ParCFD2024
35^{th} International Conference on Parallel Computational Fluid Dynamics
Sep 02-04 2024, Bonn, Germany

ASSESSING COMPUTATIONAL FLUID DYNAMICS ON GPU USING PORTABLE LANGUAGES

Youssef Faqir-Rhazoui*, Carlos García†

*Eviden, R&D Department, Spain
e-mail: youssef.el@eviden.com

†Complutense University of Madrid, Spain
e-mail: garsanca@ucm.es

Key words: CFD, GPU, OpenMP, SYCL, CUDA, HIP

Abstract. Accelerators are essential for achieving optimal performance and energy efficiency in computing. However, market segmentation often leads to language lock-ins, limiting flexibility across accelerators and increasing development costs. To address this, alternatives like SYCL or OpenMP have emerged, enabling code portability across diverse hardware platforms.

The study reveals that while both OpenMP and SYCL demonstrate comparable performance to native languages on NVIDIA and Intel GPUs, SYCL significantly outperforms OpenMP on AMD platforms. These findings underscore the potential of multi-device and open languages in enhancing performance and reducing development overhead in parallel CFD simulations.

1 Introduction

Hardware acceleration is a crucial component in the quest for enhanced performance and energy efficiency. However, due to market segmentation, many accelerators (e.g., GPUs) suffer from language lock-ins (e.g., CUDA, HIP), restricting the use of the same language across multiple vendor accelerators [1].

redMoreover, CI/CD pipelines would be compromised by these practices, as maintaining multiple pipelines for each architecture would be error-prone and increase the complexity of the process. CD would require maintaining multiple implementations of the same algorithm in different languages. CI would also be tied to each language, as each language is tied to its own compiler [2].

To address these issues, various alternatives have emerged, such as OpenCL, SYCL, or OpenMP. These languages have the capability to run on multi-vendor accelerators such as CPUs or GPUs while using the same code.

On the other hand, Computational Fluid Dynamics (CFD) is a powerful tool for simulating complex environments such as turbomachinery, aerodynamics, or heat transfer. CFD simulations were traditionally performed on CPUs and distributed across clusters using MPI. Due to the inherently parallel nature of CFD, GPU acceleration is key to

leveraging performance and energy efficiency. redHowever, some areas of CFD, such as particle simulation, still remain mostly implemented on the CPU [3].

This paper presents a suite of benchmarks for testing CFDs on NVIDIA, AMD, and Intel GPUs. We compare the native language of each platform with the portable languages OpenMP and SYCL.

The following paper is structured as follows: In Section 2, we present the environmental conditions and methods used in the experiment. Section 3 describes the results obtained redand discusses them.

2 Methods

The experiments were conducted using the GPUs listed in Table 2. The selected GPUs cover NVIDIA, AMD and Intel architectures.

While the table specifies the driver used for each GPU, it is worth mentioning that the V100 employs the CUDA 12.4 toolkit, the RX 6700XT is powered by ROCm 5.4.3, and the Max 1100 is integrated with oneAPI 2024.1.

Transitioning to portable languages (OpenMP and SYCL), it is worth distinguishing each one. While OpenMP is supported in all the previously mentioned toolkits. SYCL is supported by the oneAPI's compiler and is suitable for NVIDIA and AMD GPUs.

	NVIDIA Tesla V100	**AMD RX 6700 XT**	**Intel Max 1100**
Frecuency (GHz)	Up to 1.38	Up to 2.58	Up to 1.55
Cores	80 SM	40 CU	56 Xe cores
Perf. (FP64)	7.06 TFLOPS	0.825 TFLOPS	22.22 TFLOPS
Driver	550.54.14	5.18.3	23.52 (ocl) 1.3 (level0)

Table 1: Specifications of the GPU used in the experimentation.

Regarding the benchmarks, we employed a set of eleven CFDs combined with common CFD equations. For the sake of space, we are not providing detailed descriptions and parameters of the benchmarks here. Instead, we strongly recommend that readers refer to the repository where this information is provided.[1]

redConcerning the benchmark implementation of each language and how we keep them as similar as possible, the baseline implementation was CUDA. Translating the code to HIP was achieved using the HIPIFY tool.[2] Since OpenMP syntax is quite different from the other employed languages and there is no official tool to port it, the port was done manually to be as similar as possible to the CUDA code. Once the OpenMP code was delivered, it was compiled for each architecture using the vendor compiler for that architecture. Finally, SYCL code was mainly ported using the SYCLomatic tool.[3] In this case, SYCL code was always compiled for all architectures using the oneAPI compiler.

[1]The repository used can be found at: `https://github.com/A924404/cfd-bench`

[2]`https://github.com/ROCm/HIP?tab=readme-ov-file`

[3]`https://github.com/oneapi-src/SYCLomatic`

3 Experimental Results and Results Discussion

3.1 Results

Reviewing Figure 1 for the Tesla V100, CUDA serves as the native benchmark. redSYCL fails to run the *miniWeather* benchmark due to an issue related to its interaction with MPI. In the case of OpenMP, the *d3q19-bgk* test was not implemented in the original suite.

SYCL averaged 91% of CUDA performance, while OpenMP reached 80%. Depending on the benchmark, these percentages may vary.

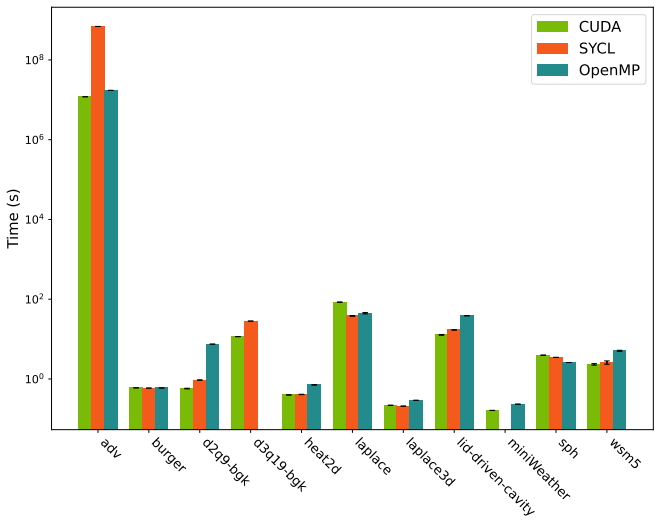

Figure 1: NVIDIA Tesla V100 performance comparison across CUDA, OpenMP, and SYCL.

Figure 2 shows results from the AMD 6700 XT. The native implementation for AMD GPU is HIP. While both HIP and SYCL ran all benchmarks, OpenMP failed the *lid-driven-cavity* testred, the main issue found relies on memory allocation. While HIP and SYCL are able to allocate the amount of memory required by the benchmark, OpenMP do not.

The results show poor performance on AMD architecture. OpenMP achieves only 51% of HIP times on average, while SYCL reaches up to 66% of native performance. The standard deviation (SD) for OpenMP is 44% and for SYCL it is 35%. SYCL's performance on AMD is primarily affected by the *adv* and *miniWeather* tests, but removing them increases performance to 81% with an SD of 19%. OpenMP's issue is spread across all benchmarks.

Finally, the Intel Max 1100 results are shown in Figure 3. SYCL runs on two backends: Level0 and OpenCL, both maintained by Intel with no notable differences expected.

redIn this instance, all three implementations failed to execute the *miniWeather* mini-app due to MPI call incompatibilities in the system, unrelated to the languages used, but rather due to the absence of an MPI installation. Additionally, the authors did not have

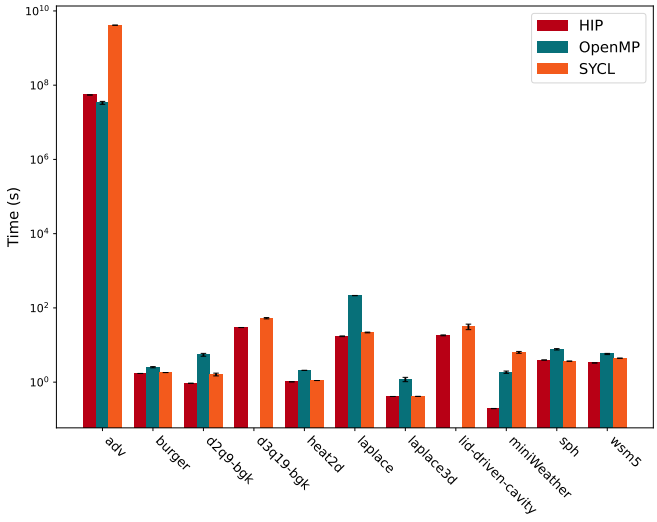

Figure 2: AMD RX 6700 XT performance comparison across HIP, OpenMP, and SYCL.

root access to the system employed for Intel GPUs.

Regarding the numbers and SYCL, both backends are virtually equivalent in performance, achieving an average speedup of $\times 1$. Transitioning to OpenMP, it shows a slight speedup over the SYCL equivalents, averaging $\times 1.05$.

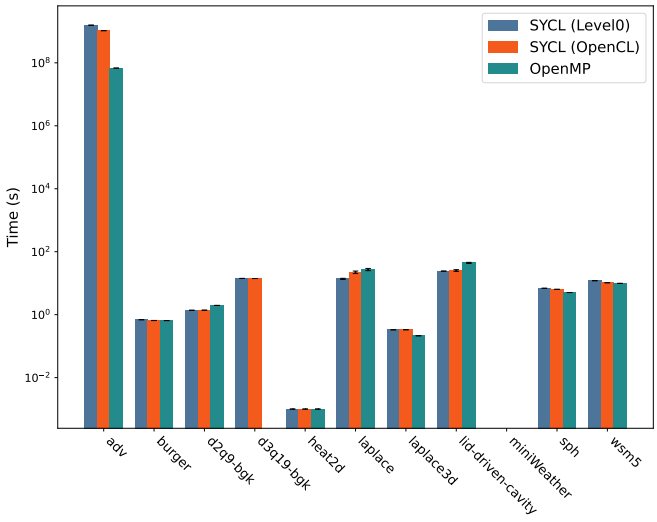

Figure 3: Intel Max 1100 performance comparison across SYCL(Level0), SYCL(OpenCL), and OpenMP.

3.2 Discussion

redOpen and portable languages such as SYCL and OpenMP are promising in scenarios where changing GPUs necessitates completely altering the underlying source code and CI/CD pipelines due to vendor lock-ins.

redIn performance results, SYCL has shown performance comparable to NVIDIA's (10% difference). However, when moving to AMD GPUs, the performance is degraded due to the immaturity of the language on this platform. SYCL is also a native language for Intel GPUs, allowing for highly optimized performance.

redRegarding OpenMP, we found mixed results. On Intel platforms, there is no performance loss, while on NVIDIA, the gap is approximately 20%, and on AMD, it is around 50%.

redBoth SYCL and OpenMP are ultimately translated into native assembly code that the GPU executes. Therefore, the efficiency of translating high-level code to low-level code is crucial for achieving performance. Table 3.2 shows the number of assembly lines into which the target languages are translated.[4] Generally, more lines imply more time for the GPU to run them. For CUDA PTX, SYCL employs 27% more code, while OpenMP uses 77% more. For Amdgcn, SYCL uses 45% more assembly code, and OpenMP uses up to 82% more.

	CUDA	**HIP**	**SYCL**	**OpenMP**
Cuda ptx	33,455	-	46,259	144,999
Amdgcn	-	23,757	43,321	132,692

Table 2: redTotal assembly code lines by language and architecture.

Acknowledgment

This paper is part of the HIDALGO2 project, co-funded by the European Union under grant agreement number: 101093457.

REFERENCES

[1] M. Breyer and et. al., "A comparison of sycl, opencl, cuda, and openmp for massively parallel support vector machine classification on multi-vendor hardware," in *Proceedings of the 10th International Workshop on OpenCL*, IWOCL '22, (New York, NY, USA), Association for Computing Machinery, 2022.

[2] P. Rostami Mazrae, T. Mens, M. Golzadeh, and A. Decan, "On the usage, co-usage and migration of ci/cd tools: A qualitative analysis," *Empirical Software Engineering*, vol. 28, no. 2, p. 52, 2023.

[3] A. Zhu, Q. Chang, J. Xu, and W. Ge, "A dynamic load balancing algorithm for cfd–dem simulation with cpu–gpu heterogeneous computing," *Powder Technology*, vol. 428, p. 118782, 2023.

[4]We did not include the same information for the Intel GPU because we could not generate the assembly codes. The required tool needs to be installed with root access, which we do not have.