Integrating oneAPI/SYCL in the ATLAS Software

Attila Krasznahorkay
on behalf of many people

Overview

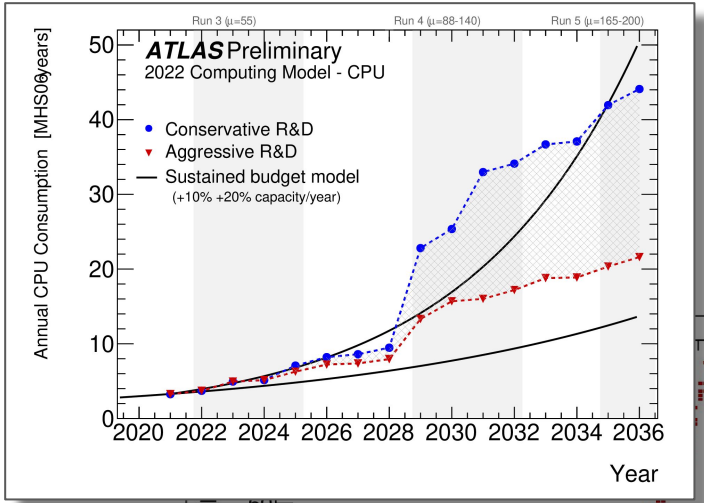
- Computing challenge(s) for ATLAS
- Accelerated code development in ATLAS
- Using SYCL in the Acts Parallelization R&D projects
- Latest performance results

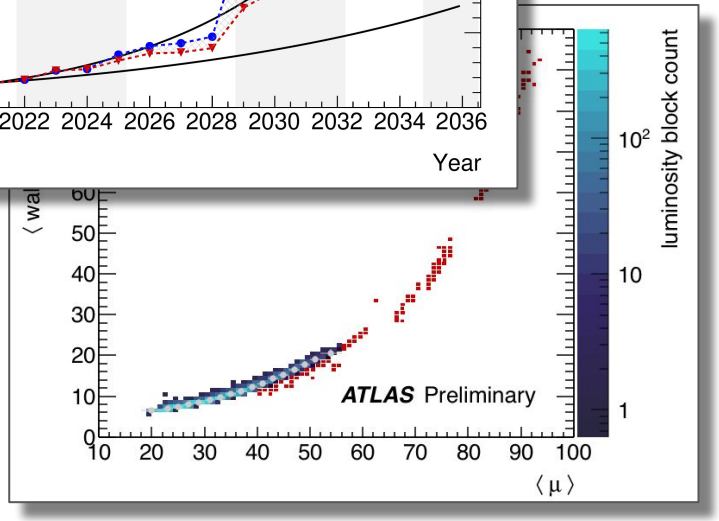
Overview

- Computing challenge(s) for ATLAS
- Accelerated code development in ATLAS
- Using SYCL in the Acts Parallelization R&D projects
- Latest performance results

Way too much material to cover in ~12 minutes. Will mostly focus on performance.

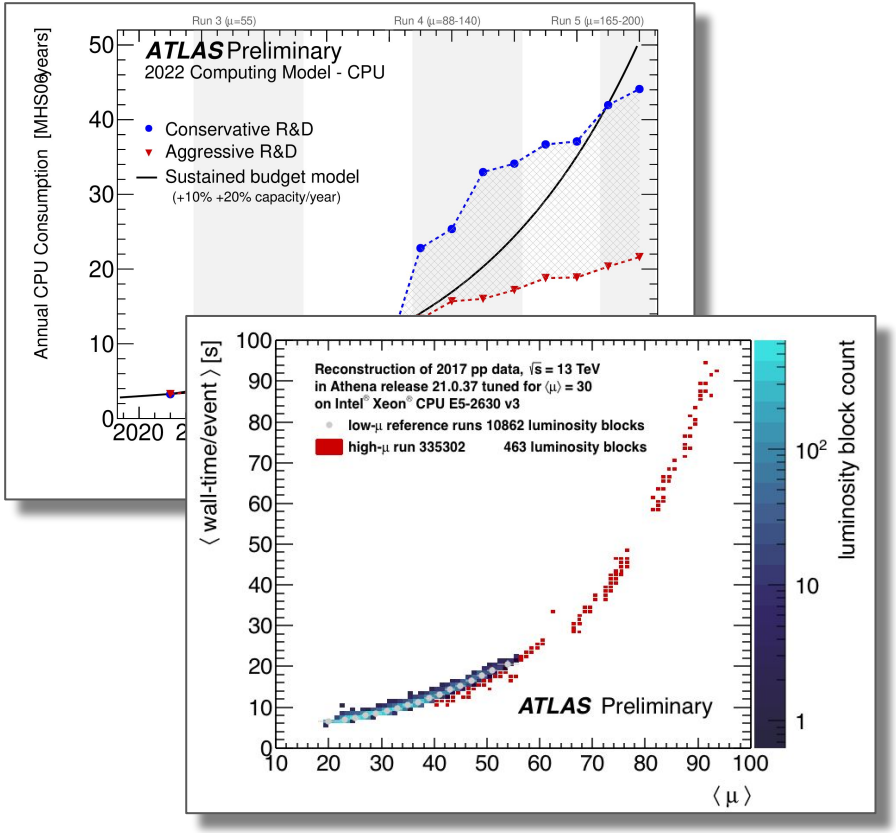
The HL-LHC Computing Challenge





- The problem is well known / widely advertised at this point
  - The “more complex” proton-proton collision events that we will record at the High-Luminosity LHC will require much more CPU power than we can afford
- This is in a large part due to the behaviour of charged particle tracking in ATLAS’s reconstruction
  - Though the CPU code did become a lot better since we made these original plots…

The HL-LHC Computing Challenge



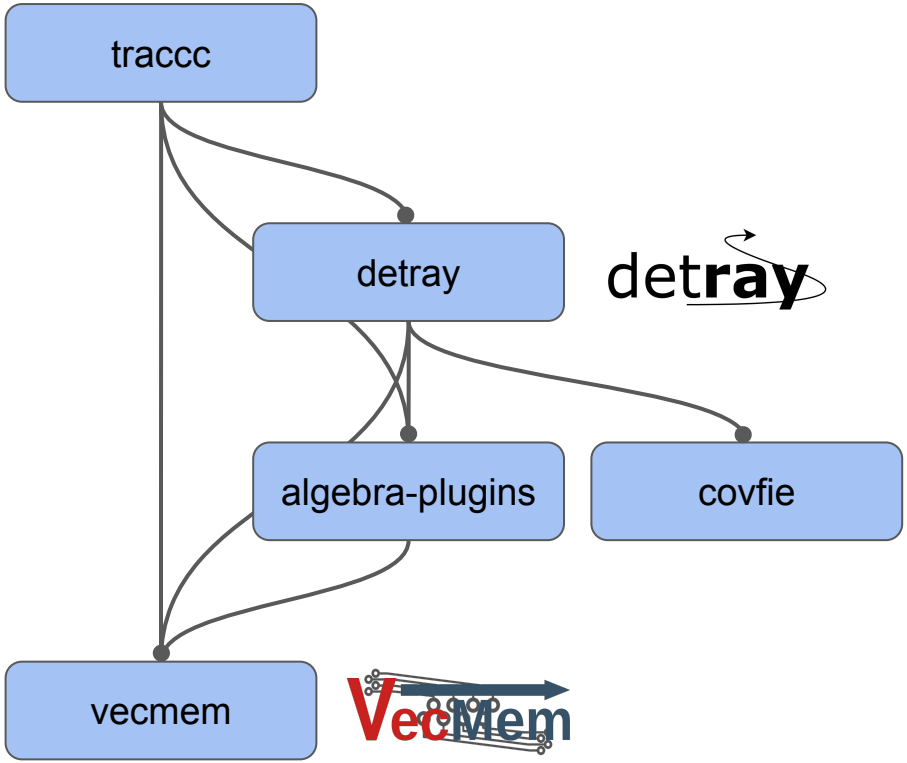
- The problem is well known / widely advertised at this point
  - The “more complex” proton-proton collision events that we will record at the High-Luminosity LHC will require much more CPU power than we can afford
- This is in a large part due to the behaviour of charged particle tracking in ATLAS’s reconstruction
  - Though the CPU code did become a lot better since we made these original plots…

Past and Ongoing Work

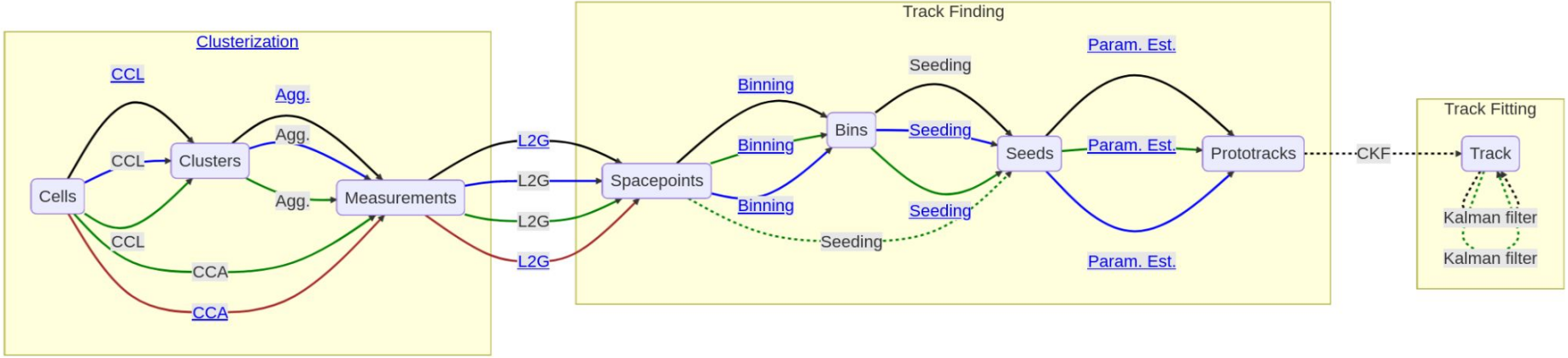
- Accelerator code development is happening in multiple areas in ATLAS, and since many years
  - I pointed to those last year already (https://indico.cern.ch/event/1100904/timetable/?view=standard#13-exploring-heterogeneous-arc)
- What I focus on today is just one of these areas
  - However one of the more significant ones…

The ACTS Parallelization R&D

- A number of projects were brought to life in the R&D effort
  - It seemed to be a good idea to create functionally distinct units independently. Even if eventually they'll likely end up in a unified repository.
- Algorithmic development is happening in traccc and detray
  - The rest are providing "non-algorithmic" helper code for the project



The traccc Algorithms



- Multiple algorithms were developed for for every step
  - Black: CPU/C++, Blue: SYCL, Green: CUDA, Brown: Futhark
- Sharing as much code between the GPU implementations (and even between the CPU and GPU implementations) as possible in the form of “common functions”

State of the Track Reconstruction Chain

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	🟡	🟡	⚊	⚪

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

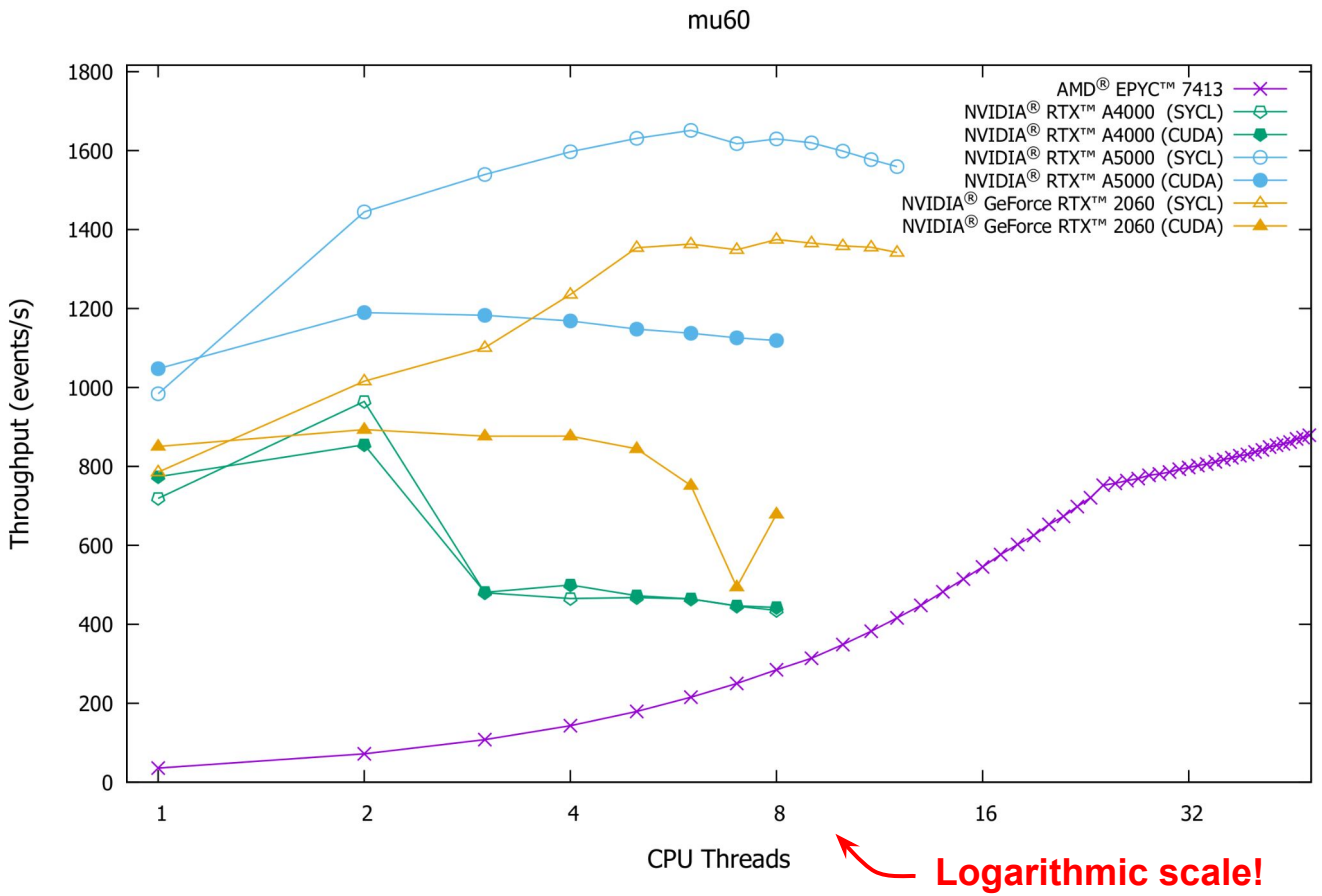
Also experimenting with Kokkos and Alpaka, but no significant algorithms with them yet.

Throughput Measurement Code

- Introduced multi-threaded executables that process events pre-loaded into host memory using TBB
  - One task per event
- For GPU devices multi-threading helps because:
  - Our algorithm chain still executes some operations on the CPU, even when a GPU is used
  - Our kernels often don't fully utilise GPUs, so multiple kernels can run in parallel

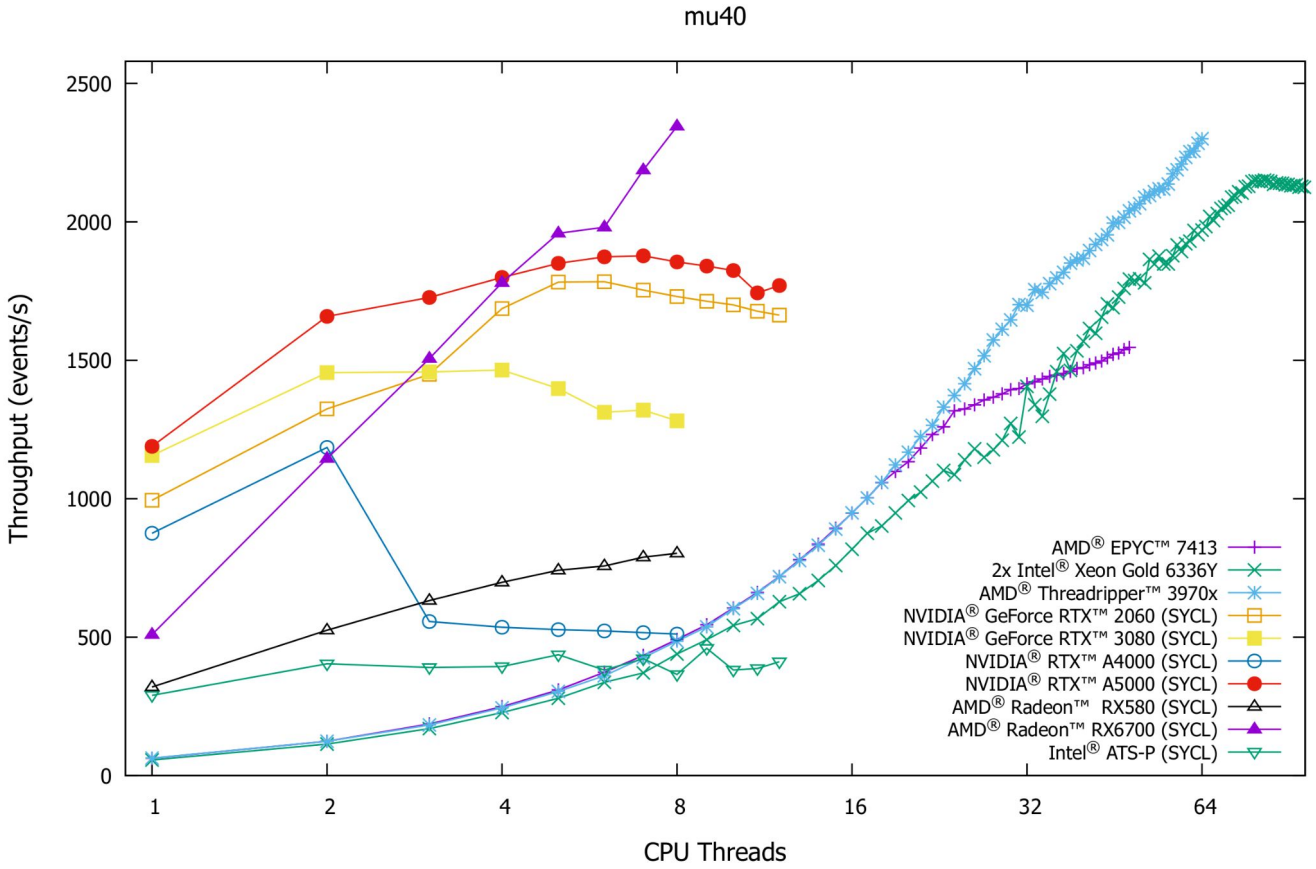
```
full_chain_algorithm::output_type full_chain_algorithm::operator()(
    const alt_cell_collection_types::host& cells,
    const cell_module_collection_types::host& modules) const {

    // Create device copy of input collections
    alt_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);

    // Execute the algorithms.
    const clusterization_algorithm::output_type spacepoints =
        m_clusterization(cells_buffer, modules_buffer);
    const track_params_estimation::output_type track_params =
        m_track_parameter_estimation(spacepoints.first,
                                     m_seeding(spacepoints.first));

    // Get the final data back to the host.
    bound_track_parameters_collection_types::host result;
    (*m_copy)(track_params, result);

    // Return the host container.
    return result;
}
```

CUDA and SYCL



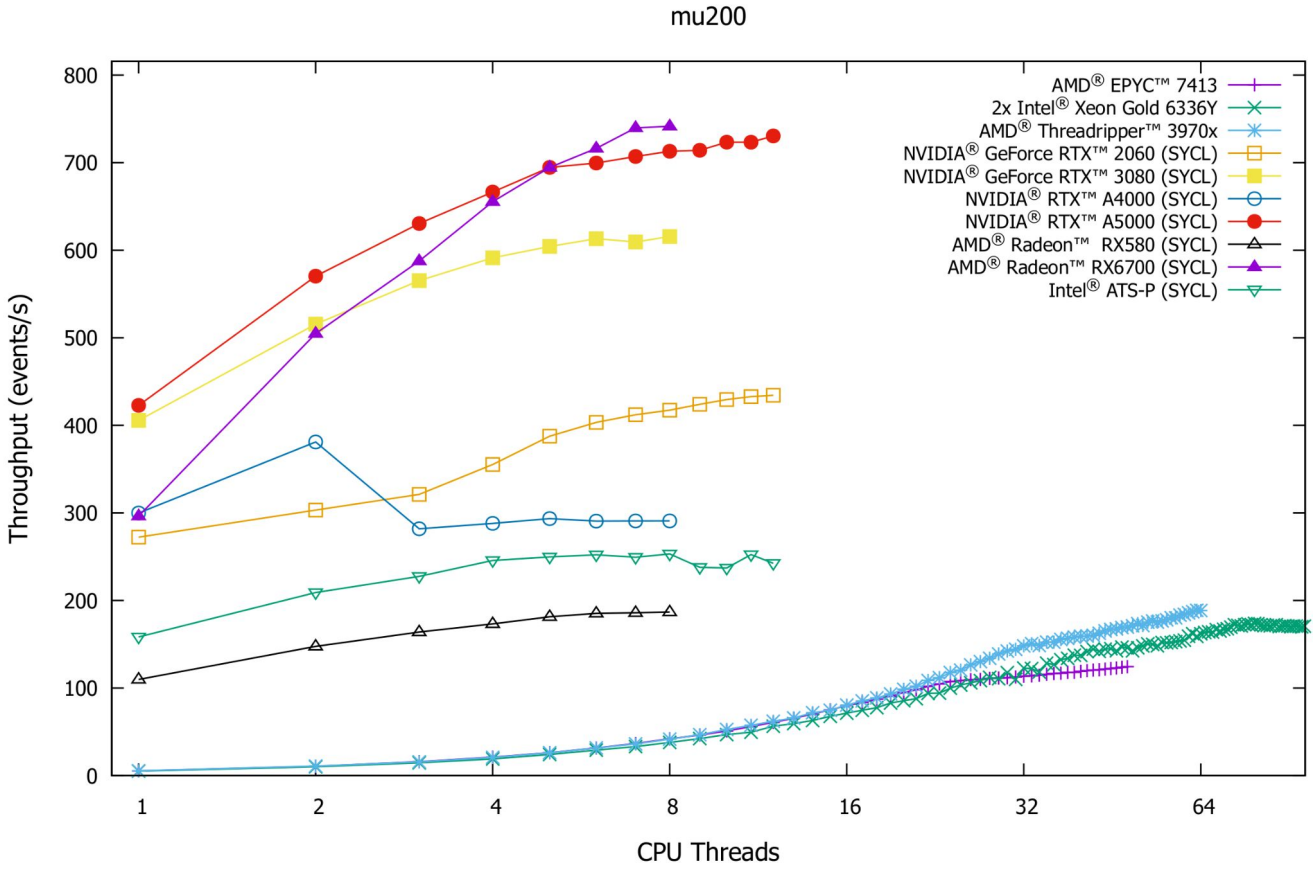
- Both our CUDA and SYCL code is sub-optimal still (not taking full advantage of parallel / asynchronous execution)
- Current SYCL code performs better with MT, so only showing those results in the rest of the plots

Throughput Measurements



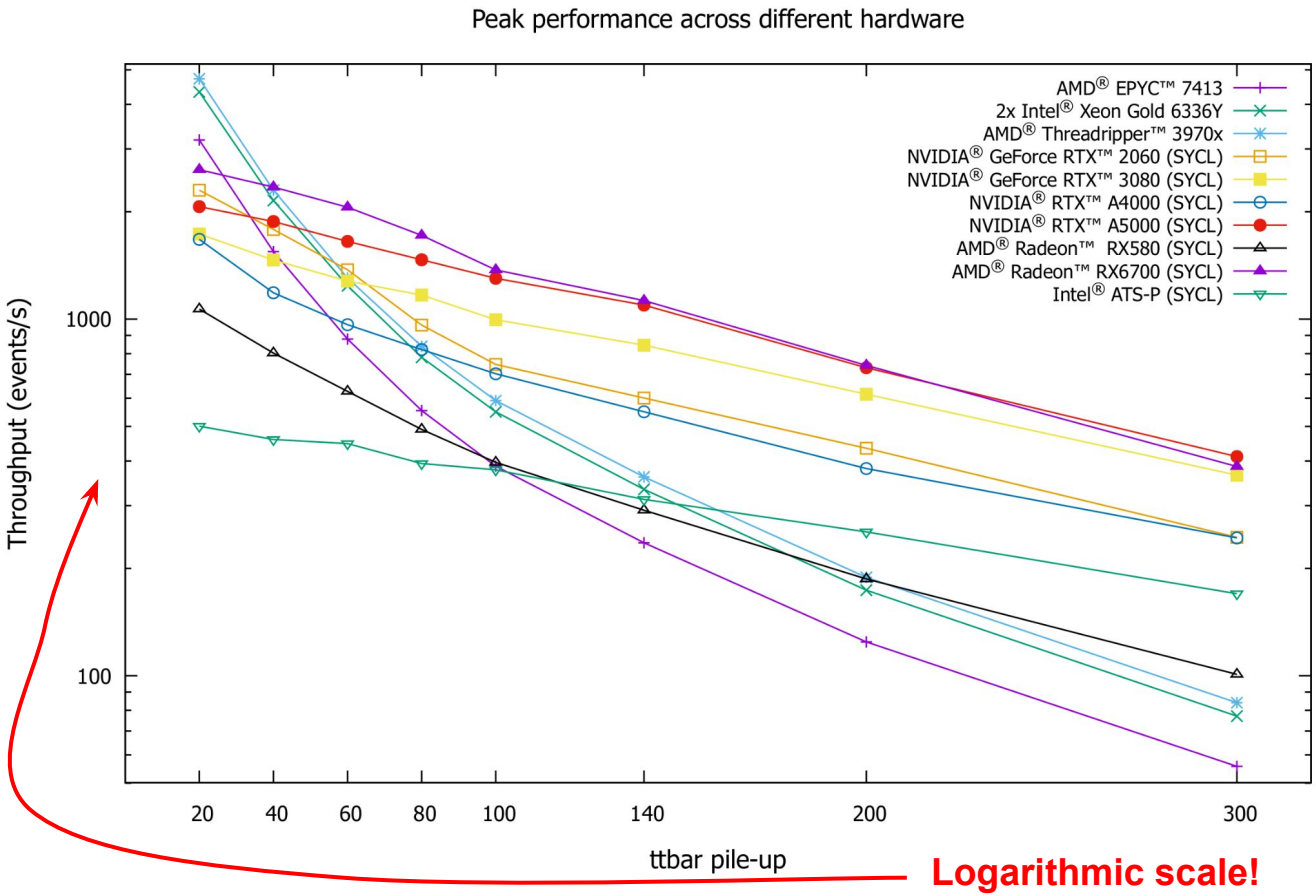
- This is “low pile-up”
- Only showing SYCL results
- High-end CPUs still beat GPUs
- The RX6700 card is doing something weird (take with huge grain of salt!)

Throughput Measurements



- This is “high pile-up”
- GPU relative performance close to constant
- CPUs losing to even low-end GPUs

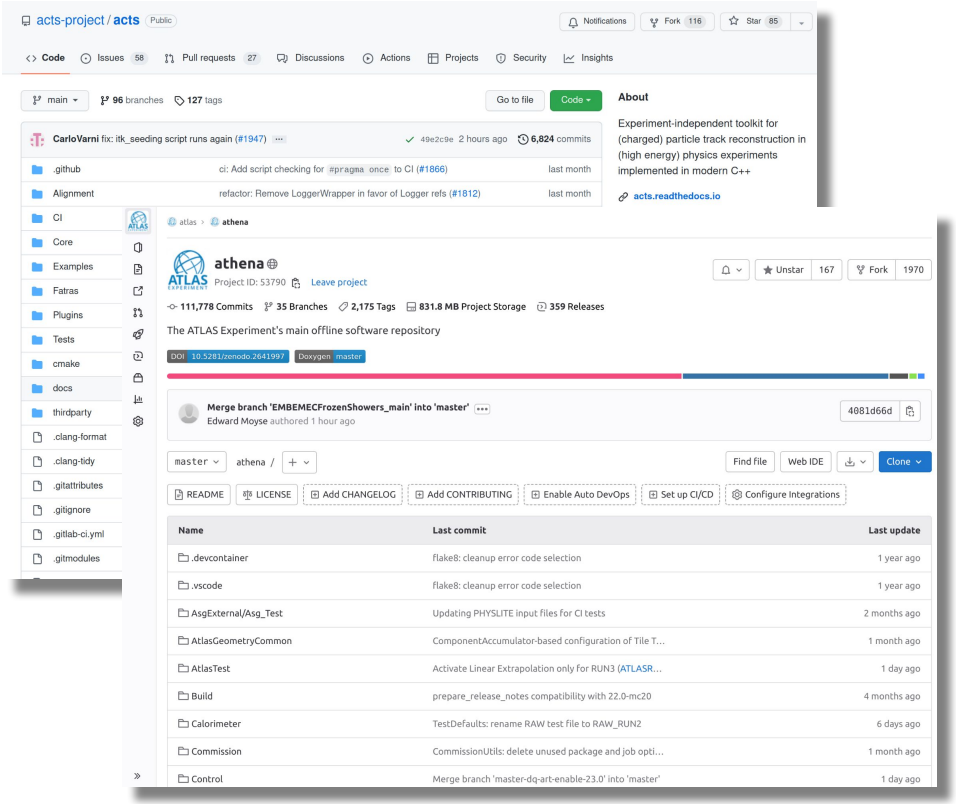
Throughput Measurements



- Putting it all together…
- GPUs (all) beat CPUs in throughput at HL-LHC event complexities
- Will use such results in our final decisions about ATLAS development strategies…

Integration Into Athena

- All of this only lives in our R&D projects at the moment
- The plan is to start moving the code into the main Acts repository this year
  - And then to pick up the code in Athena from Acts
- Still some strategic questions will need to be decided for this about the way Acts and Athena would handle “event data”
  - No blocker issues on the horizon however



Summary

- Don’t take any of these results as the end-all-be-all!
  - We’re in the process of making our performance testing code even faster with both CUDA and SYCL
- We seem to be well on track to produce maintainable code for track reconstruction that would minimise the duplication between CPU and GPU codebases
- Performance numbers are very encouraging at the moment
  - But do take them, especially the AMD ones, with some amount of salt…

CERN

http://home.cern