A vendor-unlocked GPU reconstruction for the ALICE Inner Tracking System

Matteo Concas, for the ALICE collaboration
ALICE data processing for Run 3

- **Trigger-less acquisition: continuous readout**
  - The stream of data is split into O(10ms) timeframes
  - $L_{\text{int}} > 10 \text{ nb}^{-1}$ of PbPb data at 50kHz: 50x more than Run 2

- **Online reconstruction and calibration for data compression\(^1\)**
  - Synchronous: TPC full reconstruction and calibration
  - Asynchronous: all compressed data are reconstructed
  - Single computing framework for online-offline computing: O\(^2\)

- **Operate part of the reconstruction on GPUs is mandatory**
  - Minimise the cost/performance ratio for online farm
  - 250x Event Processing Nodes (EPNs), 8x AMD MI50 GPUs

- **Efficient utilisation of available computing resources is desired**
  - Other detectors' reconstruction demands more resources
  - A larger fraction of GPUs available during the asynchronous phase

---

\(^1\) “The O2 software framework and GPU usage in ALICE online and offline reconstruction in Run 3”

Matteo Concas, for the ALICE collaboration - CHEP 2023
ITS reconstruction in Run 3

- A new upgraded Inner Tracking System
  - A cylindrical silicon detector, is the innermost detector of the apparatus
  - Thin layers, 12.5 billion pixels and 10 m² of sensitive area
  - Provide spatial information in the form of clusters of fired pixels

- Continuous readout, continuous track reconstruction
  - Digital signals from pixels are clusterised and compressed
  - Timeframes are divided into Readout Frames (ROF): ~4μs
  - Collision data can split across multiple adjacent rofs

- New standalone vertexing and tracking algorithm
  - Single implementation steered via configuration (sync/async, collision system, ...)
  - During synchronous reconstruction focuses on primaries (7 clusters-long tracks)
  - During asynchronous reconstruction is sensitive to secondaries and tracks lower $p_T$
  - In- and cross-bunch pile-up ready
ITS vertexing and tracking

- Primary vertex seeding
  - Look for correlations between hits in the three innermost ITS layers
  - Combinatorial matching followed by linear extrapolations of tracklets
  - Unsupervised clustering to find the collision point(s)

- Track finding and track fitting
  - Uses vertex position to reduce the combinatorics in matching the hits
  - Connect segments of tracks, the cells, into a tree of candidates: roads
  - Kalman filter to fit tracks from candidates and apply quality cuts

- The algorithm is decomposable into multiple parallelisable steps
  - ALICE Data Processing Layer manages parallel Timeframe scheduling
  - Each ROF can be processed independently(*)
  - In-frame combinatorics can be processed simultaneously
  - Current CPU implementation can profit from multi-threaded sections

(*) Information from adjacent ROFs can be used to recover from information splitting
A parallel implementation using GPUs

- **Accelerate** the processing using massively parallel architectures
  - Promising porting of some routines based on CUDA and OpenCL in the past

- **Today: offload the whole** vertexing and tracking on GPUs
  - Release the corresponding CPU cycles, improving resource usage efficiency
  - Integrate it into the broader reconstruction GPU chain by extending its coverage

- **First phase: load and operate standalone GPU tracking for ITS**
  - Mainstream reconstruction framework provides the interface for GPU lib loading
  - The ITS GPU library fully manages graphics card resource
  - Easy-to-contribute: plain C++ and CUDA code, focus on routines development
  - Supports CUDA and HIP with a single code base, no compatibility layer

- **Second phase: build a GPU reconstruction chain including ITS**
  - Centrally manage GPU memory and kernel scheduling for deeper integration
  - Easier to later add additional steps like the ITS-TPC matching

---

**ITS tracking workflow**

- **GPU reconstruction instance**
  - Reco Chains
  - Reco chain ...
  - ITS reco chain ...

- **ITS CPU library**
- **ITS CUDA library**
- **ITS HIP library**

---

Matteo Concas, for the ALICE collaboration - CHEP 2023
Cornerstones of the GPU implementation

- **Resource usage flexibility via configuration**
  - The amount of usable memory is a parameter that is passed to the algorithm
  - All required chunk sizes are set as a fraction of the total available memory

- **Multi-threaded streams process bunches of ROFs in parallel**
  - Each POSIX thread manages a stream, and the full tracking is independent
  - Optimise throughput by hiding memory loads behind kernel executions
  - Average timeframe size changes under different conditions: we cope with that!

- **Use case extensibility via a generic N-layers implementation**
  - All the routines in the vertexing and tracking are "local": no need to know N-layers
  - TrackerGPU<NLayers> offers native support for future use cases (ITS3/ALICE3)
Cross-platform on-the-fly code generation

- The O2 compilation via **CMake**, provides
  - **Platform autodetection** and production of corresponding target libraries
  - **Custom commands** setting dependencies between targets

- HIP code is generated in place from CUDA sources
  - **Build source of targets** parsing CUDA files and generating HIP versions
  - Currently based on **hipify-perl**: is run on all .cu files to produce HIP
  - **Develop and maintain** a single code base in place of two
  - **No need to add** a portability layer

- Headers files are shared across both the compilations
  - **Negligible boilerplate** (<0.1% LoCs) to cope with some architectural differences

```c
// CUDA code
cudamalloc(&A_d, Nbytes);
cudamalloc(&C_d, Nbytes);
cudamemcpy(A_d, A_h, Nbytes, cudamemcpyHostToDevice);
vector_square <<<512, 256>>>(C_d, A_d, N);
cudamemcpy(C_h, C_d, Nbytes, cudamemcpyDeviceToHost);

// HIP code, translated
hipmalloc(&A_d, Nbytes);
hipmalloc(&C_d, Nbytes);
hipmemcpy(A_d, A_h, Nbytes, hipmemcpyHostToDevice);
hipLaunchKernelGGL(vector_square, 512, 256, 0, 0, C_d, A_d, N);
hipmemcpy(C_h, C_d, Nbytes, hipmemcpyDeviceToHost);
```
State of the development and testing

- Names of the steps are the **main phases of the algorithms**
  - GPU-ported implementations are usually more complex due to data organisation
  - Comparison with CPU is made when steps in both versions produce the same output

- The **vertexing** is fully operative in its GPU implementation
  - Timeframe chunking is natively supported, as no extra-ROF information is required

- The porting of **tracking** is being finalised
  - The last fully ported step is the neighbour finder for cells
  - Road finder is under development: size and number of found roads are not static
  - Track fitting had a POC, which requires an in-depth review

- Tested on **both Nvidia and AMD cards**
  - First setup: workstation with AMD Ryzen™ 9 7950X CPU and Nvidia™ TITAN Xp
  - Second setup: EPN node with 2x AMD EPYC™ 7452 and AMD Instinct™ MI50

<table>
<thead>
<tr>
<th>Vertexer</th>
<th>Clock (GHz)</th>
<th>RAM (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracklet Finder</td>
<td>4.5-5.7</td>
<td>128</td>
</tr>
<tr>
<td>Tracklet Selection</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Vertex Fitter</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tracker</th>
<th>Clock (GHz)</th>
<th>RAM (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracklet Finder</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Trkl duplicate finder</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Cell finder</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Cell neighbour finder</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Road finder</td>
<td>🚧</td>
<td>🚧</td>
</tr>
<tr>
<td>Track fitting</td>
<td>✟</td>
<td>✟</td>
</tr>
</tbody>
</table>
Preliminary performance

- **Total timing** measured on real data
  - A batch of 5 timeframes of \( pp \) collisions @500kHz
  - CPU is run in single thread configuration

- **Three reported scenarios**
  - Full vertex finding on GPU and CPU
  - State-of-the-art tracking chain on CPU and GPU
  - Full tracker run in the asynchronous configuration on CPU

- **Considerations**
  - The timing is promising if the primary goal is to trade GPUs for CPUs
  - The most time-consuming part is the track fitting, high rewards expected
  - Streaming chunks of a timeframe works successfully
  - Timing decreases with memory increasing, then reaches a plateau

<table>
<thead>
<tr>
<th></th>
<th>AMD EPYC™</th>
<th>AMD Ryzen™</th>
<th>AMD Mi50</th>
<th>Nvidia™ TITAN Xp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vertexer</strong></td>
<td>2913±376</td>
<td>1416±183</td>
<td>291±38</td>
<td>478±64</td>
</tr>
<tr>
<td><strong>Tracker (Neigh. Finder)</strong></td>
<td>550±71</td>
<td>287±37</td>
<td>211±27</td>
<td>779±105</td>
</tr>
<tr>
<td><strong>Tracker Full</strong></td>
<td>13756±1780</td>
<td>6917±893</td>
<td>🚧</td>
<td>🚧</td>
</tr>
</tbody>
</table>

GPU ITS vertexer elapsed time vs memory

GPU ITS tracker (neigh-finder) elapsed time vs memory

Matteo Concas, for the ALICE collaboration - CHEP 2023
Conclusions and outlook

- ALICE plans to extend the coverage of GPU utilisation in the asynchronous reconstruction
  - The goal is to increase the efficiency in using the resources when TPC does not have the monopoly

- ITS is finalising the porting of the seeding vertexer and tracking
  - Road finding and track fitting, the last missing components, are under active development
  - Performance in $pp$ collisions from actual data is not final but shows some promising margin
  - A full investigation of heavier conditions such as PbPb will show the fundamental capabilities and limitations

- Optimisation of the algorithms is to start after the finalisation of the porting
  - Tuning for GPU parameters can be performed with general-purpose tools for optimisation

- GPU adoption in the ITS software chain can be further extended
  - Signal digitisation and Clusterisation part are good candidates that are being considered

---

[1] "A parameter optimisation toolchain for Monte Carlo detector simulation"
Backup
Heterogeneous-Compute Interface for Portability

- Support GPUs from two main vendors:
  - CUDA language and runtime for Nvidia
  - HIP language and ROCm runtime for AMD

- HIP: a C++ Runtime API and Kernel language
  - Portable AMD and NVIDIA applications from single source code
  - It is shaped around CUDA APIs to ease translation
  - CUDA libraries, like Thrust and CUB, have their HIP versions using ROCm

- ROCm has tools to translate CUDA to HIP automatically
  - hipify-clang: based on Clang, actual code translation
  - hipify-perl: script for line-by-line code conversion

- Strategy: maintain only the CUDA code and generate HIP