A vendor-agnostic, single code-based GPU tracking for the Inner Tracking System of the ALICE experiment

M Concas

Sezione di Torino, Istituto Nazionale di Fisica Nucleare, Via Pietro Giuria 1, Torino 10125, IT

E-mail: matteo.concas@cern.ch

Abstract. During the LHC Run 3 the ALICE online computing farm will process up to 50 times more Pb-Pb events per second than in Run 2. The implied computing resource scaling requires a shift in the approach that comprises the extensive usage of Graphics Processing Units (GPU) for the processing. We will give an overview of the state of the art for the data reconstruction on GPUs in ALICE, with additional focus on the Inner Tracking System detector. A detailed teardown of adopted techniques, implemented algorithms and approaches and performance report will be shown. Additionally, we will show how we support different GPUs brands (NVIDIA and AMD) with a single code-base using an automatic code translation and generation for different target architectures. Strengths and possible weaknesses of this approach will be discussed. Finally, an overview of the next steps towards an even more comprehensive usage of GPUs in ALICE software will be illustrated.

1. Introduction

During the Run 3 at the Large Hadron Collider (LHC) the accelerator will deliver a rate of Pb-Pb collisions up to 50 kHz. The A Large Ion Collider Experiment (ALICE)[1] apparatus aims at recording an integrated luminosity greater than 10 nb^{-1} of minimum bias events. Such a huge sample of data corresponds to a factor 50 times more compared to what collected in Run 2 and represents a great challenge under many experimental perspectives, both in hardware and and software. ALICE is adopting a new data acquisition strategy called "continuous readout" using a *trigger-less* approach that steadily registers input from every sub-detectors. The resulting continuous stream of data is split in *timeframes* with a duration of the order of tenths of milliseconds containing the cumulative information from multiple events. The overall starting bandwidth for raw data is greater than 3.5 TB/s, after a first pass of reduction and compression done by the "First Level Processing" it is reduced down to 600 GB/s and it is impossible to directly store them on permanent media. To this extent, ALICE is going to further reduce the amount of information by operating the online reconstruction of the data. A completely renovated framework encompassing both the online and the offline software for Run 3 in a single stack called O^2[2] will run on a dedicated on-site computing farm composed by Event Processing Nodes (EPNs), the "EPN farm".

The online reconstruction is divided in two phases: the *synchronous* and the *asynchronous* reconstruction. The former runs during the data taking strictly on the EPN cluster and operates the reconstruction of the most demanding detectors, for instance the Time Projection Chamber

(TPC) and a fraction of the Inner Tracking System (ITS)[3] to be used as calibration information. The latter performs the full reconstruction of all the data before saving them on permanent storage and will run during technical stops or during less demanding online reconstruction (e.g. runs of pp collisions) both on EPN farm and on the Grid.

The EPN computing cluster is composed by 250 nodes, each equipped with 64 cores CPUs and 8 Graphics Processing Units. The resulting amount of computing resources is then computed to provide the best cost over performance ratio to fulfill the synchronous reconstruction of the TPC detector. Using 1500 GPUs will be sufficient to cope with 50 kHz Pb-Pb collisions.

2. Using GPUs in ALICE from Run 2 to Run 3

Already during the Run 2 the TPC was offloading some of the data compression tasks on the High-Level Trigger (HLT) on GPUs. In general, the software stack designed to steer the operations was designed to support multiple target architectures and parallelism paradigms via different libraries, thus resulting able to run different systems other than the HLT. As an example OpenMP[4] was adopted for the CPU parallel version, CUDA[5] on NVIDIA graphics cards and OpenCL[6] v1.2,2 for the remaining supported accelerators and co-processors. Schema in 1 shows the structure of the approach in dynamic loading for multiple platforms. The selection is dynamically applied at program *runtime*: after an automatic detection of the underlying computing device, the corresponding algorithmic library is dynamically loaded upon request. The interface is transparent: a single class is used to call the main functions (e.g. fitting function included in a comprehensive Algorithm class) and includes the headers of libraries related to the available devices. A second class is responsible to call the actual implementations of the requested function via the dynamic loading of corresponding library.

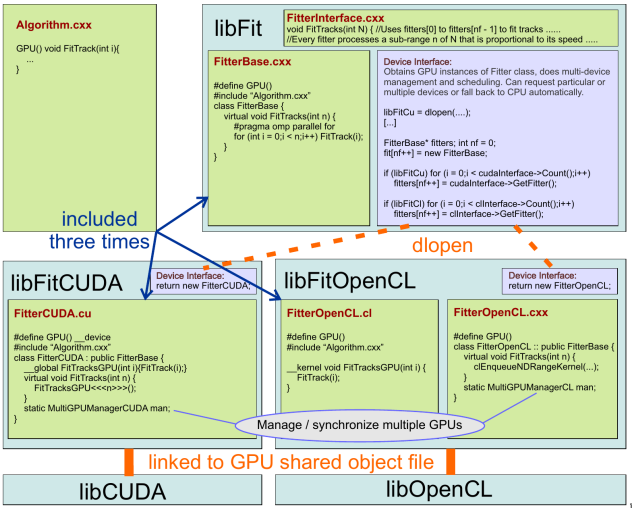

Figure 1. Example of the organization of fitting libraries in the unified framework. The `Algorithm` class provides a generic function for track fitting. A dedicated library `libFit` takes care of loading the required implementation of the symbols depending on the required device to operate with. It then dynamically loads the corresponding detailed libraries. The framework is supporting multiple GPU management in all of the libraries implemented.

In Run 3 the usage of graphics cards is required to be able to perform the synchronous reconstruction for TPC, to minimise the cost over performance ratio for the EPN farm. Figure 2 shows how GPUs can efficiently supply computing power scaling proportionally to the size of the input data in the TPC reconstruction for Run 3. In figure 3 it is also depicted the speedup

achieved by using different GPU models and vendors normalized to the number of CPU cores (AMD Rome at 3.3 GHz) showing the actual advantage in trading CPU resources for GPU ones. The framework to steer the GPU reconstruction in Run 3 has a similar schema to its predecessor

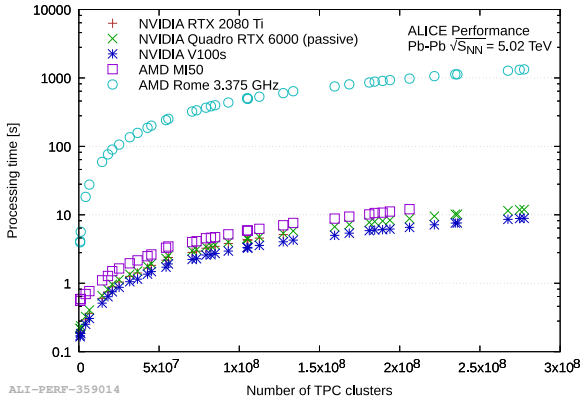

Figure 2. Computing time for TPC reconstruction linearly scales as a function of the number of processed clusters.

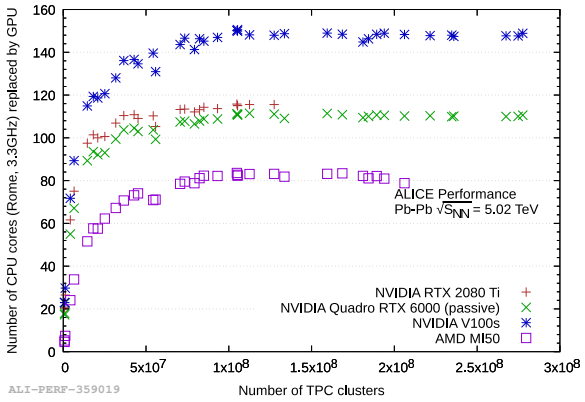

Figure 3. Average speedup normalised to 1 core for different GPUs. It's possible to trade up to 150 CPU cores with one GPU.

it evolved from, with an extended scope in term of the type of tasks that nowadays run on GPUs. For an efficient usage of the resources on the EPN farm and to be able, in the future, also to exploit GPU pledges on the Grid, an increasing amount of the computing effort for the asynchronous reconstruction is being moved to GPUs. To this extent, the GPU reconstruction framework includes enhanced capabilities, such as the ability of integrating externally developed libraries for GPU code (e.g. GPU reconstruction for ITS) and provide the convenient dynamic load of them, reducing the amount of duplicated code to implement such a functionality on a per-detector basis.

3. GPU in ITS reconstruction

The ALICE Inner Tracking System is among the detectors aiming at exploiting the GPU potential for their asynchronous reconstruction.

3.1. ITS reconstruction

The ITS is the innermost detector of the experiment, it is composed by 7 layers of silicon pixels, for a total 12.5 Gigapixels and 10 m^2 of sensitive area. Among its duties the two main one are the measure of the position of the beam collision points, also called the collision vertices, and the reconstruction of the particle trajectories at their early stages right after the collisions. These two tasks are accomplished respectively by two pieces of reconstruction software: the primary vertex seeding and the tracker. In both cases the complexity of the problem is dominated by combinatorial matches between associations of elements (e.g. clusters of pixels, segments of track candidates, etc.). The algorithms employed are typically *embarrassingly parallel* at many levels with very few exceptions in the process where more sophisticated developments are required. More demanding use cases (i.e. TPC reconstruction) share a similar type of complexity but scaled to a larger number of points per track, resulting in relevant speedups. In the case of ITS reconstruction, preliminary results shows that even in this case it is possible not only to effectively use graphics cards to operate the reconstruction, but it's also possible to achieve a faster execution that releases CPU resources for other non offloadable tasks.

3.2. GPU usage in asynchronous phase

GPU reconstruction for the ITS is a new development started with preparation for Run 3. Design choices such as the supported platforms and libraries used for development are based on recent years state of the art. Currently two target GPU brands are supported: Advanced Micro Devices (AMD) and NVIDIA. The former is developed using the Heterogeneous-Compute Interface for Portability (HIP)[7] language through the Radeon Open Compute ROCm platform and aims at efficiently running on the devices available on the EPN cluster. The latter uses the Compute Unified Device Architecture (CUDA)[5]. By using the possibility to dynamically loading external GPU libraries in the main GPU reconstruction framework previously described, it has been possible to develop and prototype separate libraries, independent from the main component. This has the advantage to reduce the learning curve related to the specific approaches used by already present GPU code, and to have something that is naturally integrated into the main framework, thus reducing the amount of duplicate code to support the execution. Especially during prototyping and testing stage, this facilitates the development. If required, the integration inside the existing GPU libraries can be postponed after the finalization.

4. Develop and maintain two different code bases with minimum redundancy

Typical aspect of supporting multiple platforms is to cope with redundant and duplicated code minimisation, so that the effort in developing and maintaining the two or more source code is very low. This can be addressed using different strategies like external libraries for portability. However, in order to reduce the dependencies and to avoid learning additional APIs, it will be later described another approach based on the features of platform already used which results in a very convenient approach for this specific use case.

4.1. HIP: Heterogeneous-Compute Interface for Portability

HIP is a C++ runtime API and a kernel language for portable AMD and NVIDIA applications. The design of its API resembles by construction the CUDA ones with a 1:1 mapping and covering all the shared capabilities among the two different platforms. The relative small set of features that related to NVIDIA-only devices is not supported. Most of the commonly used external CUDA libraries find their counterparts in the ROCm[8] ecosystem. It also provides some semantic and literal "translators" (i.e. `hipify-clang` and `hipify-perl`) which are able to convert source files written in CUDA to the corresponding HIP version. Figure 4 shows a minimal example of the corresponding mapping of HIP to CUDA APIs.

```
// 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);
```

Figure 4. Code to compute the square of an array on GPU, HIP version is automatically generated from CUDA code. CUDA and HIP API calls have a strict correspondence.

4.2. Generating HIP code on-the-fly
The compilation process of the O^2 is managed by CMake[9]. It is able to perform a platform auto-detection: it automatically produces dedicated libraries depending on supported GPU devices and installed framework (e.g. CUDA, HIP, OpenCL, ...). It is also highly configurable and it's able to perform custom commands and establish dependencies across targets. This is very useful during the development, as the final approach is to write one single code and dynamically produce two version of it to be compiled against each target. The workflow to support with a single code base can be summarised in few steps. Code is developed using CUDA, all the C++ platform specific implementations are placed in the declaration files, header files are written to be portable between CUDA and HIP. Pre-compiler macros are used to solve possible divergences between the nomenclatures in two dialects for types and function definitions. The amount of dedicated code to support portability is negligible and is only related to the inclusion of HIP headers. During the compilation CMake identifies the supported platforms and add the corresponding targets to the list of libraries to produce. If HIP is requested, it runs a custom command that feeds all the CUDA (`.cu`) files to the `hipify-perl` translation script that produces in-places HIP (`.hip.cxx`) files. Header files are used at each compilation instance and do not need to be translated, thus reducing the amount of redundant information. By establishing dependencies among targets it was also possible to add the re-trigger of the translation of CUDA files upon any change on them. Initially, this approach has been developed and tested for a memory benchmark tool written to compare GPU I/O performance of devices from different vendors. Currently this approach is successfully used in the ITS GPU reconstruction code.

5. Conclusion
During Run 3 data taking the ALICE experiment is using GPUs to enable an online data reconstruction. The main use case, the TPC reconstruction has been successfully tested during the pilot beam in October in pp collisions at 900 GeV. For ALICE asynchronous reconstruction the plan is to extend the usage of graphics cards to include more use cases The ITS aims at having a working GPU version of the tracking algorithms. The approach is to have CUDA and HIP platforms served by a single code base, duplicated at build time. Primary vertexer is already functional, and works with visible speedup: up to x12 times in a not optimized implementation on CUDA and x5 on AMD. A prototype GPU tracking has been developed in the past, it is currently under development to be updated to current reconstruction schema in ITS. The automatic translation of the CUDA code allows us to maintain and develop a single code with minimal additions with respect to what is already present in the O^2 for GPU support. Therefore it does not require any external dedicated compatibility or portability library.

References
[1] ALICE Collaboration, "The ALICE experiment at the CERN LHC", J. Inst. 3 S08002 (2008)
[2]] P. Buncic, M. Krzewicki, P. Vande Vyvre, *Technical Design Report for the Upgrade of the Online-Offline Computing System (2015)*
[3] ALICE Collaboration, "Technical Design Report for the Upgrade of the ALICE Inner Tracking System", CERN-LHCC-2013-024 (2013)
[4] OpenMP Architecture Review Board, "OpenMP Application Program Interface Version 3.0" (2008)
[5] John Nickolls, Ian Buck, Michael Garland, Kevin Skadron, "Scalable Parallel Programming with CUDA" (2008)
[6] J. E. Stone, D. Gohara and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," in Computing in Science and Engineering, vol. 12, no. 3, pp. 66-73, May-June 2010, doi: 10.1109/MCSE.2010.69.
[7] HIP, https://rocm-developer-tools.github.io/HIP/
[8] AMD, ROCm, https://rocmdocs.amd.com/en/latest/Current_Release_Notes/Current-Release-Notes.html
[9] CMake, https://cmake.org/