Experience in SYCL/oneAPI for event reconstruction at the CMS experiment

Nikolaos Andriotis[1], **Andrea Bocci**[2], **Laura Cappelli**[3], **Tony Di Pilato**[4,9], **Luca Ferragina**[5], **Juan Jose Olivera Loyola**[6], **Felice Pantaleo**[2], **Aurora Perego**[7], **Wahid Redjeb**[2,8]

[1]Barcelona Supercomputing Center, Spain, [2]CERN, Switzerland, [3]INFN CNAF Bologna, Italy, [4]Center for Advanced Systems Understanding (CASUS), [5]University of Bologna, Italy, [6]Institute of Technology and Higher Studies of Monterrey, Mexico, [7]University of Milano Bicocca, Italy, [8]RWTH Aachen University, Germany, [9]University of Geneva, Switzerland

E-mail: `aurora.perego@cern.ch`

Abstract. The CMS software framework (CMSSW) has been recently extended to integrate heterogeneous computing with the aim of performing part of the physics reconstruction with GPUs to face the computational challenge that will come with the increase of luminosity during the High-Luminosity phase of the Large Hadron Collider (HL-LHC). To avoid writing a different implementation of the code for each backend, a performance portability library is used: Alpaka has been chosen as the compatibility layer for Run-3. With the idea of exploring new solutions, SYCL has been considered since it would allow to target also Intel GPUs.

SYCL is a cross-platform abstraction C++ programming model for heterogeneous computing. It allows developers to reuse code across different hardware and also perform custom tuning for a specific accelerator. The SYCL implementation used in this work is the Data Parallel C++ library (DPC++) in the Intel oneAPI Toolkit.

This work shows the performance of physics reconstruction algorithms written in SYCL on different hardware. Strengths and weaknesses of this heterogeneous programming model will also be presented.

1. Heterogeneous computing and performance portability libraries

In the High-Luminosity phase of the LHC (HL-LHC) the accelerator will reach an instantaneous luminosity of $7 \cdot 10^{34}$ cm^{-2} s^{-1}, with an average pileup of 200 proton-proton collisions. This will lead to a computational challenge for the online and offline reconstruction software that has been and will be developed. To face this complexity online CMS has decided to leverage heterogeneous computing [1], meaning that accelerators will be used instead of CPUs only.

To be prepared for Run 4, CMS equipped the current Run 3 production High Level Trigger (HLT) nodes with NVIDIA Tesla T4 GPUs that are currently used for: pixel unpacking, local reconstruction, tracks and vertices, ECAL unpacking and local reconstruction, HCAL local reconstruction.

This choice has resulted in higher throughput thanks to the use of accelerators, better physics performance due to the fundamental redesign of the algorithms to fully exploit the parallelism capabilities of GPUs and better performance per kW and per cost. The goal for the HL-LHC is to offload at least 50% of the HLT computations to GPUs in Run 4 scaling up to 80% in Run 5.

Currently, the code to be executed on GPUs is written in CUDA specifically for NVIDIA GPUs [2]. This approach should be avoided because it introduces code-duplication that is not easily maintainable. A possible solution is the use of performance portability libraries, that allow the programmer to write a single source code which can be executed on different architectures.

The CMS experiment has evaluated some performance portability libraries and Alpaka [3] has been chosen for Run 3. The migration from CUDA to Alpaka at the HLT will be completed before the resuming of Run 3 in 2023. Despite this, studies on this kind of libraries are still ongoing and other solutions are being explored.

2. Intel implementation of SYCL: oneAPI and DPC++

Recently Intel started developing the Data Parallel C++ (DPC++), an open-source compiler project that is part of the oneAPI programming model [4]. DPC++ is based on SYCL [5], a cross-platform abstraction layer maintained by Khronos Group that allows code to be written using standard ISO C++ both for the host and the device in the same source file. DPC++ supports many backends: CPUs, Intel FPGAs, Intel, NVIDIA and AMD GPUs. The support for the CUDA and HIP backends has been introduced in the Intel oneAPI 2023 release (released December 2022), therefore in this work to target NVIDIA and AMD GPUs the open source Intel's LLVM Compiler [6] has been used.

The single-source code written in SYCL can be compiled for different accelerators allowing the user to choose the preferred device at runtime.

The SYCL programming model starts from the definition of a queue, associated to a specific device, that is used to submit memory operations and kernels. Those operations run asynchronously on the device and the execution order is sequential only if explicitly specified, therefore synchronization is required before accessing results from the host.

There are different reasons to explore this library starting from the fact that when a new compatibility layer is developed, it is always relevant to explore its performance in comparison to native code and other compatibility layers. Being a royalty-free project, SYCL can count on a variety of implementations. This allows for more flexible adaptations of the standard to specific needs. Furthermore, it is supported by many actors in the tech industry, which bodes well for the future support of the standard.

3. Writing reconstruction algorithms with SYCL/oneAPI

SYCL/oneAPI has been tested on two workflows:

- CLUE [7] [8]: CMSSW TICL clustering algorithm
- pixeltrack-standalone [9] [10]: the Patatrack pixel tracks and vertices reconstruction

The data used to perform the benchmark are available in both repositories. To ease the testing process, a simplified version of CMSSW framework [11] has been developed to allow the testing of specific modules. For both workflows, the framework and the modules have been implemented in C++ (native serial) and CUDA and then with different portability libraries.

The goal is to test those performance portability solutions based on: programming experience/ease of development, ease of Framework integration, performance with respect to the native implementation, performance of fallback to CPU, user support, supported backends and long-term prospects. In this work, all of these points will be addressed for what concerns SYCL.

Here the programming experience is assessed, based on the porting of both the framework and the algorithms from the CUDA implementation. Starting from the framework integration, the device selection, memory operations and queues and events management had to be adapted to SYCL logic. A device selector has been introduced since the CUDA version did not need one being able to target only NVIDIA GPUs. Regarding the framework, the main change in the

SYCL version is the dealing with `sycl::events` that, unlike the CUDA ones, cannot be reused during the execution leading to the creation of new ones when needed with the consequence of slowing down the execution.

The first algorithm ported to SYCL is CLUE, short for CLUstering of Energy, a fast and parallelizable density-based clustering algorithm specifically designed to be used in high granularity calorimeters. There are two possible implementations both starting the clustering procedure from hits: CLUE 2D creates clusters on a single layer of the detector and CLUE 3D creates tracksters, i.e. clusters across all the detector layers. Both algorithms have been written in SYCL and tested. CLUE contains few different kernels without complex patterns, therefore the migration from CUDA consisted mostly in adapting the kernels submission. This process was quite smooth and no major challenges were encountered.

The Patatrack pixel tracks and vertices reconstruction is more complex since it consists of more than thirty kernels with particular patterns used to better exploit parallelism, like warp-level primitives and group functions. The migration of the code in this case was not trivial and highlighted some of the current limitations of SYCL. In particular, a few bugs have been discovered and reported to Intel developers, while some features used in the CUDA version of the code are not yet supported. Both cases required the implementation of workarounds to obtain a fully working application. An additional challenge involved the Eigen library [12], used in pixel track reconstruction, which had only partial support for SYCL. Manual modifications were necessary to adapt it for use in the SYCL version of the pixel track reconstruction code.

All the physics results of the SYCL applications have been validated before looking at the computing performance.

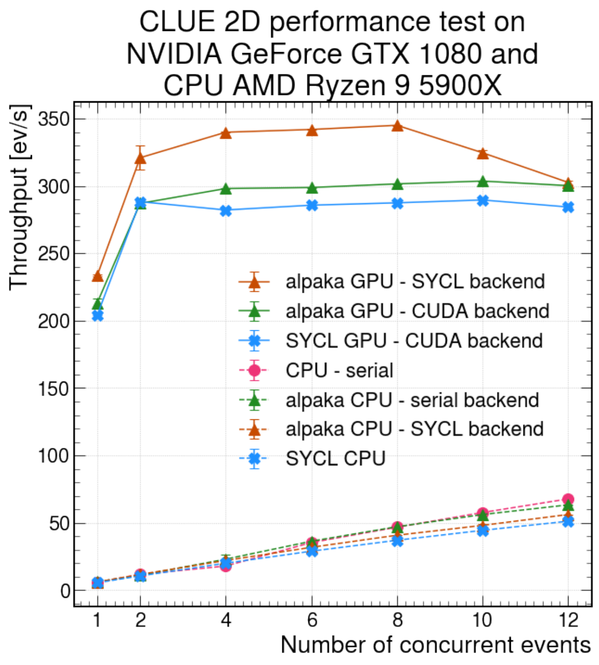
4. Computing performance

The performance has been evaluated on:

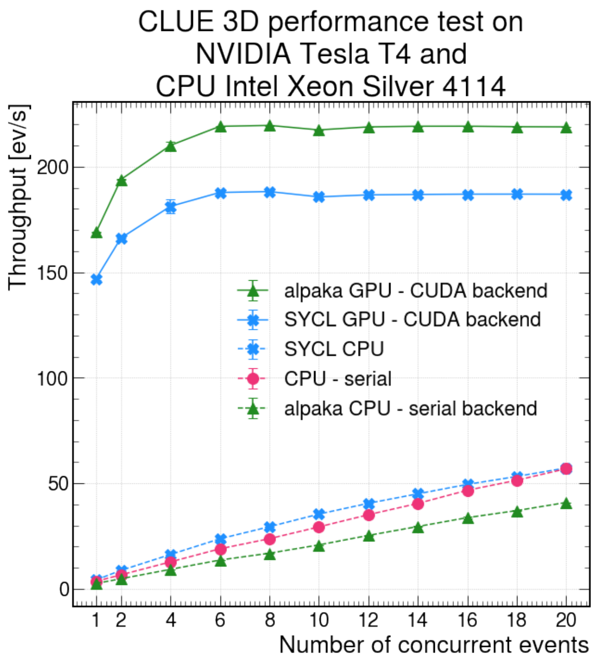
- CPU AMD Ryzen 9 5900X and GPU NVIDIA GeForce GTX 1080 for CLUE 2D and the pixel tracks reconstruction
- CPU Intel Xeon Silver 4114 and GPU NVIDIA Tesla T4 for CLUE 3D

The SYCL version (color: blue) has been compared with the already existing native versions (serial C++ and CUDA, color: pink) when available and with the Alpaka version. The native CUDA version is not available for CLUE since it is not maintained anymore (CLUE 2D) or has been written directly in Alpaka (CLUE 3D). The Alpaka library can target CPUs and NVIDIA GPUs using two different backends, one based on serial C++ or CUDA (color: green) and the other based on SYCL (color: orange). The SYCL backend of Alpaka has been introduced to target also Intel GPUs and FPGAs but is still in an experimental state. For the scope of this work, this backend has been used with little modification to target also NVIDIA GPUs. The memory management in SYCL can be achieved with either buffers or Unified Shared Memory (USM). In our applications USM has been used, while the SYCL backend of Alpaka supported only buffers. For this reason, the necessary changes have been made to add support for USM to be able to use it with our applications. At the moment of writing only CLUE 2D run successfully with the SYCL backend, while some bugs are preventing the testing on the other algorithms. Regarding the CPU backend, the main difference between the native version and the SYCL version is that the native version is serial, while SYCL introduces vectorization by default.

A plot has been realised for each application showing the throughput, i.e. the number of events per seconds, as a function of the number of concurrent events processed. As a general rule the solid lines refer to GPU benchmarks, while dashed lines are used for the CPU ones. The color code indicated above is the same for all the plots. Each point is the average of ten repetitions with 1k events on the CPU and 10k events on the GPU.



(a) Results for CLUE 2D varying the number of concurrent CPU threads on an GPU NVIDIA GeForce GTX 1080 (solid lines) and on an CPU AMD Ryzen 9 5900X (dashed lines). The GPU implementations are: the Alpaka version compiled with the SYCL backend and the CUDA backend, and the SYCL version. The CPU implementations are: the native C++ version, the Alpaka version compiled with the serial backend and the SYCL backend, and the SYCL version.



(b) Results for CLUE 3D varying the number of concurrent CPU threads on an GPU NVIDIA Tesla T4 (solid lines) and on an CPU Intel Xeon Silver 4114 (dashed lines). The GPU implementations are: the Alpaka version compiled with the CUDA backend and the SYCL version. The CPU implementations are: the SYCL version, the native C++ version and the Alpaka version compiled with the serial backend.

Figure 1: CLUE 2D and CLUE 3D performance plots

Starting from CLUE, the performance is shown in fig. 1a for CLUE 2D and in fig. 1b for CLUE 3D. The results on the CPU are comparable in both versions, while for the GPU the performance of SYCL is slightly worse than that of Alpaka. In CLUE 2D the Alpaka version running on an NVIDIA GPU through the SYCL backend outperforms the other versions, no explanation has been found to this yet. Overall, SYCL shows good performances on those application.

Looking at the performance of the pixel tracks reconstruction in fig. 2, the results show a bigger gap between SYCL and the native and Alpaka versions for both the GPU and CPU. Some profiling and analysis have been done to try to understand the reasons of this gap in performance. The main slowdowns observed concern methods that are slower with respect to their CUDA counterpart. This behaviour has been observed especially in the so-called group functions, that operate across threads of different blocks. Another worsening of the performance is due to the way events are defined in SYCL that prevents us from reusing them. Furthermore, an event is created by default for each memory operation and kernel submission even if not used. Although there is the possibility to avoid creating those events, it has the consequence of not being able to create events at all, making this option useless in our application.

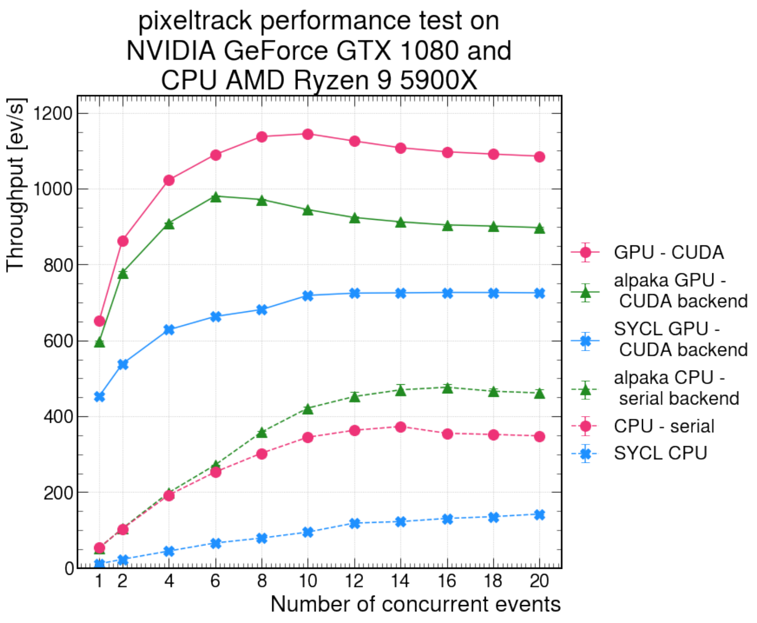


Figure 2: Results for the pixel tracks reconstruction varying the number of concurrent CPU threads on a GPU NVIDIA GeForce GTX 1080 (solid lines) and on a CPU AMD Ryzen 9 5900X (dashed lines). The GPU implementations are: the native CUDA version, the Alpaka version compiled with the CUDA backend and the SYCL version. The CPU implementations are: the Alpaka version compiled with the serial backend, the native C++ version and the SYCL version.

5. Conclusion and Future Perspectives

For the HL-LHC phase, the CMS detector will have to undergo significant upgrades to keep up with the massively increased data rates and also to renew components and modules deteriorated by the highly radioactive environment where the detectors are. This must be accompanied by corresponding software upgrades to process and analyze the collected data with very strict time constraints. This has led CMS to explore the possibilities offered by heterogeneous computing, which would allow it to offload part of the computing work to different accelerators. To avoid code duplication and enhance maintainability, compatibility layers have been explored and Alpaka has been chosen for Run 3. Throughout this work, the performance of SYCL has been explored with two CMS workflows. The results obtained in this context are somewhat promising, showing the ability to write a single source code and produce an executable able to run on CPUs, Intel GPUs, and NVIDIA GPUs with good performance results.

However, there still are some key issues in the SYCL implementation used: the good results achieved with CLUE have not been reproduced in the pixel tracks reconstruction. In this application there are many complex patterns that affected the performance since they are not optimized in SYCL or not fully supported yet. During the development, bugs in the compiler have been found that required workarounds to obtain a working version or affected the portability of the code. On the other hand, the long debugging process helped pushing the development of SYCL/oneAPI since all the issues found have been reported to the Intel team working on the compiler.

In conclusion, SYCL has shown really good performance on a simple application, but when looking at more complex patterns there is still a noticeable gap between SYCL and the native implementation that makes the former not suitable for being used as a portability layer in CMSSW in the near future.

Acknowledgments

This work has been supported and funded by CERN Openlab and Intel Corporation. Special thanks go to Igor Vorobtsov and Klaus-Dieter Oertel from the Intel Team for the support they gave us during the meetings we had throughout the course of this project.

References

[1] CMS Collaboration 2021 The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger Tech. rep. CERN Geneva URL https://cds.cern.ch/record/2759072

[2] CUDA Programming guide URL https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[3] alpaka - Abstraction Library for Parallel Kernel Acceleration https://github.com/alpaka-group/alpaka

[4] Intel® oneAPI DPC++/C++ Compiler URL https://www.intel.com/content/www/us/en/develop/documentation/oneapi-dpcpp-cpp-compiler-dev-guide-and-reference/top.html

[5] The Khronos® SYCL™ Working Group, SYCL™ 2020 Specification (revision 6) URL https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html

[6] Intel corporation, Intel llvm open source compiler, URL https://github.com/intel/llvm

[7] CLUE github repository URL https://github.com/cms-patatrack/heterogeneous-clue (commit 8dfc58c)

[8] Rovere M, Chen Z, Di Pilato A, Pantaleo F and Seez C 2020 CLUE: A Fast Parallel Clustering Algorithm for High Granularity Calorimeters in High-Energy Physics, *Front. Big Data* **3** 591315. URL https://cds.cern.ch/record/2709269

[9] Patatrack group, Standalone Patatrack pixel tracking URL https://github.com/cms-patatrack/pixeltrack-standalone (commit e9d2f33)

[10] Bocci A, Kortelainen M, Innocente V, Pantaleo F and Rovere M 2020 Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker, *Front. Big Data* **3** 601728. URL http://cds.cern.ch/record/2744911

[11] CMSSW Application Framework URL https://github.com/cms-sw/cmssw

[12] Guennebaud G, Jacob B and others 2010 URL http://eigen.tuxfamily.org