Contribution ID: **545** Contribution code: **contribution ID 545** Type: **Poster**

Comparing SyCL data transfer strategies for tracking use cases

The increased use of accelerators for scientific computing, together with the increased variety of hardware involved, induces a need for performance portability between at least CPUs (which largely dominate WLCG infrastructure) and GPUs (which are quickly emerging as an architecture of choice for online data processing and HPC centers). In the C/C++ community, OpenCL was a low level first step, which is now superseded by SyCL for C++ use cases, as far as Khronos-backed open standards go.

A key factor in GPU performance is the balance between data transfer rates, on-device computational efficiency and utilization by incoming jobs. In this work we will focus on the three SYCL-provided APIs for CPU/GPU data transfers:
1. Accessors and buffers
2. Unified Shared Memory (USM) with implicit transfers
3. USM with explicit transfers

The buffer/accessor API is higher level and somewhat more elegant, thus appealing for newly written application code, but its performance characteristics might depend more strongly on the SyCL implementation in use and its selected backend as the decision of when data should be transferred is not under the developer's control. It is also harder to port existing CUDA codebases to this approach than the other two, and the extent to which it is applicable to utility libraries like vecmem is an open question.

USM with implicit data transfers is conceptually similar to CUDA Unified Virtual Memory, and should provide a similar compromise between performance and ergonomics: the ability to share complex data structures between the CPU and the GPU without "flattening" them into a small number of large GPU buffers is very appealing, but the price to pay for this convenience is expected to be quite high unless the data of interest is heavily reused after the initial data transfer or the CPU/GPU interconnect is fast (as in the case of integrated GPUs and NVidia's upcoming "Grace" platform). In long-lived jobs, this approach should nonetheless be applicable to "long-lived" data such as detector geometry and alignment, providing extra developer convenience while retaining acceptable performance.

USM with explicit data transfers is closest to the traditional CUDA/OpenCL model of explicit CPU/GPU memory copies and is expected to provide optimal performance in the hands of skilled developers, at the cost of maximal code complexity (and more invasive rework in the case of porting existing CPU code).

The aim of this work is to compare the ease of programming and performance portability of these three data transfer strategies, over several hardware targets, including a discrete GPU with dedicated VRAM and an integrated GPU that shares its memory with the CPU.

Besides confronting the ease of programming and the performance portability of the three SYCL data transfer strategies, we want to investigate how those strategies may be mixed depending on the kind of data considered, whether this is rapidly changing event data or more stable data such as detector geometry and alignments.

Significance

While there is a wealth of published work discussing HEP tracking on GPUs, many of these publications are highly exploratory in nature, and we believe this work is somewhat unique in targeting eventual integration into a cross-experiment codebase with a level of maturity and impact similar to that of ACTS.

Furthermore, few publications on usage of GPU in HEP attempt to escape the vendor lock-in of NVidia's proprietary CUDA technology, whose negative consequences include a rapid increase in the price of computeoriented “Tesla” GPU hardware from this manufacturer and the appearance of increasingly dubious software license clauses attempting to prevent the use of less expensive “GeForce” hardware in specific application domains.

Among publications that do evaluate the potential of SyCL specifically (for HEP tracking or otherwise), few discuss the tradeoffs between the various available data transfer strategies, and the extent to which they can be combined, even though this is a key ingredient of the performance/ergonomics tradeoff of GPU programming APIs.

References

“Matrix element method for high performance computing platforms”. G Grasseau, D Chamont, F Beaudette, L Bianchini, O Davignon, L Mastrolorenzo, C Ochando, P Paganini, T Strebler and on behalf of the CMS Collaboration
2015, Journal of Physics: Conference Series, Volume 664, Performance increase and optimization exploiting hardware features. DOI: http://dx.doi.org/10.1088/1742-6596/664/9/092009
Combining Data and Computation Distribution Directives for Hybrid Parallel Programming : A Transformation System.
Rachid Habel, Frédérique Silber-Chaussumier, François Irigoin, Elisabeth Brunet, François Trahay
In International Journal of Parallel Programming (IJPP) (2016)
Automatic OpenCL Code Generation for Multi-device Heterogeneous Architectures.
Pei Li, Elisabeth Brunet, François Trahay, Christian Parrot, Gaël Thomas, Raymond Namyst
In Proceedings of International Conference on Parallel Processing (ICPP) (2015)

Speaker time zone

Compatible with Europe

Primary authors: JOUBE, Sylvain (IJCLab - Télécom SudParis); GRASLAND, Hadrien Benjamin (Université Paris-Saclay (FR)); Dr CHAMONT, David (IJCLab - IN2P3 - CNRS); Mrs ELISABETH, Brunet (Télécom SudParis, Institut Polytechnique de Paris)

Presenter: JOUBE, Sylvain (IJCLab - Télécom SudParis)

Session Classification: Posters: Apple

Track Classification: Track 1: Computing Technology for Physics Research