- traccc is the GPU track reconstruction library as an R&D project of ACTS

- The geometry has been validated with ray tracing, and the algorithms show good performance against ODD Geant4 data

- We demonstrate that GPUs can be faster and energy efficient than multi-threaded CPUs for large data set
    - Further optimization will follow

- Validating the computation with FP32 will be an important task, as the performance gain from GPU is much higher with FP32

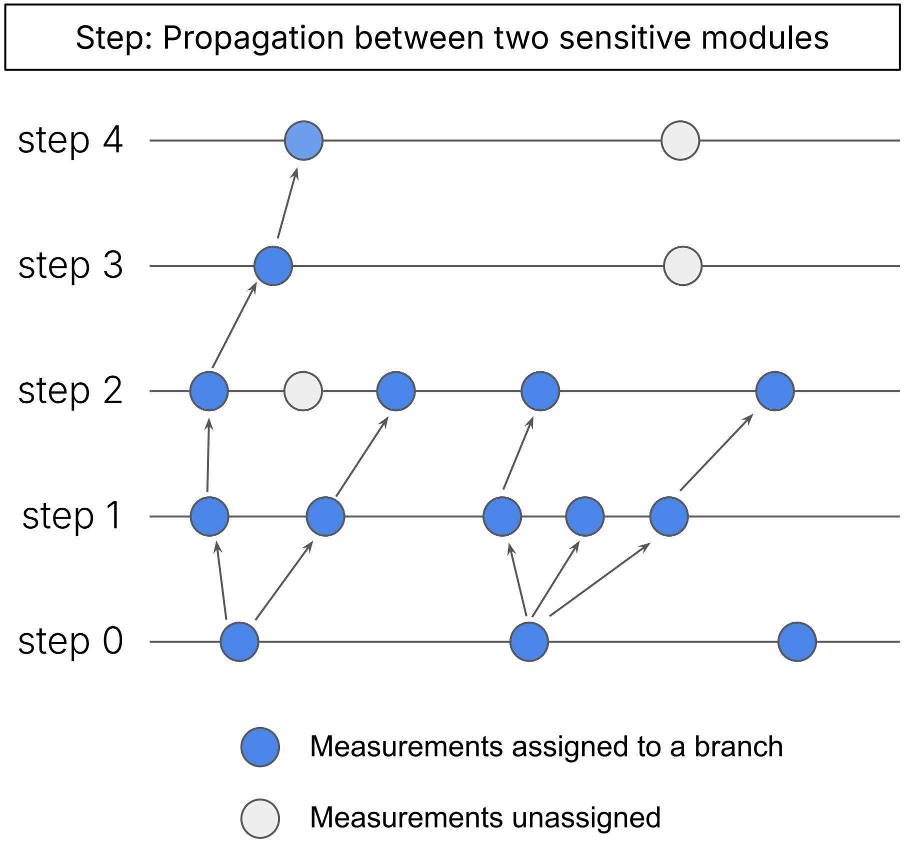
Backup Slides

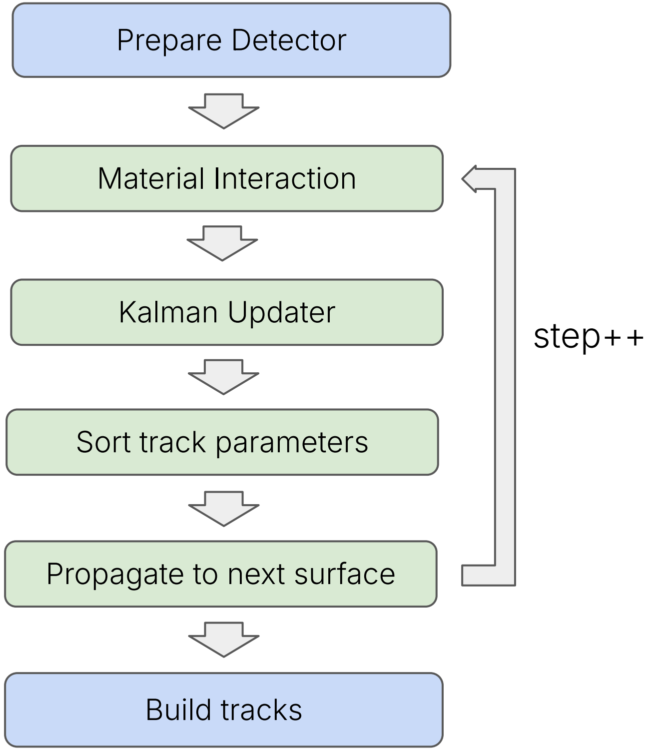
traccc Project Status

Category	Algorithms	CPU	CUDA	SYCL	Alpaka	Kokkos	Futhark
Clusterization	CCL / FastSv / etc.	✅	✅	✅	🟡	⚪	✅
	Measurement creation	✅	✅	✅	🟡	⚪	✅
Seeding	Spacepoint formation	✅	✅	✅	🟡	⚪	⚪
	Spacepoint binning	✅	✅	✅	✅	✅	⚪
	Seed finding	✅	✅	✅	✅	⚪	⚪
	Track param estimation	✅	✅	✅	✅	⚪	⚪
Track finding	Combinatorial KF	✅	✅	🟡	🟡	⚪	⚪
Track fitting	KF	✅	✅	✅	⚪	⚪	⚪
Ambiguity resolution	Greedy resolver	✅	⚪	⚪	⚪	⚪	⚪

✅: exists, 🟡: work started, ⚪: work not started yet

GPU CKF Algorithm

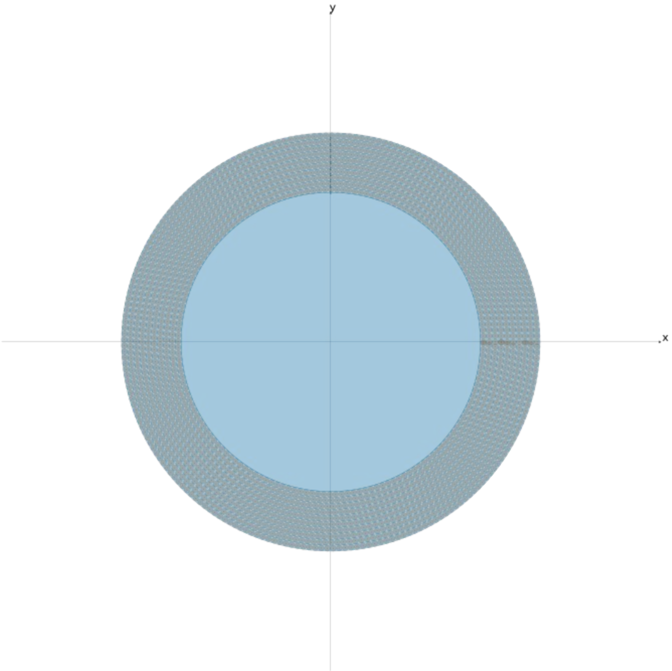




*Each box corresponds to a GPU kernel

Gaseous Detector Support

- Detray fully supports the gaseous detector (drift chamber and straw tubes) and track propagation in them
- The normal Kalman Filter of traccc works fine with a seed of good quality, but it can not handle the left-right ambiguity perfectly
  - Extension of normal Kalman Filter or least-squares regression will be implemented



Drift chamber built by detray