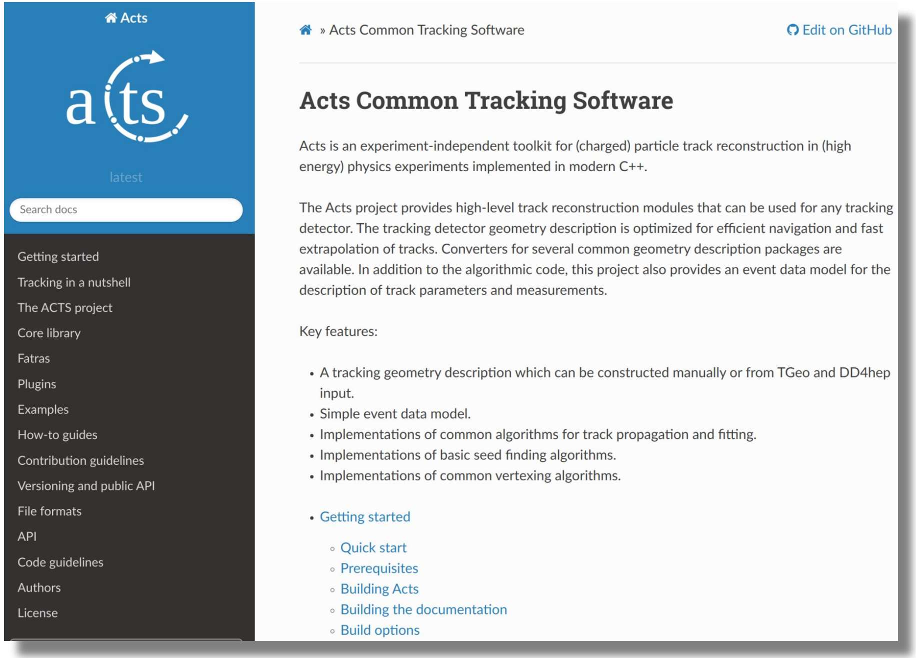
traccc - A (Close To) Single-Source Tracking Demonstrator on CPUs/GPUs

Attila Krasznahorkay

on behalf of the Acts Parallelization R&D Team

The Acts Project


Acts

latest

Search docs

Getting started
Tracking in a nutshell
The ACTS project
Core library
Fatras
Plugins
Examples
How-to guides
Contribution guidelines
Versioning and public API
File formats
API
Code guidelines
Authors
License

» Acts Common Tracking Software

Edit on GitHub

Acts Common Tracking Software

Acts is an experiment-independent toolkit for (charged) particle track reconstruction in (high energy) physics experiments implemented in modern C++.

The Acts project provides high-level track reconstruction modules that can be used for any tracking detector. The tracking detector geometry description is optimized for efficient navigation and fast extrapolation of tracks. Converters for several common geometry description packages are available. In addition to the algorithmic code, this project also provides an event data model for the description of track parameters and measurements.

Key features:

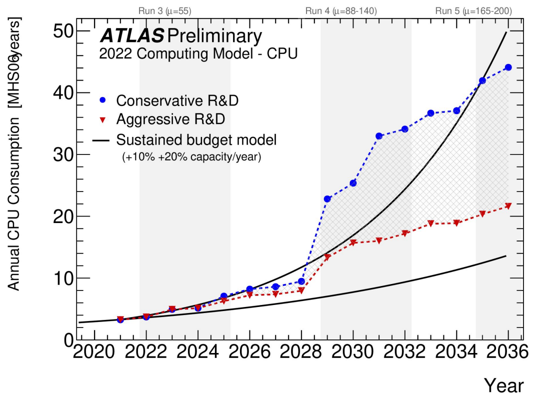
- A tracking geometry description which can be constructed manually or from TGeo and DD4hep input.
- Simple event data model.
- Implementations of common algorithms for track propagation and fitting.
- Implementations of basic seed finding algorithms.
- Implementations of common vertexing algorithms.

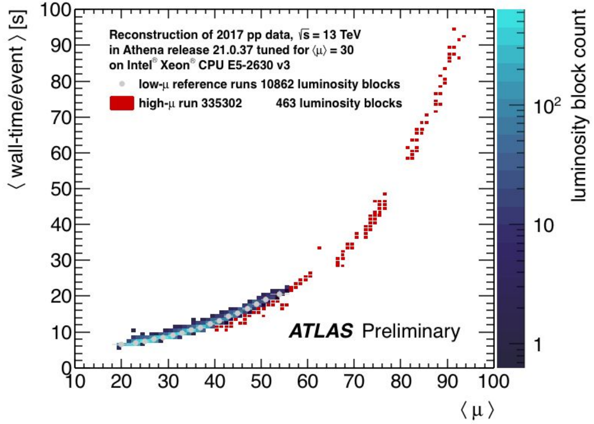
- Getting started
  - Quick start
  - Prerequisites
  - Building Acts
  - Building the documentation
  - Build options

- The Acts project aims to provide HEP / NP experiments with a toolbox for charged particle reconstruction
  - It allows experiment software to have its own specific event data types, which can be used with zero copy in the Acts components
- Its development is heavily influenced by the ATLAS Experiment at this stage, but it is truly meant to be a collaborative project across experiments
  - It is on track to be used by ATLAS in all of its track reconstruction by the LHC's Run-4

The HL-LHC Computing Challenge

- The LHC experiments will collect events of much higher complexity with much higher rate than ever before in the High Luminosity LHC era
- The computing requirements of “classical reconstruction algorithms” increase non-linearly with event complexity in many cases
  - Charged particle reconstruction being a dominant part of this
- In order to tackle this challenge new types of event reconstruction methods need to be tried, including using devices beyond classical CPUs

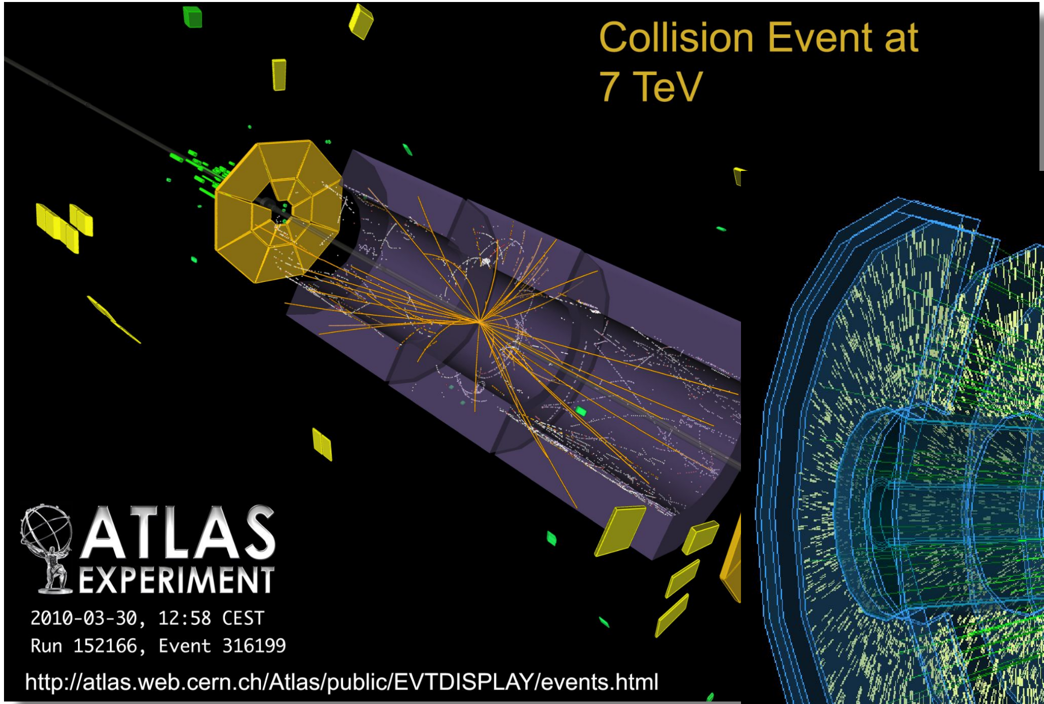


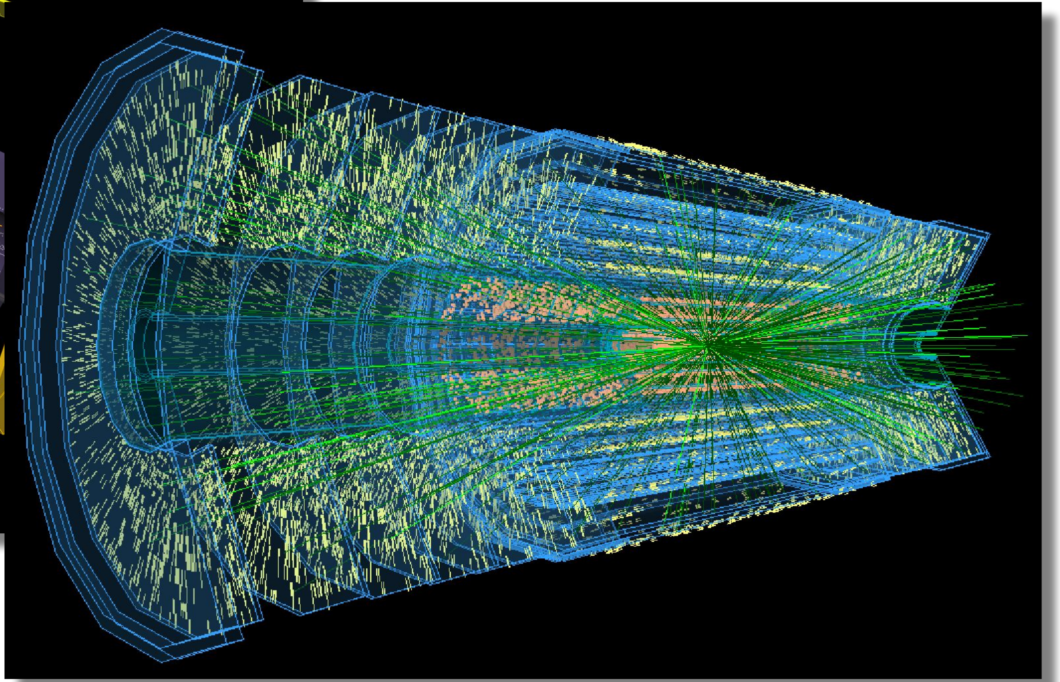


Track Reconstruction at High Pileup

Currently used algorithms were primarily designed for relatively low complexity collision events

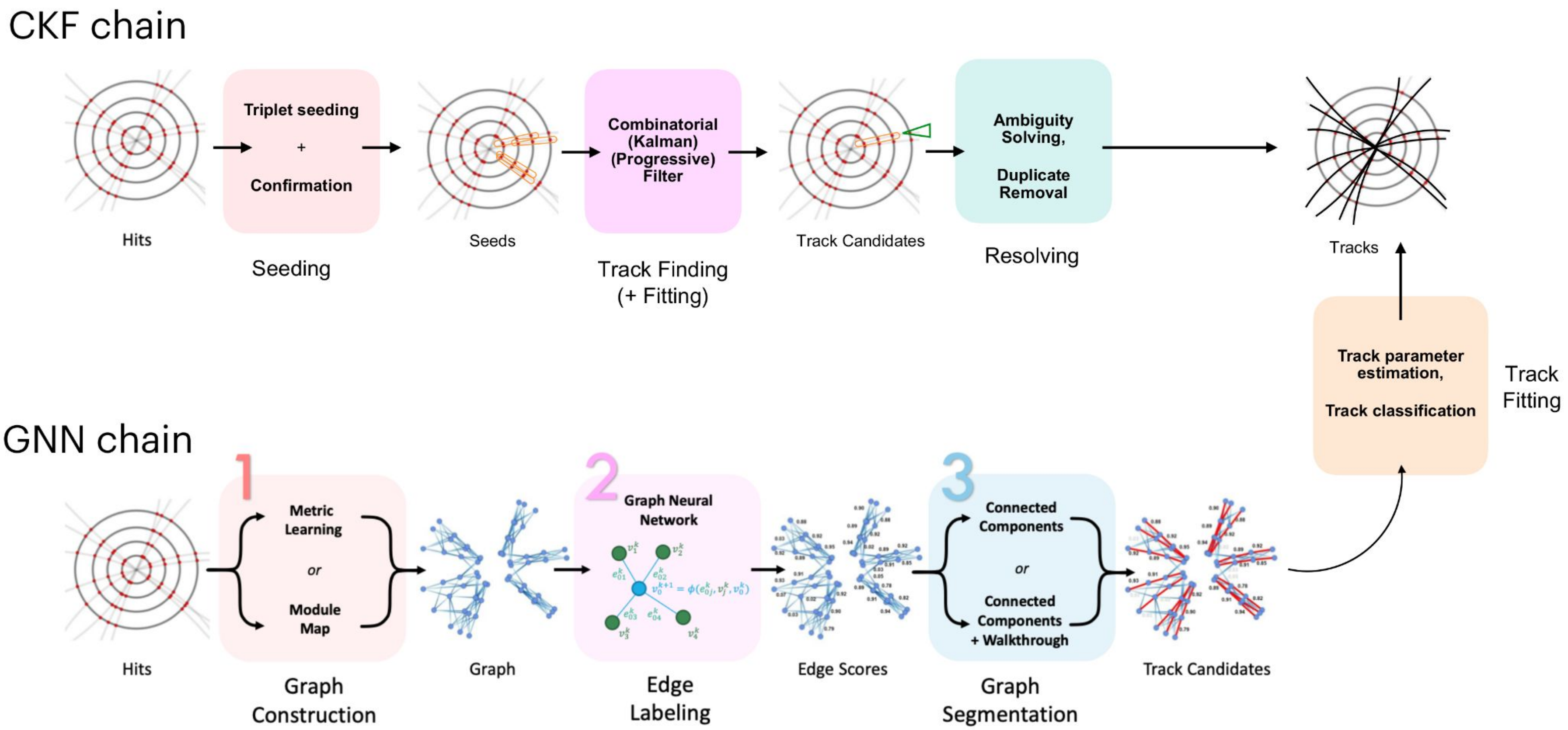
Track Reconstruction at High Pileup



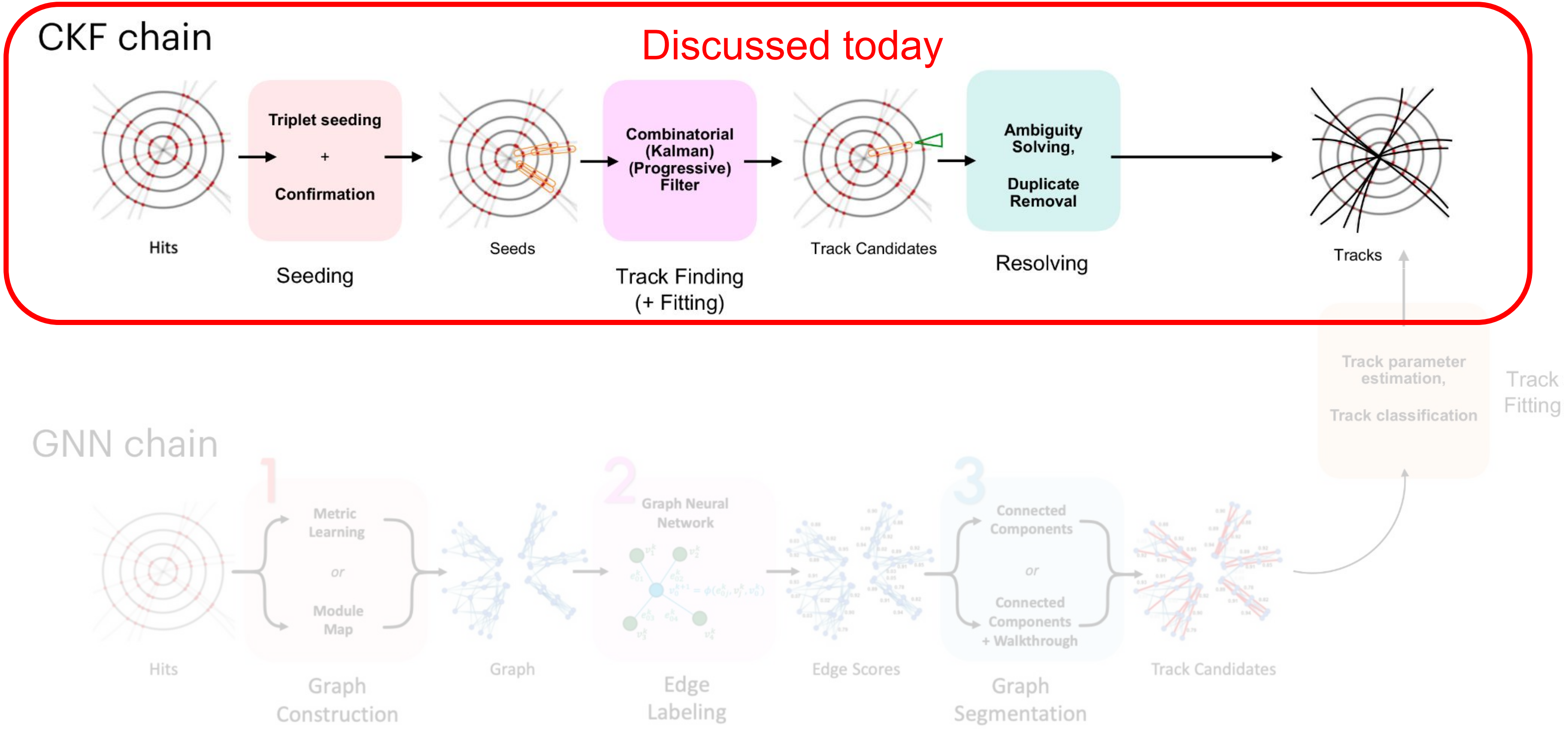


Combinatorics blow up at the expected >140 p-p interactions per collision (shown here for the ATLAS ITk geometry)

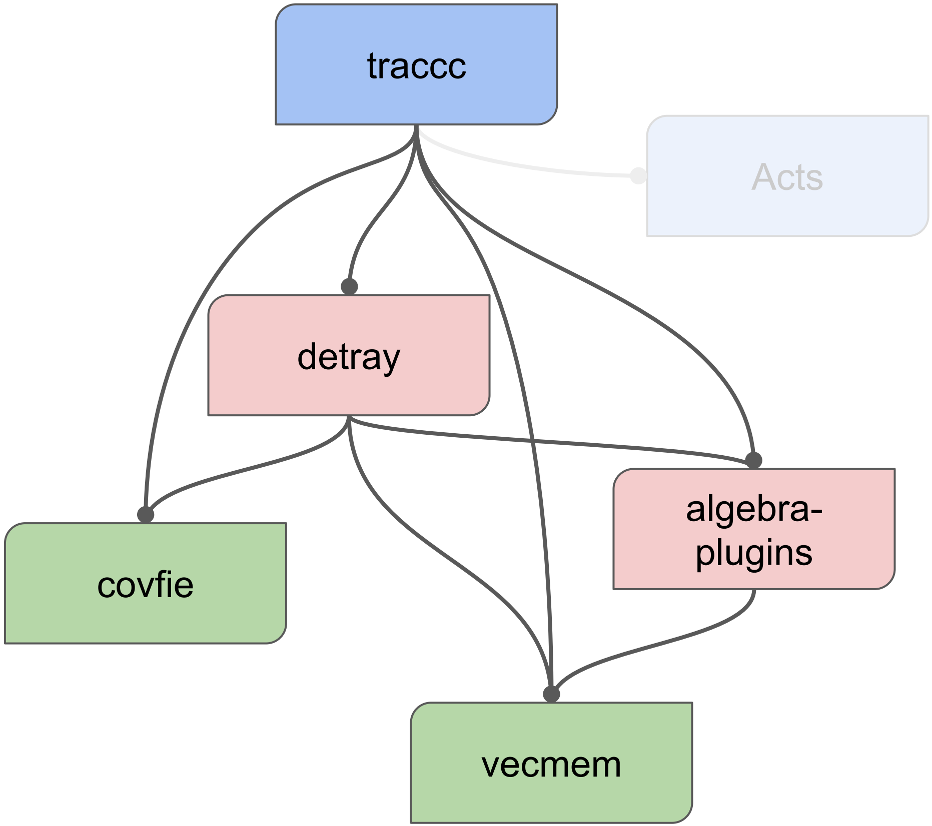
Track Reconstruction in Acts



Track Reconstruction in Acts



The Acts Parallelization R&D



- The development happens independently from the main Acts repository to make developments quicker
- Broken up into multiple, task specific projects
  - vecmem: Common memory management
  - covfie: Generic vector field handling
  - algebra-plugins: Small matrix linear algebra abstractions
  - detray: Tracking geometry handling in device code
  - traccc: The main repository of the R&D effort, holding most algorithmic code

Implementing Performance Portability

- While the CPU and GPU algorithms themselves are implemented separately, they do share a lot of functions
- These all need to be implemented **`inline`** to:
    - Allow the host and device compilers to generate code from them as they need it;
    - Allow the same function to be compiled into multiple object files (with different compilers/flags) during the build.

```
namespace traccc {

/// helper functions (for both cpu and gpu) to perform conformal transformation
///
/// @param x is the x value
/// @param y is the y value
/// @return is the conformal transformation result
inline TRACCC_HOST_DEVICE vector2 uv_transform(const scalar& x,
                                               const scalar& y) {
    vector2 uv;
    scalar denominator = x * x + y * y;
    uv[0] = x / denominator;
    uv[1] = y / denominator;
    return uv;
}

/// helper functions (for both cpu and gpu) to calculate bound track parameter
/// at the bottom spacepoint
///
/// @param seed is the input seed
/// @param bfield is the magnetic field
/// @param mass is the mass of particle
template <typename spacepoint_collection_t>
inline TRACCC_HOST_DEVICE bound_vector seed_to_bound_vector(
    const spacepoint_collection_t& sp_collection, const seed& seed,
    const vector3& bfield, const scalar mass) {

    bound_vector params;

    const auto& spB = sp_collection.at(seed.spB_link);
    const auto& spM = sp_collection.at(seed.spM_link);
    const auto& spT = sp_collection.at(seed.spT_link);

    darray<vector3, 3> sp_global_positions;
    sp_global_positions[0] = spB.global;
    sp_global_positions[1] = spM.global;
    sp_global_positions[2] = spT.global;
```

Implementing Performance Portability

```
/// CUDA kernel for running @c traccc::device::find_doublets
__global__ void find_doublets(
    seedfinder_config config, sp_grid_const_view sp_grid,
    device::doublet_counter_collection_types::const_view doublet_counter,
    device::device_doublet_collection_types::view mb_doublets,
    device::device_doublet_collection_types::view mt_doublets) {

    device::find_doublets(threadIdx.x + blockIdx.x * blockDim.x, config,
                          sp_grid, doublet_counter, mb_doublets, mt_doublets);
}
```

```
    // Find all of the spacepoint doublets.
    device::device_doublet_collection_types::view mb_view = doublet_buffer_mb;
    device::device_doublet_collection_types::view mt_view = doublet_buffer_mt;
    auto find_doublets_kernel =
        details::get_queue(m_queue).submit([&](::sycl::handler& h) {
            h.parallel_for<kernels::find_doublets>(
                doubletFindRange,
                [config = m_seedfinder_config, g2_view, doublet_counter_view,
                 mb_view, mt_view](::sycl::nd_item<1> item) {
                    device::find_doublets(item.get_global_linear_id(), config,
                                          g2_view, doublet_counter_view,
                                          mb_view, mt_view);
                });
        });
```

- Language specific kernel launches call on shared functions for the heavy lifting

```
namespace traccc::device {

/// Function finding all of the spacepoint doublets
///
/// Based on the information collected by @c traccc::device::count_doublets it
/// can fill collection with the specific doublet pairs that exist in the event.
///
/// @param[in] globalIndex       The index of the current thread
/// @param[in] config            Seedfinder configuration
/// @param[in] sp_view           The spacepoint grid to find doublets on
/// @param[in] dc_view           Collection with the number of doublets to find
/// @param[out] mb_doublets_view Collection of middle-bottom doublets
/// @param[out] mt_doublets_view Collection of middle-top doublets
///
TRACCC_HOST_DEVICE
inline void find_doublets(
    std::size_t globalIndex, const seedfinder_config& config,
    const sp_grid_const_view& sp_view,
    const doublet_counter_collection_types::const_view& dc_view,
    device_doublet_collection_types::view mb_doublets_view,
    device_doublet_collection_types::view mt_doublets_view);

}  // namespace traccc::device
```

Floating Point Precision

```
Running ./bin/traccc_seq_example_sycl tml_detector/trackml-detector.csv tml_full/ttbar_mu140/ 1
Running Seeding on device: NVIDIA GeForce RTX 3080
===>>> Event 0 <<<===
Number of spacepoints: 67442 (host), 67442 (device)
  Matching rate(s):
    - 97.9256% at 0.01% uncertainty
    - 99.3046% at 0.1% uncertainty
    - 99.3076% at 1% uncertainty
    - 99.3076% at 5% uncertainty
Number of seeds: 11514 (host), 11514 (device)
  Matching rate(s):
    - 86.5902% at 0.01% uncertainty
    - 98.7233% at 0.1% uncertainty
    - 99.2705% at 1% uncertainty
    - 99.3399% at 5% uncertainty
Number of track parameters: 11514 (host), 11514 (device)
  Matching rate(s):
    - 98.3933% at 0.01% uncertainty
    - 99.8002% at 0.1% uncertainty
    - 99.835% at 1% uncertainty
    - 99.835% at 5% uncertainty
```

FP32

```
Running ./bin/traccc_seq_example_sycl tml_detector/trackml-detector.csv tml_full/ttbar_mu140/ 1
Running Seeding on device: NVIDIA GeForce RTX 3080
===>>> Event 0 <<<===
Number of spacepoints: 67442 (host), 67442 (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: 11515 (host), 11515 (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: 11515 (host), 11515 (device)
  Matching rate(s):
    - 100% at 0.01% uncertainty
    - 100% at 0.1% uncertainty
    - 100% at 1% uncertainty
    - 100% at 5% uncertainty
```

FP64

- With the minimal vectorization that HEP code typically has, modern x86_64 CPUs have (virtually) the same performance for FP32 and FP64 operations
  - However accelerators usually do not
- On the other hand FP64 operations generally provide much better agreement between CPU and GPU algorithms 🤔
- All projects are set up to use a user-defined floating point type
  - For the moment higher level projects select the type on a project level, but it shall be possible to select the precision algorithm-by-algorithm in the future

Status of the Project

- Many programming techniques tried for GPUs during development
    - But only seriously using CUDA and SYCL for now
- The most complicated part, CKF, is under heavy development at the moment
    - It will be the final word on whether the project would fully succeed
- Track fitting works, but is not integrated into the “full chain” of algorithms at this point

Category	Algorithms	CPU	CUDA	SYCL	Futhark
Clusterization	CCL	✓	✓	✓	✓
	Measurement creation	✓	✓	✓	✓
	Spacepoint formation	✓	✓	✓	⚪
Track finding	Spacepoint binning	✓	✓	✓	⚪
	Seed finding	✓	✓	✓	⚪
	Track param estimation	✓	✓	✓	⚪
	Combinatorial KF	🟡	🟡	⚪	⚪
Track fitting	KF	✓	✓	✓	⚪

Measuring Performance

```
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);
    cell_module_collection_types::buffer modules_buffer(modules.size(),
                                                        *m_cached_device_mr);
    m_copy(vecmem::get_data(modules), modules_buffer);

    // Run the clusterization (asynchronously).
    const clusterizat
        m_clusterizat
    const track_param
        m_track_param


    // Get the final
    bound_track_param
    m_copy(track_para
    m_stream.synchron

    // Return the hos
    return result;
}
```

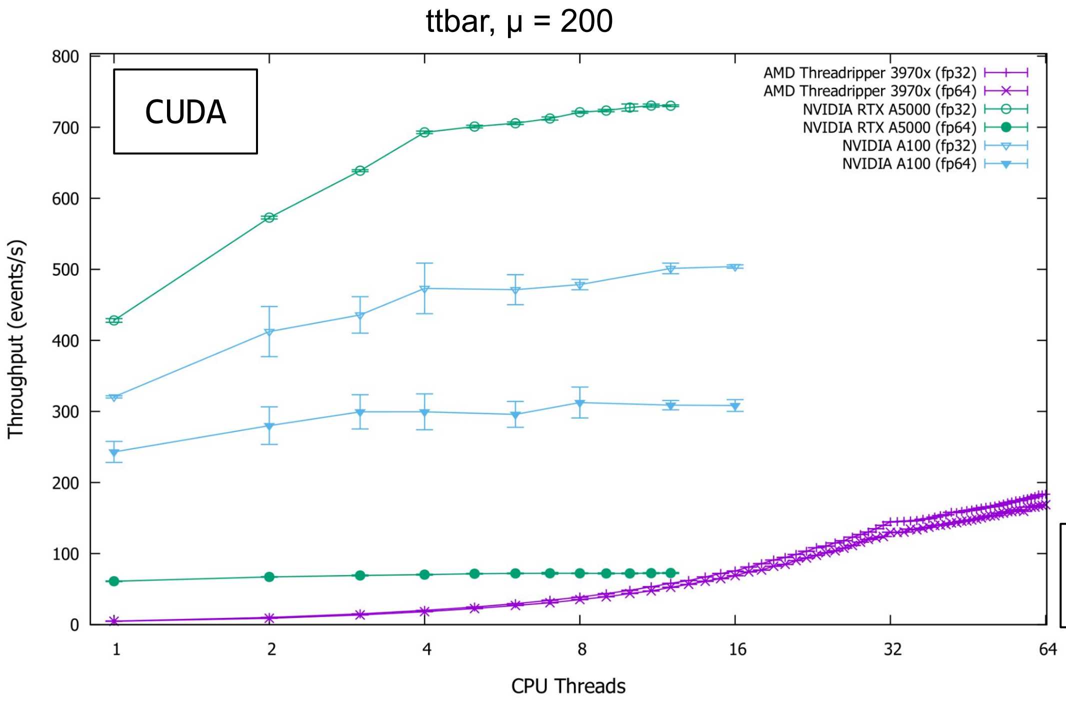
```
Multi-threaded CUDA GPU throughput tests

>>> Throughput options <<<
Input data format          : csv
Input directory            : tml_full/ttbar_mu140/
Detector geometry          : tml_detector/trackml-detector.csv
Digitization config        : tml_detector/default-geometric-config-generic.json
Target cells per partition : 1024
Loaded event(s)            : 10
Cold run event(s)          : 10
Processed event(s)         : 10000
Log_file                   :
>>> Multi-threading options <<<
CPU threads: 4

Using CUDA device: NVIDIA GeForce RTX 3080 [id: 0, bus: 1, device: 0]
Using CUDA device: NVIDIA GeForce RTX 3080 [id: 0, bus: 1, device: 0]
Using CUDA device: NVIDIA GeForce RTX 3080 [id: 0, bus: 1, device: 0]
Using CUDA device: NVIDIA GeForce RTX 3080 [id: 0, bus: 1, device: 0]
Using CUDA device: NVIDIA GeForce RTX 3080 [id: 0, bus: 1, device: 0]
Reconstructed track parameters: 104221093
Time totals:
                  File reading  8546 ms
            Warm-up processing  18 ms
              Event processing  10780 ms
Throughput:
            Warm-up processing  1.89721 ms/event, 527.091 events/s
              Event processing  1.07808 ms/event, 927.578 events/s
```

- On top of the many tests for the correctness of the output of the algorithms, we also build executables testing the throughput of the algorithms
  - Pre-load "cell data" for a set number of events into host memory;
  - Process the data into track parameters on possibly multiple CPU threads (using TBB);
  - Copy the results back into host memory if needed.

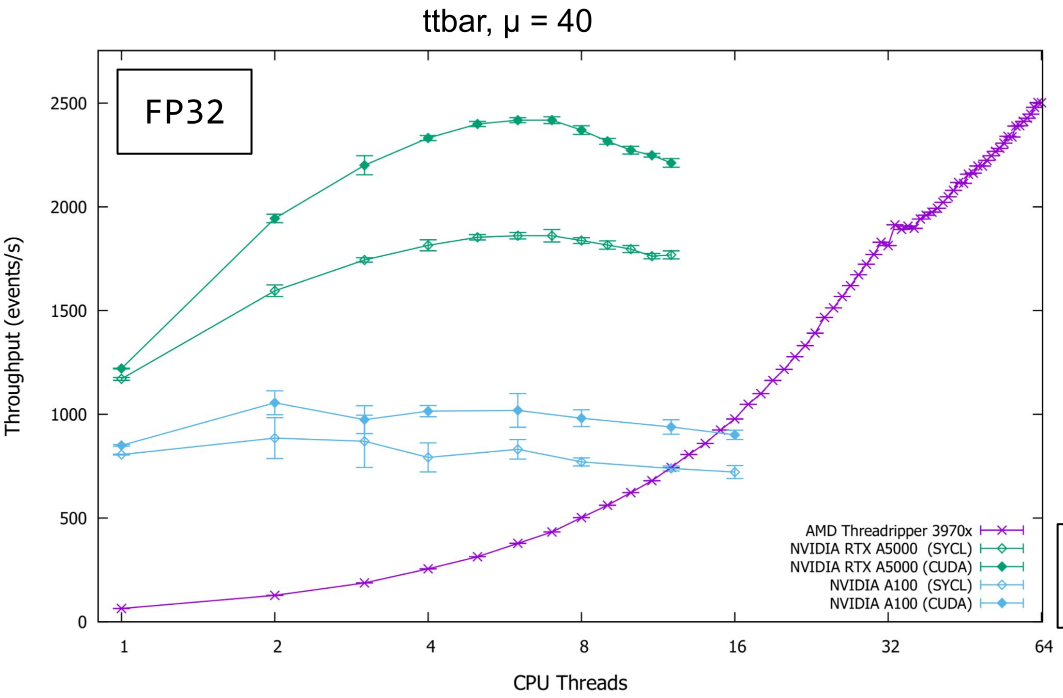
Measuring Performance (Precision)



Results partially generated on DAS-6.

With FP64 operations one needs (with our current code) a very high-end GPU to compete with a high-end CPU. With FP32 operations lower-end GPUs perform much better.

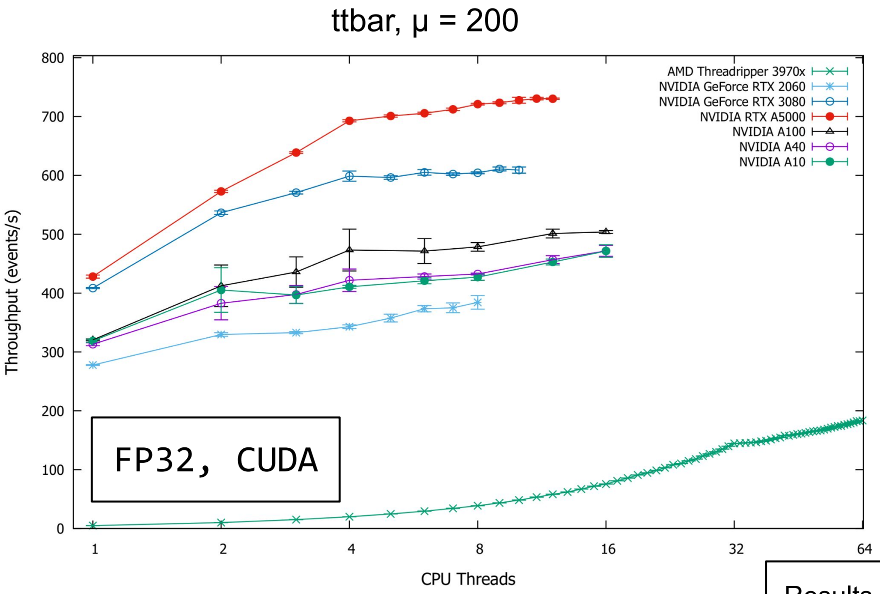
Measuring Performance (Language)

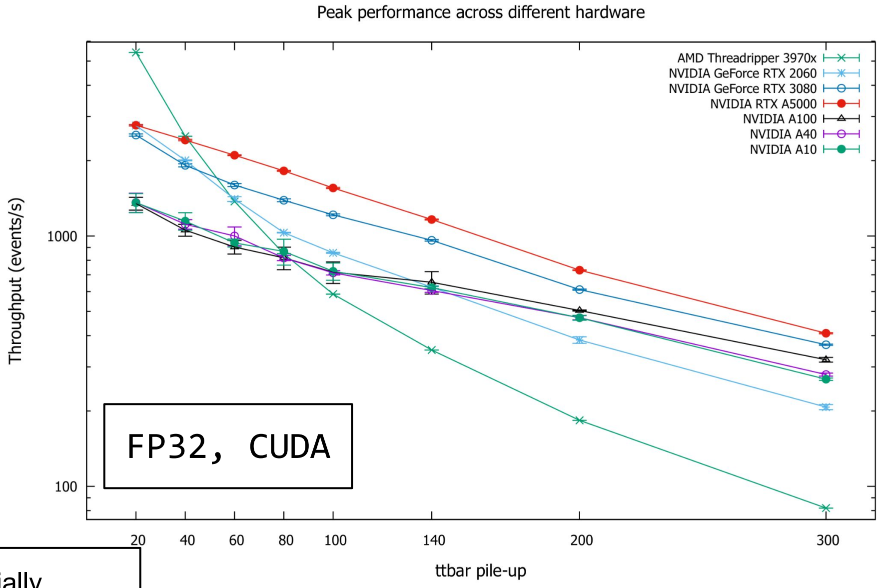


Results partially generated on [DAS-6](DAS-6).

With our current code CUDA performs better at low μ. (At high μ the difference is insignificant.)
Though we know about current inefficiencies in both implementations.

Measuring Performance (Summary)





Results partially generated on [DAS-6](DAS-6).

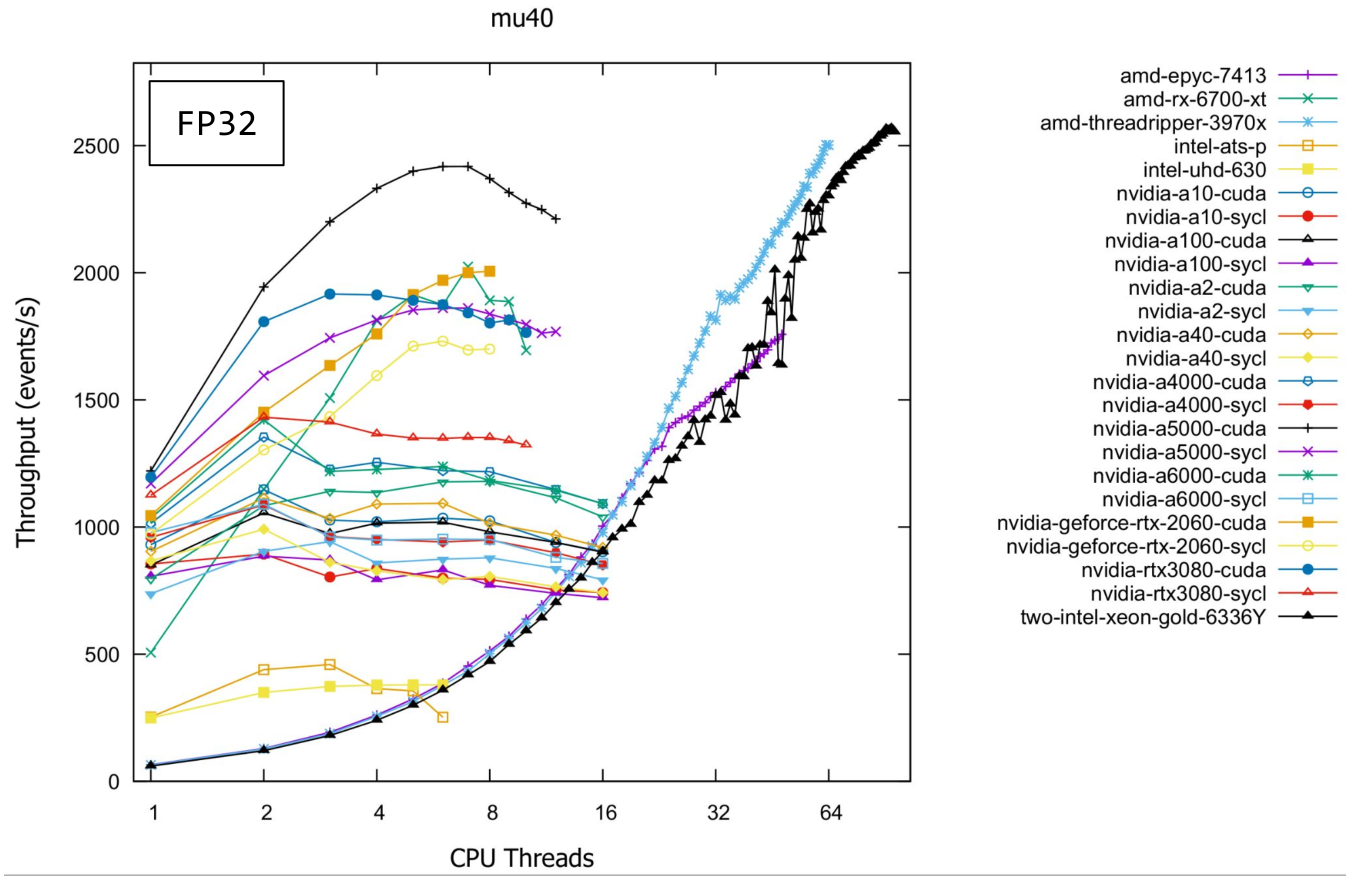
GPUs become competitive at high pile-up. Highest performance observed on NVIDIA® workstation GPUs so far.

Summary

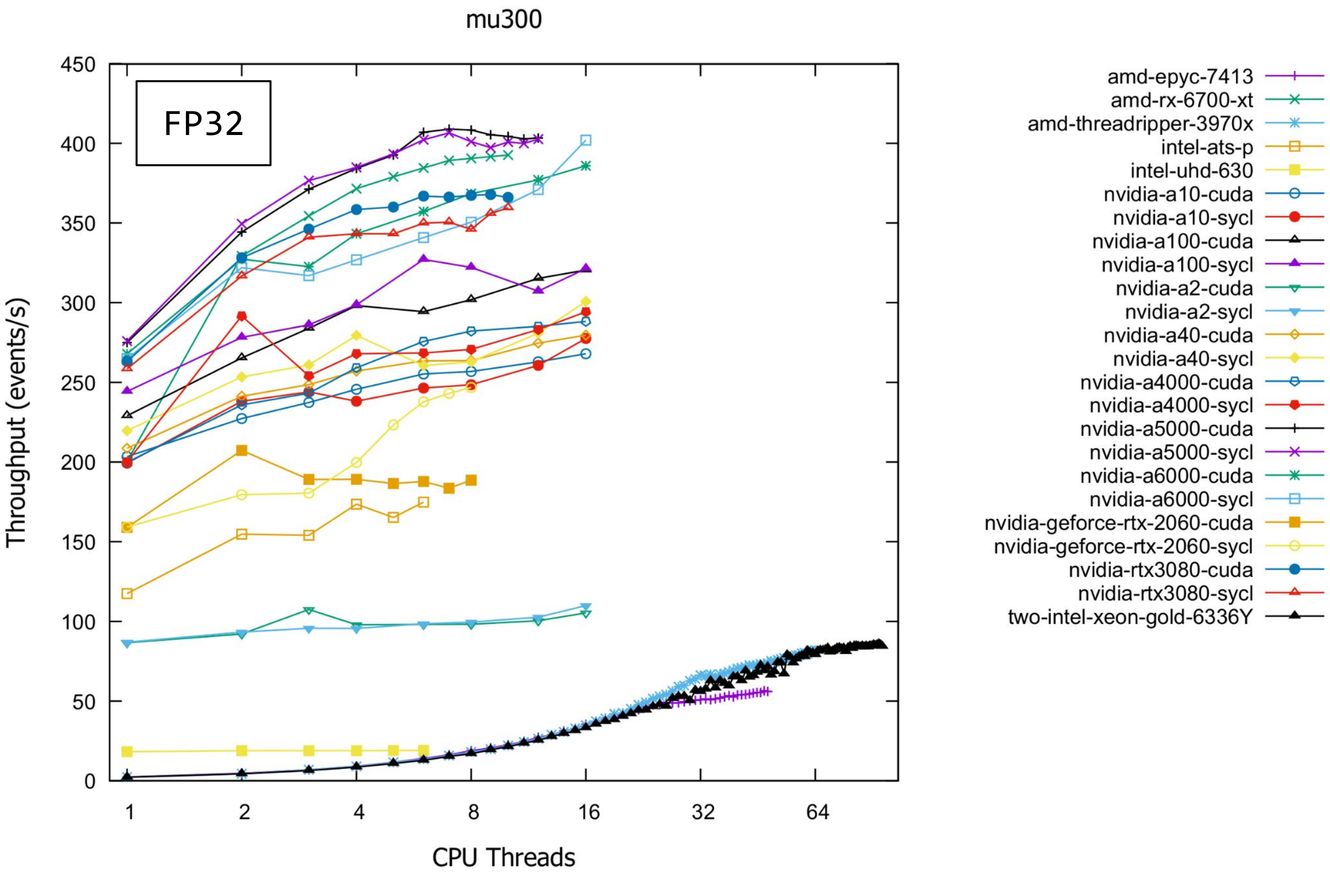
- The Acts Parallelization R&D is a significant effort, providing one of the demonstrators for the ATLAS HLT upgrade for the HL-LHC
  - Developments will start soon on exercising the code with the simulations / geometry of the ATLAS ITk
- The performance of the code is promising, providing a higher throughput with attainable GPUs than with the fastest CPUs that we could run tests on
  - What makes sense to use of course also very much depends on price and power usage, which we are not testing for / including in our results at the moment
- One significant development step still in the works: CKF

Backup

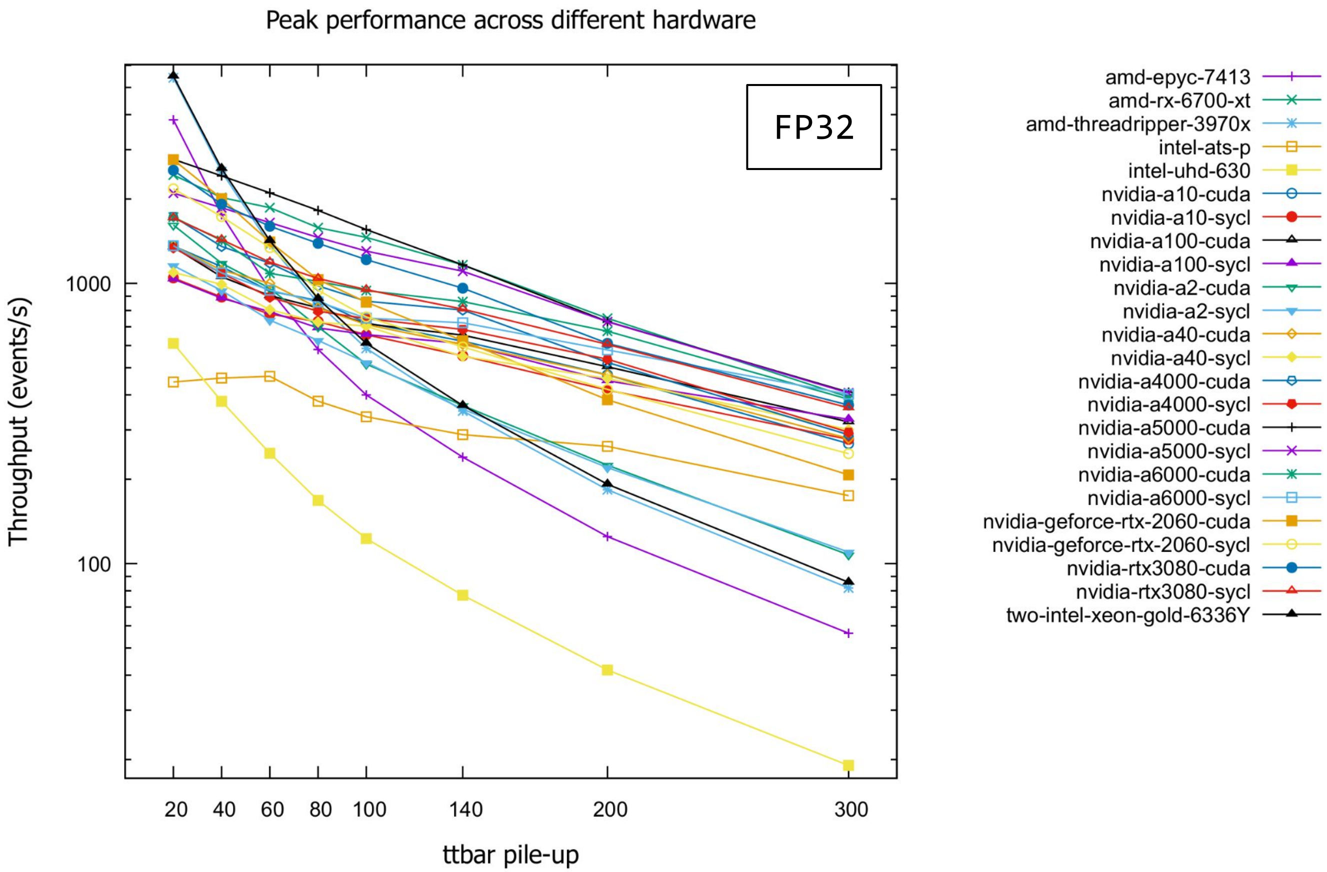
Performance



Performance



Performance



CERN

http://home.cern