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

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

CKF chain

Hits → Triplet seeding + Confirmation → Seeds → Combinatorial (Kalman) (Progressive) Filter → Track Candidates → Ambiguity Solving, Duplicate Removal → Tracks

GNN chain

1. Metric Learning or Module Map → Hits
2. Graph Neural Network → Edge Labeling → Edge Scores → Connected Components or Connected Components + Walkthrough → Graph Segmentation → Track Candidates

Track parameter estimation, Track classification

Track Fitting
Track Reconstruction in Acts

CKF chain

Seeding

Triplet seeding + Confirmation

Combinatorial (Kalman) (Progressive) Filter

Ambiguity Solving, Duplicate Removal

Tracks

GNN chain

Graph Construction

Metric Learning or Module Map

Graph Neural Network

Connected Components or Connected Components + Walkthrough

Track Candidates

Discussed today

Hits

Seeds

Track Candidates

Resolving

Tracks

Track parameter estimation, Track classification

Track Fitting
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
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.
Implementing Performance Portability

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

```
62 // CUDA kernel for running &c tracc::device::find_doublets
63 __global__ void find_doublets(
64     seedfinder_config config, sp_grid_const_view sp_grid,
65     device::doublet_counter_collection_types::const_view doublet_counter,
66     device::device_doublet_collection_types::view mb_doublets,
67     device::device_doublet_collection_types::view mt_doublets) {
68     device::find_doublets(threadIdx.x + blockIdx.x * blockDim.x, config,
69         sp_grid, doublet_counter, mb_doublets, mt_doublets);
70 }

150 // Find all of the spacepoint doublets.
151 device::device_doublet_collection_types::view mb_view = doublet_buffer_mb;
152 device::device_doublet_collection_types::view mt_view = doublet_buffer_mt;
153 auto find_doublets_kernel =
154     details::get_queue(m_queue).submit([&](sycl::handler& h) {
155         h.parallel_for<class find_doublets>(
156             doubletFindRange,
157             [config = m_seedfinder_config, g2_view, doublet_counter_view,
158             mb_view, mt_view]::sycl::nd_item<1> item) {
159             device::find_doublets(item.get_global_linear_id(), config,
160                 g2_view, doublet_counter_view,
161                 mb_view, mt_view);
162         });
163     });
```

```
Floating Point Precision

- 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

<table>
<thead>
<tr>
<th>Category</th>
<th>Algorithms</th>
<th>CPU</th>
<th>CUDA</th>
<th>SYCL</th>
<th>Futhark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clusterization</td>
<td>CCL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Measurement creation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spacepoint formation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Track finding</td>
<td>Spacepoint binning</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Seed finding</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Track param estimation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combinatorial KF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Track fitting</td>
<td>KF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Measuring Performance

- 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.
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.

Results partially generated on DAS-6.
With our current code CUDA performs better at low $\mu$. (At high $\mu$ the difference is insignificant.) Though we know about current inefficiencies in both implementations.
GPUs become competitive at high pile-up. Highest performance observed on NVIDIA® workstation GPUs so far.
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.
Performance

![Graph showing performance comparison of different hardware configurations. The x-axis represents CPU threads, and the y-axis represents throughput (events/s). Different hardware models are plotted in various lines, indicating their performance under different thread counts.](image)
Performance

mu300

Throughput (events/s)

CPU Threads

FP32

amd-epyc-7413
amd-rx-6700-xt
amd-threadripper-3970x
intel-ats-p
intel-uhd-630
nvidia-a10-cuda
nvidia-a10-sycl
nvidia-a100-cuda
nvidia-a100-sycl
nvidia-a2-cuda
nvidia-a2-sycl
nvidia-a40-cuda
nvidia-a40-sycl
nvidia-a4000-cuda
nvidia-a4000-sycl
nvidia-a5000-cuda
nvidia-a5000-sycl
nvidia-a6000-cuda
nvidia-a6000-sycl
nvidia-geforce-rtx-2060-cuda
nvidia-geforce-rtx-2060-sycl
nvidia-rtx3080-cuda
nvidia-rtx3080-sycl
two-intel-xeon-gold-6336Y
Performance

Peak performance across different hardware

Throughput (events/s)

ttbar pile-up

FP32

amd-epyc-7413
amd-rx-6700-xt
amd-threadripper-3970x
intel-ats-p
intel-uhd-630
nvidia-a10-cuda
nvidia-a10-sycl
nvidia-a100-cuda
nvidia-a100-sycl
nvidia-a2-cuda
nvidia-a2-sycl
nvidia-a40-cuda
nvidia-a40-sycl
nvidia-a4000-cuda
nvidia-a4000-sycl
nvidia-a5000-cuda
nvidia-a5000-sycl
nvidia-a5000-cuda
nvidia-a6000-cuda
nvidia-a6000-sycl
nvidia-geforce-rtx-2060-cuda
nvidia-geforce-rtx-2060-sycl
nvidia-rtx3080-cuda
nvidia-rtx3080-sycl
two-intel-xeon-gold-6336Y