# traccc

## *Track Reconstruction on GPUs in Acts*

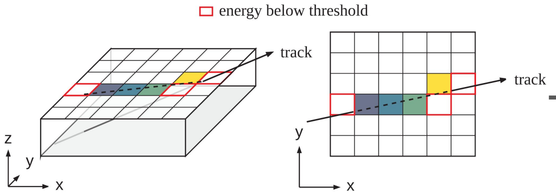
Attila Krasznahorkay
*on behalf of* ***a lot of*** *people…*

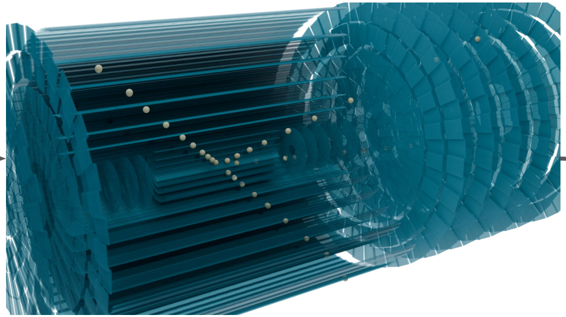
HEP Software Foundation

WLCG

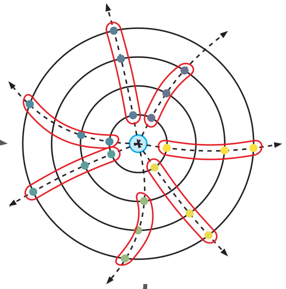
# (Classical) Track Finding 101

Clusterization, measurement and spacepoint creation

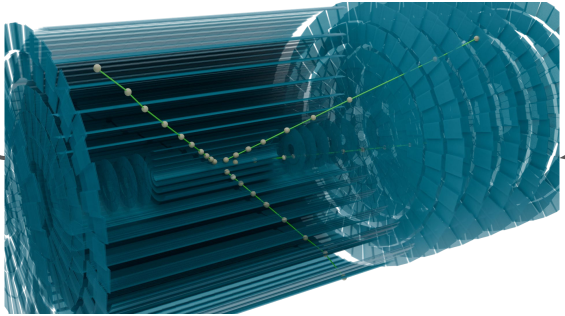


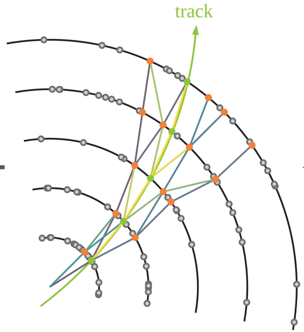



Track seeding

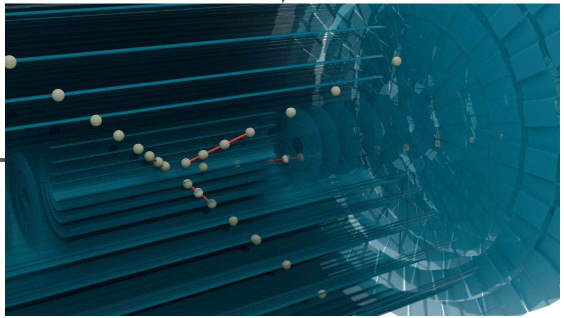


Ambiguity resolution



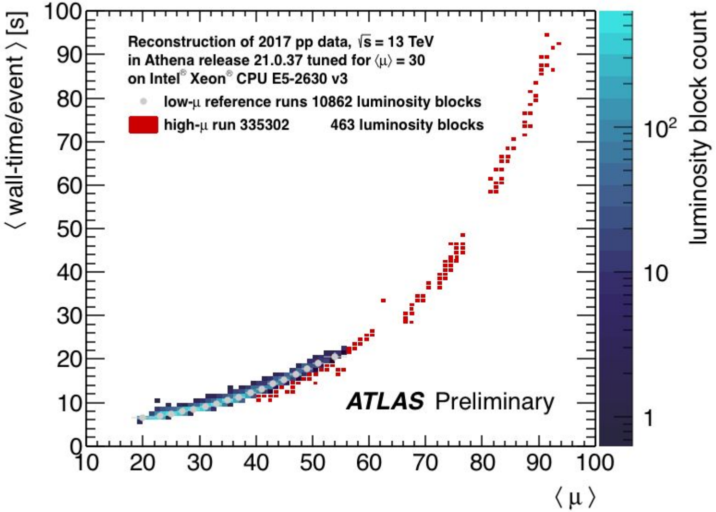


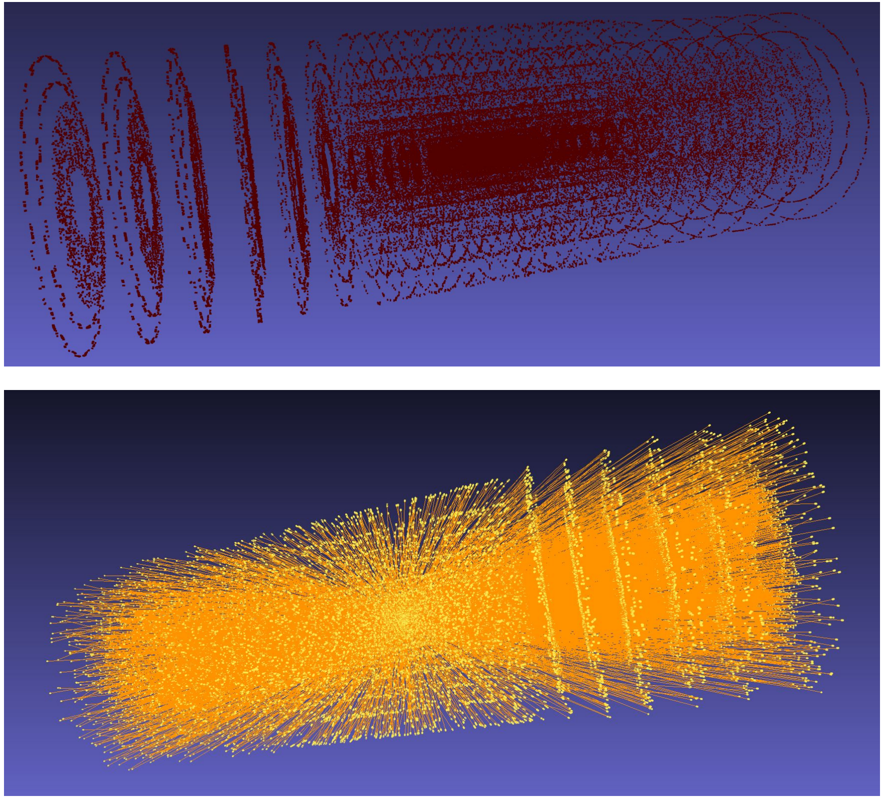

Track finding



# The Need For GPUs

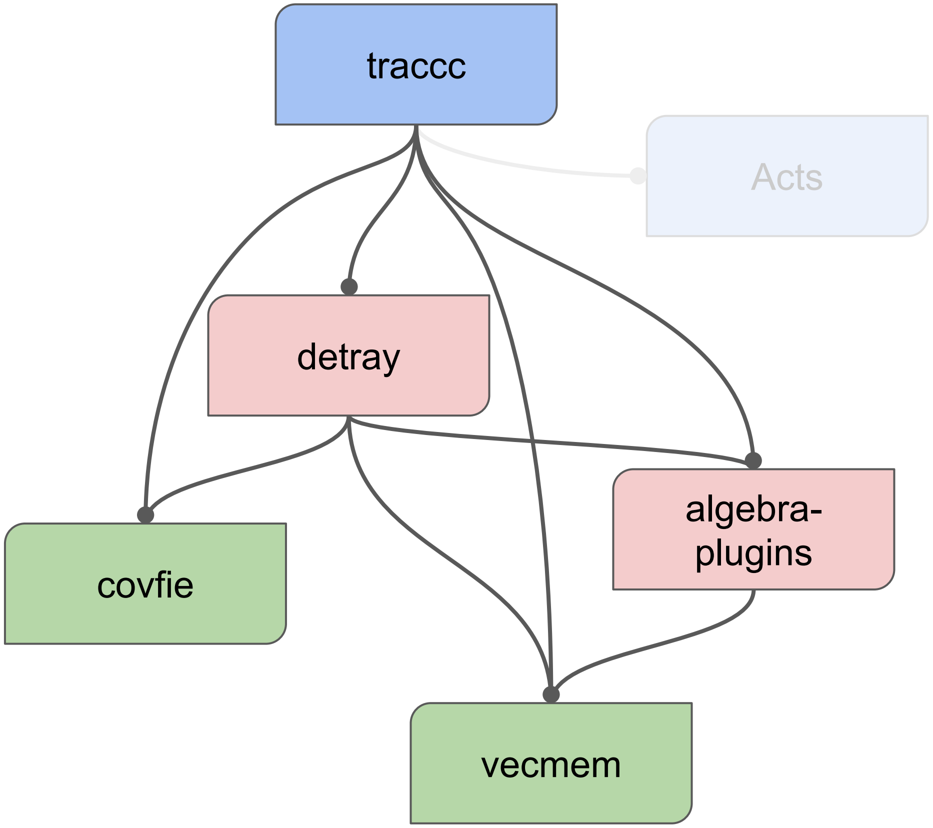
- The sort of events that we will need to reconstruct during the HL-LHC, are the ones shown here
  - On which the combinatorics of our algorithms explode



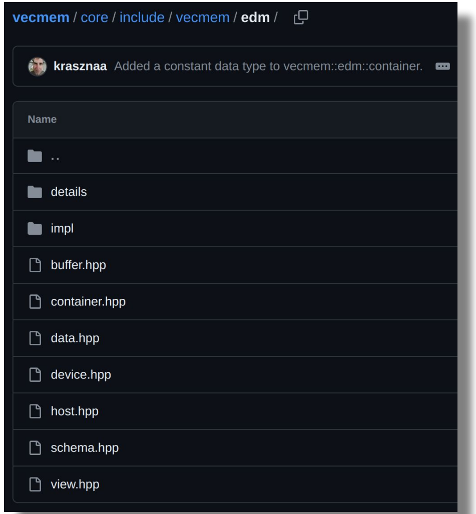
$t\bar{t}$ event in the ODD at μ = 200

# The Acts Parallelization R&D




- To explore fundamentally new ways for reconstructing particle tracks, we created a set of “standalone” projects
  - With the top reconstruction algorithms sitting in traccc, and all other projects serving various purposes for making that happen
- The overall goal is to demonstrate that we could run track reconstruction on GPUs without any shortcuts in reconstruction / physics quality
  - Using the same (type of) combinatorial Kalman filtering used by Acts, with detector geometry and magnetic field modeled at the same level of accuracy

# Base Projects



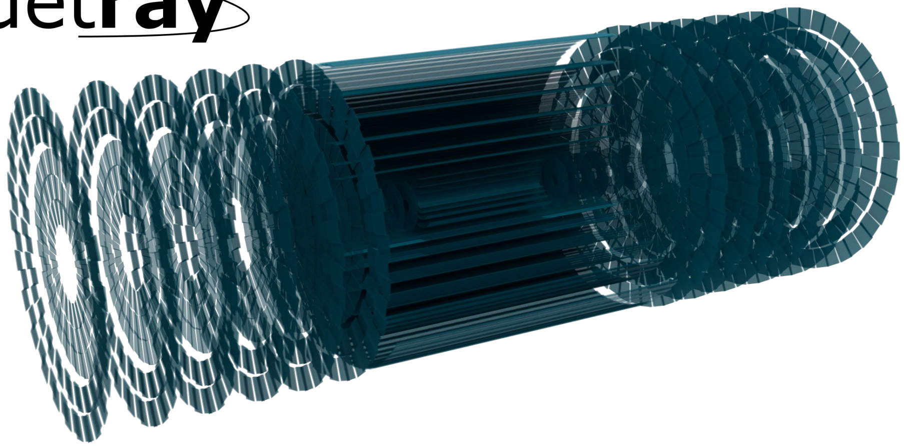

- Good technical work has happened in vecmem, algebra-plugins and covfie
  - But those are not the main things for today..
- vecmem introduced basic support for SoA containers
  - But they did not make their way into traccc yet
- algebra-plugins improved its vectorization support in host code
  - Both for auto- and explicit-vectorization


Current contributors:
Joana Niermann, Beomki Yeo, Stephen Swatman

# detray

- Is maybe our most ambitious project
- It provides a surface based geometry for tracking, with efficient navigation / propagation support between the surfaces
    - Including the management of surface material and magnetic field during the navigation
- All implemented **without** using “GPU hostile” programming methods
    - Virtual inheritance, dynamic memory allocation, etc.

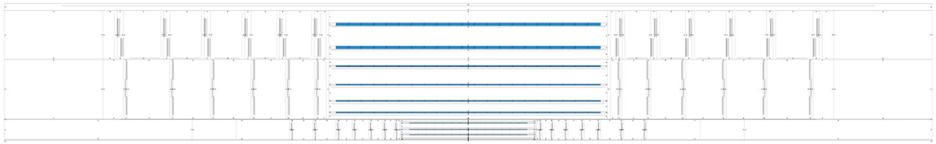
detray



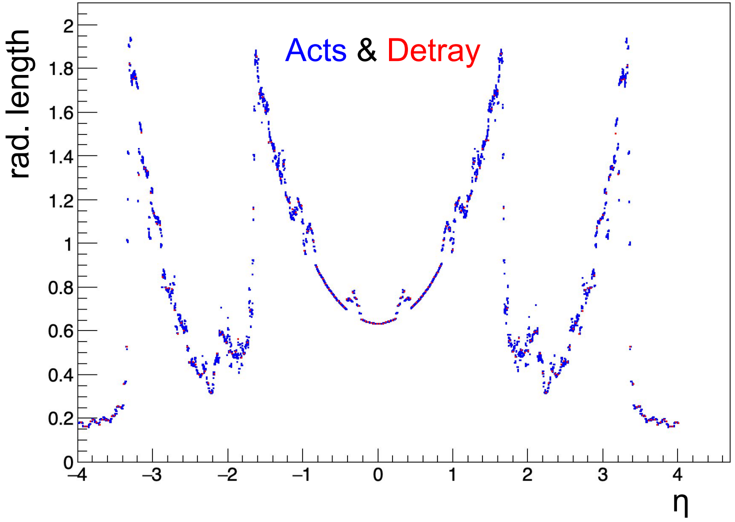
Current contributors:
Joana Niermann, Beomki Yeo, Andreas Salzburger, Frederik Verdoner Barba, Eleni Xochelli, Stephen Swatman

# Latest Developments

- After updates in Acts and ODD, created JSON descriptions of the ODD for Detray
  - Including the properly defined "surface grids" and "material maps"
  - Still a little manually for these tests, but will make it a lot more automatic soon. Making it possible to convert any "Acts geometry" to a Detray one.
- Can now exactly reproduce the behaviour of Acts's existing tracking geometry code
  - Material mapping comparisons on device to some soon, current comparisons all done in host code.
- Tons of technical developments done to make it all happen…



ODD surfaces and grids



# traccc

| 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

Current contributors:
Beomki Yeo, Joana Niermann, Ryan Joseph Cross, Stewart Martin-Haugh, Shima Shimizu, Sylvain Joube, Stephen Swatman

- It is our main repository, combining the capabilities of all of the other ones
  - GPU code development initially happens in CUDA most of the time, then generalising it to work with SYCL, Alpaka, etc. as well.
- As was the original goal, significant code sharing is achieved between the host and device, and the different device implementations
  - Technically in all cases happening through shared, inlined functions (working on “GPU friendly” data types)

# Reconstruction Algorithm Status

- The full ODD reconstruction chain now works on the host and with CUDA! 🎉
  - Without ambiguity resolution... For that we still only have an algorithm for the host.
  - Technically the “full CUDA chain” can fit on a single screen! 😜
- Geant4 simulation files for its input can now be produced using Acts’s main branch
  - See: [acts-project/acts#3169](acts-project/acts#3169)

```
device
  alpaka
  common
  cuda
    include/traccc/cuda
      clusterization
      finding
        finding_algorithm.hpp
      fitting
      seeding
      utils
    src
    CMakeLists.txt
  futhark
  kokkos
  sycl
    include/traccc/sycl
      clusterization
      fitting
        fitting_algorithm.hpp
      seeding
      utils
    src
    CMakeLists.txt
```

```cpp
full_chain_algorithm::output_type full_chain_algorithm::operator()(
    const cell_collection_types::host& cells,
    const cell_module_collection_types::host& modules) const {

    // Create device copy of input collections
    cell_collection_types::buffer cells_buffer(cells.size(),
                                               *m_cached_device_mr);
    m_copy(vecmem::get_data(cells), cells_buffer)->ignore();
    cell_module_collection_types::buffer modules_buffer(modules.size(),
                                                        *m_cached_device_mr);
    m_copy(vecmem::get_data(modules), modules_buffer)->ignore();

    // Run the clusterization (asynchronously).
    const clusterization_algorithm::output_type measurements =
        m_clusterization(cells_buffer, modules_buffer);
    m_measurement_sorting(measurements);

    // Run the seed-finding (asynchronously).
    const spacepoint_formation_algorithm::output_type spacepoints =
        m_spacepoint_formation(measurements, modules_buffer);
    const track_params_estimation::output_type track_params =
        m_track_parameter_estimation(spacepoints, m_seeding(spacepoints),
                                     m_field_vec);

    // If we have a Detray detector, run the track finding and fitting.
    if (m_detector != nullptr) {

        // Create the buffer needed by track finding and fitting.
        auto navigation_buffer = detray::create_candidates_buffer(
            *m_detector,
            m_finding_config.navigation_buffer_size_scaler *
                m_copy.get_size(track_params),
            *m_cached_device_mr, &m_host_mr);

        // Run the track finding (asynchronously).
        const finding_algorithm::output_type track_candidates =
            m_finding(m_device_detector_view, m_field, navigation_buffer,
                      measurements, track_params);

        // Run the track fitting (asynchronously).
        const fitting_algorithm::output_type track_states =
            m_fitting(m_device_detector_view, m_field, navigation_buffer,
                      track_candidates);

        // Copy a limited amount of result data back to the host.
        output_type result{&m_host_mr};
        m_copy(track_states.headers, result)->wait();
        return result;
```

# Host <-> Device Agreement(?)

**FP32**

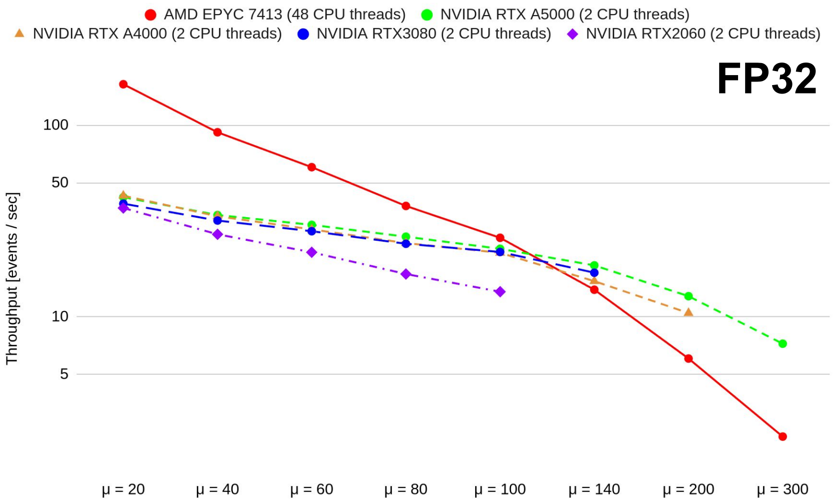
```
===>>> Event 1 <<<===
Number of measurements: 637 (host), 637 (device)
  Matching rate(s):
    - 98.2732% at 0.01% uncertainty
    - 99.843% at 0.1% uncertainty
    - 99.843% at 1% uncertainty
    - 99.843% at 5% uncertainty
Number of spacepoints: 637 (host), 637 (device)
  Matching rate(s):
    - 98.2732% at 0.01% uncertainty
    - 99.843% at 0.1% uncertainty
    - 99.843% at 1% uncertainty
    - 99.843% at 5% uncertainty
Number of seeds: 96 (host), 96 (device)
  Matching rate(s):
    - 72.9167% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
Number of track parameters: 96 (host), 96 (device)
  Matching rate(s):
    - 60.4167% at 0.01% uncertainty
    - 96.875% at 0.1% uncertainty
    - 98.9583% at 1% uncertainty
    - 100% at 5% uncertainty
Number of track candidates (header): 108 (host), 108 (dev
  Matching rate(s):
    - 62.963% at 0.01% uncertainty
    - 96.2963% at 0.1% uncertainty
    - 99.0741% at 1% uncertainty
    - 100% at 5% uncertainty
  Track candidates (item) matching rate: 100%
Number of track states: 108 (host), 108 (device)
  Matching rate(s):
    - 44.4444% at 0.01% uncertainty
    - 94.4444% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
```

**FP64**

```
===>>> Event 1 <<<===
Number of measurements: 637 (host), 637 (device)
  Matching rate(s):
    - 100% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
Number of spacepoints: 637 (host), 637 (device)
  Matching rate(s):
    - 100% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
Number of seeds: 96 (host), 96 (device)
  Matching rate(s):
    - 100% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
Number of track parameters: 96 (host), 96 (device)
  Matching rate(s):
    - 100% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
Number of track candidates (header): 108 (host), 108 (device)
  Matching rate(s):
    - 100% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
  Track candidates (item) matching rate: 100%
Number of track states: 108 (host), 108 (device)
  Matching rate(s):
    - 100% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
```

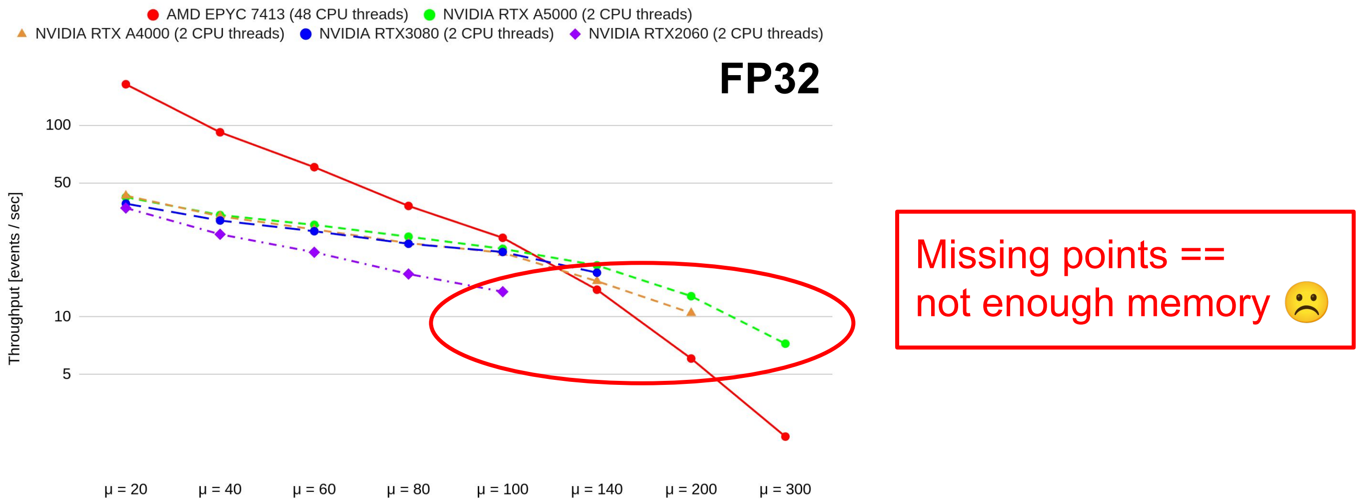
- Our main “development applications” are ones executing the algorithms one by one, checking their outputs at every step
  - Allowing us to measure the “physics performance” of the code, and to compare results between different implementations of the same algorithm
- At FP32/single precision, agreement between the host and GPU is not perfect. But it’s also not terrible.
  - While at FP64/double precision the GPU code finds the exact same tracks, with the exact same properties.

# ODD Reconstruction Compute Performance




- We also have tests that load N events into (host) memory, and process them over- and over again to test the throughput of our algorithms
  - Just copying stuff back to the host at the end, but not analyzing the output of the reconstruction
- Even with the so far hardly optimized algorithms, we can beat a single “decent” CPU with a single “workstation” GPU *at HL-LHC luminosities*

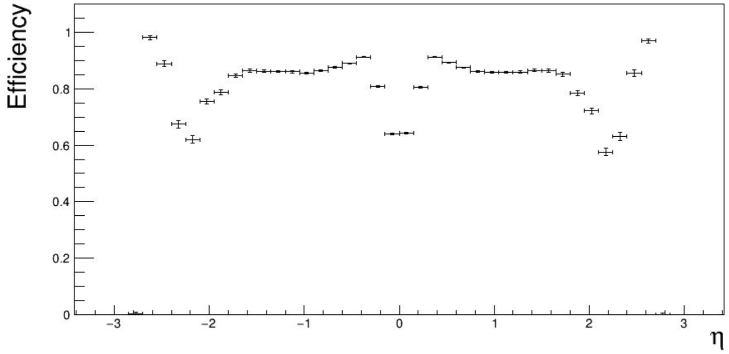
# ODD Reconstruction Compute Performance

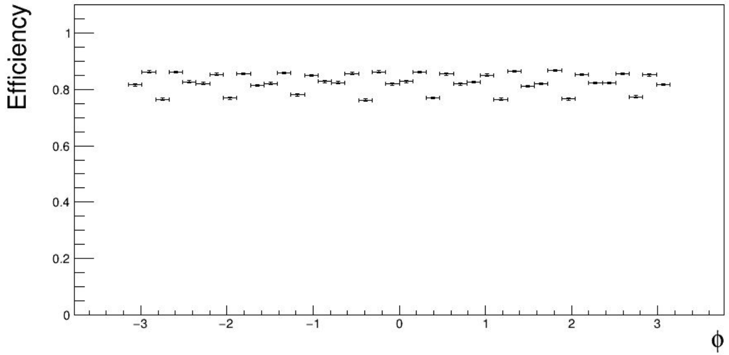



- We also have tests that load N events into (host) memory, and process them over- and over again to test the throughput of our algorithms
  - Just copying stuff back to the host at the end, but not analyzing the output of the reconstruction
- Even with the so far hardly optimized algorithms, we can beat a single “decent” CPU with a single “workstation” GPU *at HL-LHC luminosities*

# ODD Reconstruction Physics Performance

- Makes it very clear that all compute performance numbers are to be taken with some salt
  - These efficiencies (for high-$p_T$ muons) should be ~100%. We will make sure that they would be.
- With this in mind, such efficiencies without any ODD specific settings for our code, are not a terrible starting point 🤔



# The Bugs / Next Steps

- With the full chain only starting to work a few weeks ago, and only running on larger simulation samples now (this week) for the first time, we are finding a lot of errors still…
  - I'm not too worried about this though
- We will need to demonstrate that the algorithms can find tracks in the ODD efficiently
  - Already identified a few places where our default algorithm configurations don't seem to work well
  - Making proper use of material maps during reconstruction will also help
- Will need to make the code work with ATLAS's HL-LHC inner detector geometry (ITk)
  - With the infrastructure developed with the ODD geometry, this should be a finite amount of effort
- We will switch to a fully-SoA Event Data Model from the current, naive AoS one
- Implement the missing algorithms with CUDA, SYCL, Alpaka, etc.
- Integrate everything into Acts!
  - With a unified UI with all the existing / CPU tools

# Summary

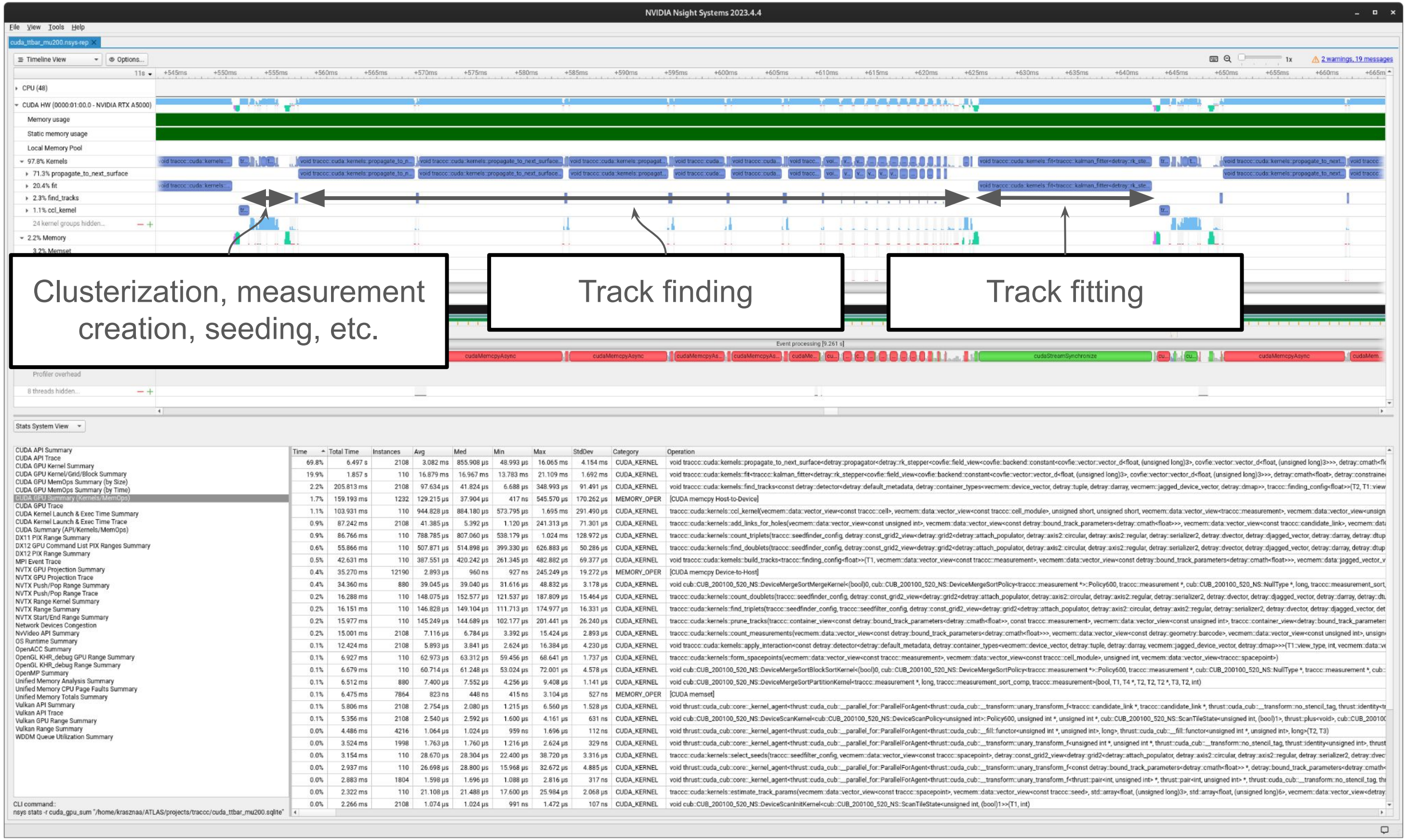
- I believe the future is bright for Act's GPU capabilities!
  - The very first version of the code that works on the ODD (v0.10.0), has a lot to improve still
  - However the performance, as is, makes me very hopeful already!
- Much of the current code is held together by sellotape, spit and blind luck…
  - But we have a plan for making it all a lot more robust, and (hopefully) significantly faster
- A lot of work already done, and a lot of good work still ahead of us! 😉

# Backup

# ODD Reconstruction Compute Performance

| Device | ttbar event processing rate [events / sec] | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | μ = 20 | μ = 40 | μ = 60 | μ = 80 | μ = 100 | μ = 140 | μ = 200 | μ = 300 |
| AMD EPYC 7413 (48 CPU threads) | 163.71 | 91.8513 | 60.359 | 37.8601 | 25.8034 | 13.8167 | 6.03643 | 2.35974 |
| NVIDIA RTX A5000 (2 CPU threads) | 42.0662 | 33.9328 | 30.1514 | 26.1469 | 22.6047 | 18.5172 | 12.7826 | 7.21733 |
| NVIDIA RTX A4000 (2 CPU threads) | 42.8472 | 33.4305 | 28.555 | 24.2146 | 21.5356 | 15.314 | 10.4362 | |
| NVIDIA RTX3080 (2 CPU threads) | 38.9144 | 31.7598 | 27.9324 | 24.0226 | 21.7591 | 16.9548 | | |
| NVIDIA RTX2060 (2 CPU threads) | 36.941 | 26.9102 | 21.679 | 16.6888 | 13.4879 | | | |

# Throughput Measurement Profile



CERN

http://home.cern