Brief Performance Portability Analysis of a Matrix Multiplication Kernel on Multiple Vendor GPUs

Manuel Costanzo[1], Enzo Rucci[1]*, Carlos García-Sánchez[2], and Marcelo Naiouf[1]

[1] III-LIDI, Facultad de Informática, UNLP – CIC.
La Plata (1900), Bs As, Argentina
{mcostanzo,erucci,mnaiouf}@lidi.info.unlp.edu.ar
[2] Dpto. Arquitectura de Computadores y Automática, Universidad Complutense de Madrid. Madrid (28040), España
garsanca@dacya.ucm.es

Abstract. The heterogeneous computing paradigm has led to the need for portable and efficient programming solutions that can leverage the capabilities of various hardware devices, such as NVIDIA, Intel, and AMD GPUs. This study evaluates the performance and portability of the SYCL and CUDA languages for a matrix multiplication (MM) application across different GPU architectures. The experimental work showed that, while the CUDA implementation outperforms the SYCL implementation on NVIDIA devices due to optimizations provided by the `nvcc` compiler, the latter implementation demonstrated remarkable code portability to other GPU architectures, such as AMD and Intel. Furthermore, the architectural efficiency percentages obtained on AMD and Intel GPUs showed consistency with the results observed on NVIDIA devices.

Keywords: oneAPI · SYCL · GPU · CUDA· Performance portability

1 Introduction

In the last decade, the quest to improve the energy efficiency of computing systems has fueled the trend toward heterogeneous computing and massively parallel architectures [1]. Nowadays, GPUs can be considered the dominant accelerator, and Nvidia, Intel, and AMD are the biggest manufacturers. In the 4th quarter of 2022, Intel and AMD had 9% of the market, with Nvidia dominating the discrete graphics card market at 82%. By including integrated and embedded graphics, Intel had 71% of the market, Nvidia 17% and AMD 12% [3].

Focusing on the programming aspect, CUDA is the most popular GPU programming language. However, CUDA codes only run on Nvidia GPUs. This fact

* Corresponding author.
[3] https://www.pcgamer.com/intel-is-already-matching-amd-for-gaming-graphics-market-share/

imposes severe limitations to code portability, also affecting maintenance, extension, and development cost. One effort to face this issue is SYCL [4], a new open standard from Khronos Group. SYCL is a royalty-free, cross-platform abstraction layer that allows the programmer to write single-source C++ host code including accelerated code expressed as functors. In this sense, the same SYCL code can run on various hardware platforms, including CPUs, GPUs, and FPGAs. In this way, SYCL seeks to reduce development and maintenance costs and also improve programming productivity.

In this context, while reaching functional portability is already hard, performance portability becomes a major challenge. In this paper, we evaluate the functional and performance portability of two GPU-accelerated implementations of a matrix multiplication (MM) kernel across Intel, Nvidia, and AMD GPUs using Marowka's method [2].

2 Background

2.1 SYCL and the oneAPI Programming Ecosystem

SYCL is a cross-platform programming model based on C++ language for heterogeneous computing and features asynchronous task graphs, hierarchical parallelism, buffers defining location-independent storage, automatic overlapping kernels and communications, and interoperability with OpenCL, among other characteristics [3]. Recently, Intel announced the *oneAPI* programming ecosystem that provides a unified programming model for a wide range of hardware architectures. At the core of the oneAPI environment is the Data-Parallel C++ (DPC++) programming language, which can be summarized as C++ with SYCL. Additionally, DPC++ also features some vendor-provided extensions that might be integrated into these standards in the future [4]. Last, oneAPI provides different programming utilities, including a compatibility tool (`SYCLomatic`) that facilitates the migration to the SYCL-based DPC++ programming language.

2.2 Performance portability

According to Penycook [5], *performance portability* refers to *"A measurement of an application's performance efficiency for a given problem that can be executed correctly on all platforms in a given set"*. These authors define two different performance efficiency metrics: *architectural efficiency* and *application efficiency*. The former represents the ability of an application to utilize hardware efficiently and is a fraction of "peak" theoretical hardware performance; while the latter represents the ability of an application to use the most appropriate implementation and algorithm for each platform, and is a fraction of the best-observed performance.

The metric for performance portability presented by Penycook [5] was later reformulated by Marowka [2] to address some of its flaws. The latter is presented

[4] `https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf`

next. Formally, for a given set of platforms H from the same architecture class, the performance portability $\bar{\Phi}$ of a case-study application α solving problem p is:

$$\Phi(\alpha, \bar{p}, H) = \begin{cases} \frac{\sum_{i \in H} e_i(\alpha,p)}{|H|} & \text{if } i \text{ is supported } \forall i \in H \\ \text{not applicable (NA)} & \text{otherwise} \end{cases}$$

where $e_i(\alpha, p)$ is the performance efficiency of case-study application α solving problem p on the platform i.

3 Experimental Work and Results

3.1 Case-Study Applications: Matrix Multiplication (MM)

Two GPU-accelerated implementations of matrix multiplication (MM) kernel were considered for the performance portability evaluation:

- `CUDA`: this version was extracted from the CUDA Demo Suite [5]. This app computes a MM using shared memory through a tiled approach and loop unrolling technique to increase throughput.
- `SYCL`: this code is based on the implementation provided in [6], which represents an SYCL-equivalent, migrated version of the previous one.

It is important to note that, according to Nvidia, the code "*It has been written for clarity of exposition to illustrate various CUDA programming principles, not with the goal of providing the most performant generic kernel for matrix multiplication.*" [6]

3.2 Experimental Results

The experiments were performed on eight systems equipped with different GPUs. The main features of these systems are described in Table 1. A single workload was configured for MM ($nIter = 10$; $wA, wB, hA, hB = \{16384\}$). Finally, to run SYCL code on Nvidia and AMD GPUs, several modifications had to be made to the build, as it is not supported by default [7]. After these modifications, it was possible to run DPC++ code on an Nvidia GPU, but using the Clang++ compiler (`nvcc 11.7`, `clang 16.0`).

Table 2 presents the GFLOP/s and architectural efficiencies of both CUDA and SYCL codes on the experimental platforms. On the one hand, it becomes evident that CUDA outperforms SYCL using all Nvidia GPUs. In particular, CUDA version runs (on average) 1.2× faster than SYCL. This superior performance can be primarily attributed to the fact that `nvcc`, performs a more efficient

[5] `https://docs.Nvidia.com/cuda/cuda-samples/`
[6] `https://docs.Nvidia.com/cuda/cuda-c-programming-guide/index.html`
[7] https://intel.github.io/llvm-docs/GetStartedGuide.html

Table 1: Experimental platforms

CPU			GPU	
Processor	**RAM Memory**	**Vendor**	**Model (Architecture)**	**GFLOPS peak (SP)**
Intel Xeon E5-2695	16 GB		GTX 980 (Maxwell)	4980
Intel Xeon E5-2695	16 GB		GTX 1080 (Pascal)	8872
Intel Core i5-7400	64 GB	Nvidia	RTX 2070 (Turing)	7464
Intel Core i5-10400F	64 GB		RTX 3070 (Ampere)	20313
Intel Xeon Gold 6138	64 GB		Tesla V100 (Volta)	14131
Intel Core i9-9900K	32 GB	Intel	Arc 770 (Gen 12.5)	19660
Intel Xeon iE5-1620	64 GB	AMD	RX 6700 XT (RDNA 2.0)	13214

Table 2: GFLOP/s and architectural efficiencies of both CUDA and SYCL codes on the experimental platforms.

Platform		CUDA		SYCL	
GPU	**GFLOP/s peak**	**GFLOP/s achieved**	**Arch. eff.**	**GFLOP/s achieved**	**Arch. eff.**
GTX 980	4980	552	11.1%	430	8.6%
GTX 1080	8872	603	6.8%	556	6.3%
RTX 2070	7464	1011	13.6%	810	10.9%
RTX 3070	20313	1316	6.5%	1084	5.3%
Tesla V100	14131	1582	11.2%	1345	9.5%
Arc 770	19660	×	NA	1836	9.3%
RX 6700 XT	13214	×	NA	1553	11.8%

code translation than `clang++` when it comes to shared memory accesses [8], causing SYCL code to use more registers and perform additional computation. On the other hand, architectural efficiencies are low for both code versions (8% on average). This behavior is related to the educative aspect of the original code that was detailed in Section 3.1.

Performance portability of both CUDA and SYCL codes is presented in Table 3. First, it becomes evident that SYCL code provides higher functional portability, successfully running on different hardware vendor platforms. Moreover, CUDA fails to demonstrate the same level of adaptability, just being able to run on Nvidia GPUs. Second, both codes present a similar performance efficiency when executing on the different supported GPUs, demonstrating their performance portability.

[8] `https://support.codeplay.com/t/poor-performance-on-matrix-multiplication/575/2?u=mcostanzo`

Table 3: Performance portability of both CUDA and SYCL codes on the experimental platforms.

	$\Phi(\alpha, p, H)$	
Platform set (H)	**CUDA**	**SYCL**
Nvidia	9.8%	8.1%
Intel	NA	9.3%
AMD	NA	11.8%
Nvidia ∪ AMD	NA	8.7%
Nvidia ∪ Intel	NA	8.3%
Intel ∪ AMD	NA	10.5%
Nvidia ∪ AMD ∪ Intel	NA	8.8%

3.3 Discussion

While SYCL code proved to be slower than its CUDA counterpart in this study, it showcased performance portability across a wider range of GPU vendors, highlighting its versatility and potential. However, it is important to note that the observed performance difference between SYCL and CUDA codes does not occur in all cases; [7, 8, 9] show that SYCL codes can achieve the same or even better performance than CUDA versions.

4 Conclusions and Future Work

In this paper, we have evaluated both the performance and portability of SYCL and CUDA languages for a MM application on Nvidia, Intel, and AMD GPUs. The main findings of this study can be summarized as follows:

- The performance comparison between the SYCL and CUDA implementations on Nvidia devices revealed that the latter outperforms the former due to the optimizations applied by the `nvcc` compiler.
- We have successfully demonstrated the code portability of the SYCL implementation to other GPU architectures, such as AMD and Intel. Moreover, the architectural efficiency percentages obtained on these GPUs were found to be consistent with those observed on Nvidia devices.

In summary, this brief study highlights the potential of SYCL as a performance-portable alternative to CUDA for the development of high-performance computing applications. Although the current performance of SYCL on Nvidia GPUs may be lower than that of CUDA, this gap will decrease as SYCL compilers continue to improve.

Future work focuses on exploring the use of SYCL in different application domains. This could provide valuable insights into its performance and portability features in a broader context, enabling a more comprehensive understanding of its strengths and limitations.

References

[1] H. Giefers et al. "Analyzing the energy-efficiency of sparse matrix multiplication on heterogeneous systems: A comparative study of GPU, Xeon Phi and FPGA". In: *2016 IEEE ISPASS*. 2016, pp. 46–56.

[2] Ami Marowka. "Reformulation of the performance portability metric". In: *Software: Practice and Experience* 52.1 (2022), pp. 154–171. DOI: https://doi.org/10.1002/spe.3002.

[3] Ronan Keryell and Lin-Ya Yu. "Early Experiments Using SYCL Single-Source Modern C++ on Xilinx FPGA". In: *Proceedings of the IWOCL '18*. Oxford, UK: ACM, 2018. DOI: 10.1145/3204919.3204937.

[4] S. Christgau and T. Steinke. "Porting a Legacy CUDA Stencil Code to oneAPI". In: *2020 IEEE IPDPSW*. May 2020, pp. 359–367. DOI: 10.1109/IPDPSW50202.2020.00070.

[5] S.J. Pennycook, J.D. Sewall, and V.W. Lee. "Implications of a metric for performance portability". In: *Future Generation Computer Systems* 92 (2019), pp. 947–958. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.08.007.

[6] Manuel Costanzo et al. "Early Experiences Migrating CUDA codes to oneAPI". In: *IX Jornadas de Cloud Computing, Big Data & Emerging Topics*. 2021.

[7] Manuel Costanzo et al. "Migrating CUDA to oneAPI: A Smith-Waterman Case Study". In: *Bioinformatics and Biomedical Engineering*. Cham: Springer International Publishing, 2022, pp. 103–116. ISBN: 978-3-031-07802-6. DOI: 10.1007/978-3-031-07802-6_9.

[8] Zheming Jin and Jeffrey S. Vetter. "Performance Portability Study of Epistasis Detection Using SYCL on NVIDIA GPU". In: *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. BCB '22. Northbrook, Illinois: ACM, 2022. ISBN: 9781450393867. DOI: 10.1145/3535508.3545591.

[9] Goutham Kalikrishna Reddy Kuncham, Rahul Vaidya, and Mahesh Barve. "Performance Study of GPU applications using SYCL and CUDA on Tesla V100 GPU". In: *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. 2021, pp. 1–7. DOI: 10.1109/HPEC49654.2021.9622813.