ALICE GPU Algorithms

David Rohr for the ALICE Collaboration
drohr@cern.ch, CERN
Joint HSF/OSG/WLCG Workshop
Thomas Jefferson National Accelerator Facility
19.3.2019
ALICE will record a large minimum bias sample in continuous read-out during Run 3.  
- Online data compression mandatory.  
- Need online TPC tracking to achieved required data reduction.  
- ALICE will use GPUs in the online computing farm.  
- How to:  
  - use the GPUs efficiently?  
  - ensure that the same algorithms can run on processors in the GRID?  
  - remain independent from vendors?  
  - keep the code maintainable?  

- Overlapping events in TPC with realistic bunch structure @ 50 kHz Pb-Pb.  
- Timeframe of 2 ms shown (will be 10 – 20 ms in production).  
- Tracks of different collisions shown in different color.
Online / Offline Computing in ALICE in Run 3

- **Two reconstruction phases in Run 3:**
  - Synchronous reconstruction (during data taking):
    - **Calibration**
    - **Data compression**
  - Asynchronous reconstruction (when no beam):
    - **Full reconstruction with final calibration**

- Synchronous reconstruction at the O2 compute farm.
  - Farm fully under our control.
  - Must handle peak load of 50 kHz Pb-Pb in real time.
  - Workload mostly TPC tracking → Runs on GPU.

- Asynchronous reconstruction: partially O2 farm, partially GRID
  - More diverse workload, TPC tracking not dominant.
  - Large contribution from ITS tracking, can run on GPUs.
  - Must run on CPUs in the GRID.

![Diagram of online/offline computing in ALICE in Run 3](image-url)
GPU compatible workloads

- Synchronous reconstruction dominated by TPC tracking.
  - Fully running on GPUs and defines capacity of O2 farm → 1500 GPUs.
  - Other steps might run on GPU (e.g. ITS tracking of few % of events), but GPUs mostly needed for TPC.

- GPUs are available during asynchronous reconstruction.
  - Not dominated by TPC tracking → Asynchronous reconstruction at O2 farm has GPU capacity available.
  - GPUs in the GRID not so clear, but makes sense to leverage whatever might become available.
  - ITS tracking large part of asynchronous reconstruction, and most steps supports GPUs already.

Baseline solution (almost available today):
TPC + part of ITS tracking on GPU

Optimistic solution (what could we do in the ideal case):
Run most of tracking + X on GPU.

- Possible candidate for extended GPU usage: Full Barrel Tracking + related (medium priority).
- Running reconstruction of other detectors on GPUs could be studied in parallel (low priority).
- Both not strictly needed, baseline solution sufficient and must be finalized first (high priority).
Barrel Tracking Chain

- Many steps of barrel tracking must run consecutively.
  - TPC Standalone Tracking
  - ITS Standalone Tracking
  - TPC – ITS Matching
  - Prolongation into TRD
  - Prolongation into TOF
  - Global Refit

- 7 layers ITS (Inner Tracking System – silicon tracker)
- 152 pad rows TPC (Time Projection Chamber)
- 6 layers TRD (Transition Radiation Detector)
- 1 layer TOF (Time Of Flight Detector)
Barrel Tracking Chain

- Many steps of barrel tracking must run consecutively.
- Makes sense to port consecutive steps to GPU to avoid data transfer.
  - Although not strictly needed, depends also on data size. TPC clusters are most critical.
- Beginning of tracking chain with TPC / ITS well established on GPU already.
- TRD already available, but TPC / ITS matching missing.
- Following steps could be ported when there is manpower available.
- Primary focus right now: consolidate baseline solution.

TPC Track Finding \(\rightarrow\) TPC Track Merging \(\rightarrow\) TPC Track Fit

ITS Vertexing \(\rightarrow\) ITS Track Finding \(\rightarrow\) ITS Track Fit

TPC Track Model Compression \(\rightarrow\) TPC dE/dx \(\rightarrow\) TRD Tracking

TPC Junk Identification \(\rightarrow\) TPC Cluster removal

TPC Entropy Compression

Global Fit

TOF Matching

Match TPC tracks to remaining hits in ITS.

V0 Finding

In operation
Nearly ready
Being studied
Not started
TPC Tracking performance

- Speed-up normalized to single CPU core.
  - Red curve: algorithm speed-up.
  - Other curves: GPU v.s. CPU speed-up corrected for CPU resources.
    - How many cores does the GPU replace.

- Significant gain with newer GPU (blue v.s. green).

- GPU with Run 3 algorithm replaces > 800 CPU cores
  Running Run 2 algorithm. (blue * red).
  (at same efficiency / resolution).

- We see ~30% speedup with new GPU generation
  (RTX 2080 v.s. GTX 1080)

Algorithm speed-up on CPU
20 - 25x v.s. to Run 2 Offline

Modern GPU replaces
40 CPU cores @ 4.2 GHz

GPU of Run 2 HLT replaces 17 cores

Min.bias collision

Occupancy @ 50kHz

ALICE Performance 2018/03/20
2015, Pb-Pb, $\sqrt{s_{NN}} = 5.02$ TeV

ALI-PERF-143852
TPC Tracking performance

- Speed-up normalized to single CPU core.
  - Red curve: algorithm speed-up.
  - Other curves: GPU v.s. CPU speed-up corrected for CPU resources.
    - How many cores does the GPU replace.

- Significant gain with newer GPU (blue v.s. green).

- GPU with Run 3 algorithm replaces > 800 CPU cores
  Running Run 2 algorithm. (blue * red). (at same efficiency / resolution).

- We see ~30% speedup with new GPU generation (RTX 2080 v.s. GTX 1080)

- Modern GPU replaces 40 CPU cores @ 4.2 GHz
  Algorithm speed-up on CPU 20 - 25x v.s. to Run 2 Offline

- GPUs alone are not enough!
- We need faster algorithms as well!
- (obtained 1 order of magnitude by software optimizations and 1 order of magnitude via GPUs)
Performance Tuning

- GPUs of different vendor’s / generation’s might favor different tuning.
  - Many algorithms have tunable parameters (for processing speed).
  - We implemented most features such, that they can be switched off.
    - Worst case, at compile time via preprocessor definition.
- One example: Distribution of tracks among GPU threads during track following:
  - Illustration of active GPU threads over time (time on y-axis).
    - Number of average idle threads reduced by factor ~3, but large overhead for rescheduling.
    - Yields ~50% speedup on some GPUs, but becomes even slower on others.
- For new GPUs:
  - Run a benchmark with a parameter range scan to find best settings.
  - After 3 iterations (GPU generations), we got good results out of the box.
Performance Tuning

- **Handling of asynchronous computation / data transfers**
  - **1st iteration (Run 1 HLT):** Split event in chunks, to pipeline CPU processing, GPU processing, and PCIe transfer.
  - **2nd iteration (Run 2 HLT):** Processing of two events in parallel on the GPU concurrently.
    - ~20% faster than first version – GPUs have become wider and this exploits the parallelism better.
    - Not possible during Run 1 due to GPU limitations at that time.
    - We still kept the pipeline-scheme within each event, to maximize performance.
  - **3rd iteration (Run 3):** Go back to the old scheme from Run 1 – with time frames instead of events.
    - Time frames are large \(\rightarrow\) avoid keeping multiple in memory.
    - Enough parallelism inside one time frame.
Run 3 processing on GPUs

- Which of the ALICE Run 3 computing workload can benefit from GPUs?

<table>
<thead>
<tr>
<th>Processing type</th>
<th>Location</th>
<th>GPU usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous real-time reconstruction</td>
<td>O2 Farm</td>
<td>yes</td>
</tr>
<tr>
<td>Asynchronous raw-data reconstruction on the O2 farm.</td>
<td>O2 Farm</td>
<td>yes</td>
</tr>
<tr>
<td>Asynchronous raw-data reconstruction elsewhere</td>
<td>GRID</td>
<td>If possible</td>
</tr>
<tr>
<td>Monte-Carlo Reconstruction</td>
<td>GRID</td>
<td>If possible</td>
</tr>
<tr>
<td>Monte-Carlo Simulation</td>
<td>GRID</td>
<td>Not foreseen yet</td>
</tr>
<tr>
<td>Analysis</td>
<td>GRID</td>
<td>Not foreseen yet</td>
</tr>
</tbody>
</table>
Shared source code between CPU and GPU

- We employ a single source-code for CPU and GPU.
  - It can parallelize on the CPU via OpenMP.
  - We support GPUs via CUDA, OpenCL, HIP.

- CPU and GPU tracker (CUDA, OpenCL, ...) share common source files.
- Specialist wrappers for CPU and GPU exist, that include these common files.

⇒ Same source code for CPU and GPU version
  - The macros are used for API-specific keywords only.
  - The fraction of common source code is above 90%.

```c
common.cpp:
__DECL FitTrack(int n) {
    ....
}

cpu_wrapper.cpp:
#define __DECL void
#include ``common.cpp``
void FitTracks() {
    for (int i = 0; i < nTr; i++) {
        FitTrack(n);
    }
}

cuda_wrapper.cpp:
#define __DECL __device void
#include ``common.cpp``
__global void FitTracksGPU() {
    FitTrack(threadIdx.x);
}

→ Opencl_wrapper.cl
#define __DECL void
#include ``common.cpp``
__kernel void FitTracksGPU() {
    FitTrack(threadIdx.x);
}

Opencl_wrapper.cxx
void FitTracks() {
    clEnqueueNDRangeKernel(...
    FitTracksGPU<<<nTr>>>();
}
```
Distribution of same software to CPU and non-GPU nodes

- A lesson learned in the HLT farm: we should have a common software package:
  - The HLT farm consisted of GPU-equipped nodes and nodes without GPU.
  - GPU code that links to the CUDA runtime requires the CUDA runtime library to be present.
  - Just installing the library on non-GPU nodes is insufficient, as the library fails to load when the kernel module is not loaded.
  - Back then, the kernel module failed to load without an NVIDIA device present.

- Still, we do not want to ship different software packages.
  - To facilitate software distribution, we have one binary package that contains all versions.
    - The GPU versions of the code are contained in special GPU-tracking libraries.
    - These GPU-tracking libraries are accessed via dlopen.
    - Only the GPU-tracking libraries link to the GPU driver / runtime.
  → The tracking software (without GPU acceleration) runs on all compute nodes, irrespective of the presence of a driver.

```
Common Tracking Part (including CPU)  Tracking part for CUDA  CUDA Runtime
  dlopen                                    normal .so linking
Tracking part for OpenCL  OpenCL Runtime
```

Provided by NVIDIA / AMD driver
- Detailed example of the track fit code.
  - Majority of the code is in Algorithm.cxx, which is shared between CPU and all GPU versions.
  - libFit can be loaded on all compute nodes (no dependency on CUDA / OpenCL).
    - The cuda and OpenCL tracking libraries (libFitCUDA and libFitOpenCL) can be loaded when the respective runtime (libCUDA or libOpenCL) is present.
• Try to keep as much code shared as possible.
  • Using some #ifdefs to have some shortcuts for serial CPU version, or use special GPU optimizations.
  • Some compiler macros, e.g. to cache data in GPU shared memory.

• Can use different depth of abstraction:

  - **Baseline:**
    - Kernels are common and called by wrappers

  - **More abstraction:**
    - Also kernel calls, outer loops, and general workflow common (used in TPC)
    - Exact same algorithm for CPU / GPU

  - **More flexibility:**
    - Common code for base classes, virtual class with “traits” for CPU and GPU steer the workflow. Main tracking class is common and uses the traits. (used in ITS)

  - **Less API-specific code**

  - **Less complicated abstraction**
Details of GPU implementation

- **Software can run in 3 modes:**
  - Enforce GPU.
  - CPU-only.
  - Opportunistic GPU usage with automatic CPU fallback.

- **In any case, the same virtual interface class is used to drive the backend.**

- O2 farm will enforce GPU usage.
- CPU only might be for some GRID sites, or when we encounter problems.
- Opportunistic usage is for your laptop, use whatever hardware is available.
• All our software requires C++.
  • We started with CUDA in 2010 (OpenCL was C only).

• Now, there are some more options:
  • CUDA still supported.
  • CPU version available anyway.
  • OpenCL 1.2 with AMD OpenCL C++ extensions
    – Used in the Run 2 HLT farm, still supported to reused GPUs for O2 tests until commissioning of final O2 farm.
    – No longer supported on modern AMD GPUs.
  • AMD HIP
    – Similar to CUDA, added to support modern AMD GPUs independent of OpenCL.
  • OpenCL 2.2 with C++
    – Supported easily, but neither software nor hardware available yet.
  • Clang OpenCL C++
    – Basically OpenCL 2.0 API with OpenCL 2.2 kernel language).
    – Found several compiler bugs so far, but the clang team is fast in fixing them.
    – Hope to have a working version with the release of clang 9.

→ Feature set is restricted to smallest common denominator, but no problem for us!
GPU Features

• We have no need for professional GPU features.
  • All software is single precision.
  • Raw data comes from a radiation environment anyway.
  • No requirement for special virtualization or container support so far.
  • Full duplex PCIe / RDMA to GPU / Faster inter-GPU link not needed so far.

• For Run 2 reconstruction, 2 GB of memory was enough.
• Run 3 will need much more memory due to large time frames.
  • Either use GPUs with large memory.
  • Or split time frames in chunks, with special treatment at the borders.
  • Global cost/performance optimization.

• We’ll go with new C++ standards, for Run 2 we might be at C++20 and would aim for a compatible CUDA, etc. version. That might require up to date drivers at that time.
  • Not clear whether we want to update drivers / OS during the Run.
Consistency of Tracking Results

- **First version: Inconsistent results between CPU / GPU**
- **Root causes:**
  - Concurrency
  - Non-associative floating point arithmetic

- **Problem:** Cluster to track assignment was depending on the order of the tracks.
  - Each cluster was assigned to the longest possible track. Out of two tracks of the same length, the first one was chosen.
  - Concurrent GPU tracking processes the tracks in an undefined order.

- **Solution:** Need a continuous (floating point) measure of the track quality.
  - Two 32-bit floats can still be identical, but that is unlikely.
• To simplify Q/A, GPU and CPU results should be as close as possible!
• 100% identical result on binary level is difficult.
  • Prevents all fast-match optimization → >10% performance loss.
  • Difficult to kill concurrency issue completely:
    – 32-bit float gives ca 25 bit of entropy (do not use full range of exponent).
    – 1000 Events of ~ 10000 tracks per time frame → collision probability non-zero.
    – Could use double / 64-bit integer…

• Good measure for us:
  • Do not compare track parameters (float values might differ).
  • Compare cluster to track association.
    – Almost binarily compatible: 0.00024% of clusters assigned differently.
Some lessons learned

- **Usually no issue with consumer cards, but hard to guarantee**
  - Had one problem with a batch of 30 GTX580 cards we added to the HLT Run 1 farm in addition to the GTX480 cards we already had:
    - Produced memory errors after several days of constant operation, was never understood, replaced by new GTX480 cards.
    - Found similar reports in the NVIDIA Forum for GTX580, but infrequent rendering errors in games are no real issue.

- **Active GPU cooling is a real issue in server chassis.**
  - For tracking, the GPU will not overheat regardless of inefficient cooling (For the Linpack benchmark, it will).
  - It might be more efficient to remove the fan and leave only the heatsink, depending on the orientation of the fins.
  - Some fans are only specified to be rotated 90 deg, but not to operate upside-down.
  - Side pressure in server chassis with high-performance fans can damage GPU fans.

- **Didn’t see much difference between ECC and non-ECC memory.**
  - If at all, performance degrades with ECC, even disabled ECC in some cases.
  - All raw data comes from a radiation environment anyway.

- **GPUs are not more likely to fail than other components such as mainboards / RAM.**
Summary

- ALICE will heavily employ GPUs during Run 3 online processing.
- Mandatory for TPC tracking during synchronous phase.
- Use for other detectors and during asynchronous phase where available.
- Baseline solution with TPC + part of ITS tracking nearly fully implemented.
- One order of magnitude speedup by new algorithm, one order of magnitude via GPU usage.
- Tuning should be switchable and in a general way – must run fast on CPU and GPU.
- Code written in generic C++ language, supports OpenMP, CUDA, OpenCL, HIP with common source code.
- Identical binaries used on GPU-servers and normal servers.
- No strict need for professional features like double precision, ECC, … / but no active cooling!
- Difficult to achieve binarily identical results on CPU / GPU:
  - Requires disabling of math optimization and usage of 64 bit weight functions → performance degradation.