Analyzing the Performance Portability of SYCL across CPUs, GPUs, and Hybrid Systems with SW Sequence Alignment

Manuel Costanzo[a], Enzo Rucci[a], Carlos García-Sánchez[b], Marcelo Naiouf[a], Manuel Prieto-Matías[b]

[a]*III-LIDI, Facultad de Informática, Universidad La Plata (Argentina)*
[b]*Fac. Informática, Universidad Complutense Madrid (Spain)*

Abstract

The high-performance computing (HPC) landscape is undergoing rapid transformation, with an increasing emphasis on energy-efficient and heterogeneous computing environments. This comprehensive study extends our previous research on SYCL's performance portability by evaluating its effectiveness across a broader spectrum of computing architectures, including CPUs, GPUs, and hybrid CPU-GPU configurations from NVIDIA, Intel, and AMD. Our analysis covers single-GPU, multi-GPU, single-CPU, and CPU-GPU hybrid setups, using two common, bioinformatic applications as a case study. The results demonstrate SYCL's versatility across different architectures, maintaining comparable performance to CUDA on NVIDIA GPUs while achieving similar architectural efficiency rates on AMD and Intel GPUs in the majority of cases tested. SYCL also demonstrated remarkable versatility and effectiveness across CPUs from various manufacturers, including the latest hybrid architectures from Intel. Although SYCL showed excellent functional portability in hybrid CPU-GPU configurations, performance varied significantly based on specific hardware combinations. Some performance limitations were identified in multi-GPU and CPU-GPU configurations, primarily attributed to workload distribution strategies rather than SYCL-specific constraints. These findings position SYCL as a promising unified programming model for heterogeneous computing environments, particularly for bioinformatic applications.

Keywords: Julia, Rodinia, Portability, CUDA, GPU

1. Introduction

The parallel processing landscape is undergoing rapid transformation, with an increasing emphasis on energy-efficient and heterogeneous computing environments [1]. GPUs are now the dominant accelerator in this field, offering incomparable performance for parallel workloads. Still, the roles of CPUs and the potential of hybrid CPU-GPU systems are also actively explored in computational research. On the one hand, GPU-based Systems on Chip (SoC) are becoming more popular every day. On the other hand, hybridization now extends to CPU architectures, with the emergence of heterogeneous designs like Intel's Raptor Lake and Meteor Lake. Depending on the workload, these processors incorporate up to three different types of cores on the same die to optimize performance and energy efficiency. This architectural heterogeneity at the system and processor levels reflects the larger market dynamics and the ongoing evolution of computing architectures.

As of 2024, NVIDIA maintains its dominance in the GPU market, particularly in the discrete GPU (dGPU) segment, where it possesses an 88% market share. AMD has demonstrated significant progress, capturing 12% of the market, while Intel experienced a decline, with its market share dropping to less than 1% [2]. In the integrated graphics (iGPU) sector, Intel retains a substantial advantage with a 68% market share, followed by NVIDIA (18%) and AMD (14%). This market structure highlights the importance of considering diverse architectures in GPU-based computing research [3].

Concerning parallel programming, the proliferation of diverse computing architectures (including CPUs, GPUs, and hybrid CPU-GPU systems), has created a pressing need for unified programming models that can effectively harness their collective potential. In this context, SYCL has emerged as a promising solution that offers a cross-platform abstraction layer that facilitates code portability across various hardware architectures. By enabling developers to write high-performance code that can be seamlessly executed on a wide range of devices, SYCL plays a crucial role in addressing the challenge of performance portability in heterogeneous computing environments [4].

This article evolves from our systematic exploration of SYCL's capabilities in bioinformatics applications across heterogeneous computing platforms. In our initial work [5], we focused on migrating a CUDA-based, protein database search software (SW#db) to SYCL using Intel's oneAPI ecosystem. This study demonstrated the tool's effectiveness for code migration and it defined the basic cross-GPU-architecture portability of the migrated code, achieving comparable or slightly better efficiency rates compared to the native CUDA implementation. Building upon these findings, in [6] we expanded the study to evaluate SYCL's performance portability across different GPU architectures, including single and multi-GPU

*Corresponding author
Email address: erucci@lidi.info.unlp.edu.ar (Manuel Prieto-Matías)

configurations from different vendors. This research revealed that while CUDA and SYCL versions achieved similar performance on NVIDIA devices, SYCL reached higher architectural efficiency rates in most test cases from the other GPU vendors. Finally, in [7] we completed the SYCL-migration process of SW# biological suite to support pairwise alignment and different alignment algorithms. Moreover, its performance in a wide variety of biological test scenarios was analyzed and cross-SYCL-implementation portability was verified on several GPUs and some CPUs from a functional perspective.

This work extends our performance portability study to include pairwise sequence alignment and (pure and hybrid) CPUs and CPU-GPU systems. The key contributions of this paper are:

- An extension of our previously adapted performance model from [8], to encompass pure and hybrid CPU architectures and also combined CPU-GPU systems.
- An extension of our previous work on the functional and performance portability of *SW#* applications across different GPU architectures to now include pairwise sequence alignment and Intel and AMD CPUs, as well as hybrid CPU-GPU configurations. On the one side, by investigating the performance characteristics of SYCL on these diverse platforms, this study aims to provide valuable insights into its potential as a unified programming model for the increasingly complex landscape of modern computing. On the other side, by expanding the study to include CPUs and hybrid systems, this research seeks to offer a more comprehensive understanding of how SYCL-based applications perform across the full spectrum of available computational resources.
- An analysis of the trade-offs and synergies between CPU and GPU computations in hybrid configurations for two common bioinformatic workloads.

The remainder of this paper is structured as follows: Section 2 presents the background; Section 3 details the case study applications and the expanded performance model; Section 4 presents our findings on functional and performance portability; Section 5 discusses related work; and Section 6 presents the conclusions and future research directions.

2. Background

2.1. CPUs, GPUs, and Programming Models for Heterogeneous Computing

The evolution of high-performance computing (HPC) has been significantly influenced by the emergence of general-purpose GPU programming. In 2007, NVIDIA introduced CUDA [9] alongside the Tesla GPU architecture, enabling non-graphics applications to harness GPU computational power. Although CUDA quickly became a prominent low-level programming model for GPU computing, its proprietary nature and exclusive compatibility with NVIDIA hardware led to the development of more versatile alternatives, such as OpenCL [10]. OpenCL offers a level of abstraction comparable to that of CUDA while supporting multiple devices and vendors.

The field has also expanded to include multicore CPUs and hybrid GPU-CPU systems, accompanied by the development of high-level programming initiatives such as OpenMP [11], OpenACC [12, 13], and SYCL [4]. Although CUDA is limited to NVIDIA hardware, and OpenCL provides broad coverage but is challenging to learn and costly to implement, SYCL has emerged as a promising solution for heterogeneous computing.

SYCL [4] enables developers to write code for various processors, including CPUs, GPUs, and hybrid systems, using standard ISO C++. It incorporates host and device code in a single source file and supports multiple acceleration APIs, such as OpenCL, facilitating seamless integration with lower-level code across different hardware architectures.

Several SYCL implementations are now available, with Intel's oneAPI [14] standing out as a mature developer suite. OneAPI eliminates the need for separate code bases for host and device, which is required in OpenCL, and supports multiple programming languages and tools for different architectures. Data Parallel C++ (DPC++), oneAPI's core language [14], integrates the SYCL and OpenCL standards without additional extensions, facilitating the development of applications that can efficiently utilize the full spectrum of available computational resources.

2.2. Smith-Waterman Sequence Alignment

A key process in bioinformatics and computational biology is sequence alignment, which aims to identify regions of similarity between sequences to uncover their structural, functional, and evolutionary relationships [15]. The Smith-Waterman (SW) algorithm is commonly employed to determine the optimal local alignment between two sequences [16]. This technique relies on dynamic programming and is highly sensitive, as it examines all possible alignments between the sequences.

Given two sequences Q and D of length $|Q| = m$ and $|D| = n$, the recurrence relations for the SW algorithm with the modification of Gotoh [17] are defined as follows:

$$H_{i,j} = max \begin{cases} 0 \\ H_{i-1,j-1} + SM(Q[i], D[j]) \\ E_{i,j} \\ F_{i,j} \end{cases} \quad (1)$$

$$E_{i,j} = max \begin{cases} H_{i,j-1} - G_o \\ E_{i,j-1} - G_e \end{cases} \quad (2)$$

$$F_{i,j} = max \begin{cases} H_{i-1,j} - G_o \\ F_{i-1,j} - G_e \end{cases} \quad (3)$$

The similarity score $H_{i,j}$ is calculated to identify a common subsequence; $H_{i,j}$ contains the score to align the prefixes $Q[1..i]$ and $D[1..j]$. Moreover, $E_{i,j}$ and $F_{i,j}$ correspond to the scores of prefix $Q[1..i]$ and $D[1..j]$ aligned to a gap, respectively. *SM* denotes the *scoring matrix* and defines the match/mismatch scores

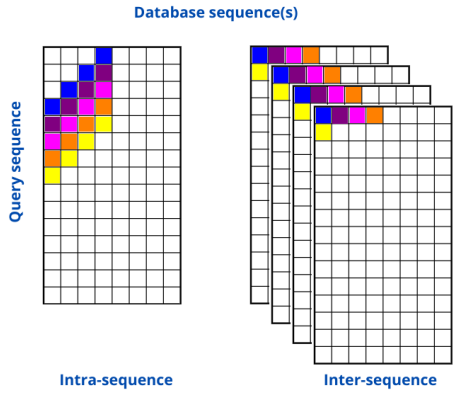

Figure 1: Parallelization approaches in similarity matrix computations (adapted from [18]). Each color indicates the cells that can be computed together in a SIMD manner.

between the residues. Lastly, G_o and G_e refer to the gap open and gap extension penalties, respectively.

Firstly, H, E and F must be initialized with 0 when $i = 0$ or $j = 0$. Then, the recurrences should be calculated with $1 \leq i \leq m$ and $1 \leq j \leq n$. The highest value in the matrix H (S) corresponds to the optimal local alignment score between $Q[1..i]$ and $D[1..j]$. If required, the optimal local alignment is finally obtained by following a traceback procedure whose starting point is S. From a computational perspective, it is important to highlight the dependencies of any H element. Any cell can be calculated only after the values of the upper, left, and upper-left neighbors are known; imposing restrictions on how H can be processed.

SW is usually employed to compute: (a) pairwise alignments of two very long DNA sequences, and (b) database similarity search (one-to-many), involving shorter protein sequences. As SW runs in quadratic time to compute optimal alignment, widely adopted parallelization approaches are *intra-sequence* (a single matrix is calculated and all Processing Elements (PEs) work collaboratively) and *inter-sequence* (simultaneously calculating multiple matrices without communication between the PEs). Fig. 1 illustrates both approaches.

2.3. Performance portability

According to Penycook et al. [19], *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 denotes the capacity of an application to effectively utilize hardware resources, measured as a proportion of the theoretical peak performance. The latter signifies the application's ability to select the most suitable implementation for each platform, representing a fraction of the highest observed performance achieved.

The performance portability metric presented by Penycook et al. [19] was later reformulated by Marowka [20] to address some of its flaws. Formally, for a given set of platforms H from the same architecture class, the portability $\bar{\mathbf{\Phi}}$ of a case study application α to solve the problem p is:

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

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

The *performance portability* concept emphasizes the capability to write code that can efficiently utilize the available computing resources, such as CPUs, GPUs, or specialized accelerators, while maintaining high performance regardless of the specific hardware configuration. With performance portability, developers can write code once and have it deliver optimal performance on various target platforms. This eliminates the need for extensive manual code optimizations or platform-specific modifications, reducing development time and effort.

3. Case-Study Applications and Performance Model

3.1. Case-Study Applications

Two GPU-accelerated implementations of SW sequence alignment were considered for the performance portability evaluation:

- `CUDA`: The *SW#* framework represents a CUDA-implemented solution for biological sequence alignment that prioritizes memory efficiency. Operating as both a standalone program and an integrated library, it enables protein and DNA sequence comparisons through pairwise alignment and database searches. The framework provides customizable alignment parameters, including Smith-Waterman algorithm options, gap penalty specifications, and scoring matrix selection. Through CPU-GPU coordination, *SW#* achieves good performance by implementing a dynamic load balancing system that assigns tasks based on sequence length characteristics. The parallelization architecture primarily leverages GPU capabilities through both inter-task and intra-task approaches. GPU processing follows a dual-kernel strategy: shorter database sequences are processed via inter-task parallelism in a *short kernel*, while longer sequences utilize intra-task parallelism in a *long kernel*. For multi-GPU configurations, *SW#* employs an adaptive execution protocol. When the GPU count exceeds query sequence numbers, all devices synchronously process different database segments for a single query. Alternatively, with more queries than GPUs, each GPU independently processes distinct queries against the complete database [21, 22].

- `SYCL`: this code is based on the implementation presented in the paper [7], representing a SYCL equivalent. The migration of the *SW#* suite was performed using `dpct` (the Data Parallel Compatibility Tool available in the oneAPI suite) and some hand-coding modifications. The resulting implementation is pure SYCL

code without device-specific optimizations or native code paths, ensuring that all computations—whether on GPUs or CPUs—are executed through SYCL. This approach maintains complete portability across different architectures while allowing for a fair comparison of SYCL's performance in various computing environments.

3.2. Performance Model

Peak theoretical hardware performance must be estimated for all selected devices in this study to compute the performance portability metric. This step requires considering both hardware and algorithm features. In our previous work [6], we have adapted the performance model from [8] to the *SW#* algorithm and several GPUs. For the present investigation, we extend this adapted model to encompass CPUs from various vendors. In fact, the presented equations can serve as a generic framework for estimating the theoretical peak performance of multiple devices, including CPUs, GPUs, and even FPGAs. The computing capability of different devices can be estimated using Eq. 4:

$$Capability = Clock\_Rate \times Throughput \times Lanes \quad (4)$$

where *Clock_rate* denotes the device's clock frequency, *Throughput* represents the number of instructions the device can execute per clock cycle, and *Lanes* indicates the quantity of SIMD vector lanes. Subsequently, it is imperative to enumerate the instructions issued during each cell update of the similarity matrix. In the sequence alignment context, the predominant performance metric is Cell Updates Per Second (CUPS), representing the time for a complete computation of one cell in the similarity matrix (H). The GCUPS (billion CUPS) value is calculated by:

$$GCUPS = \frac{Q \times D}{t \times 10^9} \quad (5)$$

where $|Q|$ is the total number of residues in the query sequence, $|D|$ is the total number of residues in the database and t is the runtime in seconds [23]. Consequently, the theoretical peak GCUPS of any device can be modeled using Equation 6:

$$Theo\_peak = \frac{Capability}{Instruction\_count\_one\_cell\_update} \quad (6)$$

Table 1 provides a comprehensive summary of the theoretical peak performance for selected GPUs, calculated using Equation 6. Additionally, the performance metrics for CPUs are elucidated in Table 2. Further detailed analysis and discussion are presented in the subsequent sections of this paper.

3.2.1. SW# core instructions

SW# computes the similarity matrix using 32-bit integers and performs 12 instructions per cell update. Algorithm 1 presents the snippet of cell update in similarity matrix as in Eq. 1, Eq. 2, and Eq. 3. Just adding, subtracting, and maximum instructions are required to perform a single-cell update.

Algorithm 1 Core instructions per cell update in similarity matrix

```
1:  E^1 = E_l − G_e          ▷ E_l: E of its left neighbor
2:  E^2 = H_l − G_o          ▷ H_l: H of its left neighbor
3:  E = max(E^1, E^2)
4:  F^1 = F_u − G_e          ▷ F_u: F of its upper neighbor
5:  F^2 = H_u − G_o          ▷ H_u: H of its upper neighbor
6:  F = max(F^1, F^2)
7:  H = H_ul + S M           ▷ H_ul: H of its upper-left neighbor
8:  H = max(H, E)
9:  H = max(H, F)
10: H = max(H, 0)
11: A = H                    ▷ A: an auxiliary variable
12: S = max(H, S)            ▷ S: optimal score
```

3.2.2. Architectural features on NVIDIA GPUs

The # Cores in an NVIDIA GPU refers to the number of Streaming Multiprocessors. CUDA does not strictly follow a SIMD execution model but it adopts a similar one denoted as the SIMT model. A *warp* comprises a group of 32 threads that execute the same instruction stream. According to [8], "a *warp* in SIMT is equivalent to a *vector* in SIMD, and a *thread* in SIMT is equivalent to a *vector lane* in SIMD". The instruction throughput depends on the CUDA Compute Capability (CC) of each NVIDIA GPU [1].

3.2.3. Architectural features on AMD GPUs

In the RDNA2.0 architecture, the # Cores parameter corresponds to the number of Compute Units (CUs), which are organized in pairs to form Workgroup Processors (WPs). AMD employs the terms *wavefront* and *work-item* as analogs to NVIDIA's *warp* and *thread*, respectively. While RDNA2.0 supports wavefront sizes of both 32 and 64 work items, the former has priority over the latter. Each CU encompasses two SIMD32 vector units, enabling the execution of 64 add/subtract/max instructions per cycle (`Int32`). Consequently, the instruction throughput per work item is quantified as 2.

3.2.4. Architectural features on Intel GPUs

In the dGPU segment, Intel's GPU design philosophy diverges significantly from that of NVIDIA and AMD. The fundamental building block of the Intel Xe microarchitecture is the Xe Core, each comprising 16 Xe Vector Engines (XVEs) [2], capable of executing 8 add/subtract/max instructions per cycle (`Int32`). In the context of the proposed model, Xe Cores and XVEs correspond to # Cores and # Lanes, respectively.

In the iGPU segment, both Gen9 and Gen12 microarchitectures exhibit similar design principles, primarily differentiated by the number of computational resources. These microarchitectures utilize the Subslice as their fundamental block, each containing 8 Execution Units (EUs) that can process 8 add/subtract/max instructions per cycle (`Int32`). Within the framework

[1] `https://docs.nvidia.com/cuda/cuda-c-programming-guide/#maximize-instruction-throughput`

[2] Also referred to as Execution Units (EUs)

of the proposed model, Subslices and EUs are analogous to # Cores and # Lanes, respectively.

3.2.5. Architectural features on CPUs

For Intel and AMD CPUs, # Cores represents the number of processor cores, each capable of independently executing instructions, which is crucial for performance in parallel applications. The # Lanes is defined by the number of Vector Processing Units (VPUs) and their supported vector width/instruction set (such as SSE, AVX, AVX-512), which allows a core to perform multiple operations simultaneously. The instruction throughput depends on the operation that is being performed.

Since intensive use of VPUs leads to increased heat dissipation and power consumption, modern processors incorporate various technologies that reduce the clock frequency to counteract their impact [24]. For this reason, the operating frequency when all cores perform intensive computation is usually the base frequency or a slightly higher one, particularly on the server segment. For this work, we have monitored the *SW#* execution on each CPU device to verify the information provided.

Finally, a caveat must be made for hybrid CPU architectures (such as Intel's Alder Lake) [25]. As these architectures present up to three types of cores (P-cores, E-cores, and LP E-cores), the maximum theoretical performance is obtained by summing up the partial maximum performances.

4. Experimental Results

4.1. Experimental Design

The experiments were carried out on 12 GPUs, including 6 NVIDIA dGPUs, 1 AMD dGPU, 1 AMD iGPU, 3 Intel iGPUs, and 1 Intel dGPU. In addition, 8 Intel CPUs (from different segments) and 1 AMD CPU were included in the study. The specific details of these platforms can be found in Table 1. The oneAPI and CUDA versions used were `2022.1.0` and `11.7`, respectively. For both CUDA and SYCL, the optimization flag `-O3` was used during compilation. To run SYCL code on NVIDIA and AMD GPUs, several modifications had to be made to the build process, as SYCL is not supported by default on these platforms[3] but Codeplay recently announced free binary plugins[4] to support it. After these modifications, it was possible to run DPC++ code on both NVIDIA and AMD GPUs using the Clang++ compiler (`16.0`).

To ensure that the CPU executes only SYCL code and avoid the use of SW# "native" CPU code, the flag `T=0` was used in all tests. This setting forces all sequence alignments to be thoroughly computed using SYCL, allowing for a fair evaluation of SYCL performance on the CPU.

Performance evaluation was carried out using real biological data for the two application benchmarks:

- Protein database search: searching 20 query protein sequences against the well-known Environmental Non-Redundant database (Env. NR) (`2021_04 Release`), which contains 995210546 amino acid residues in 4789355 sequences, with a maximum length of 16925. Query sequences were selected from the Swiss-Prot database [5], with lengths ranging from 144 to 5478. The access numbers for these queries are: P02232, P05013, P14942, P07327, P01008, P03435, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8, P20930, P08519, Q7TMA5, P33450, and Q9UKN1. Besides, *SW#* was configured with `BLOSUM62` as the substitution matrix, and 10/2 as the penalty for insertion/extension.
- Pairwise alignment: computing five different DNA alignments. Table 3 presents the accession numbers and sizes of the DNA sequences used. The score parameters were configured as +1 for match, -3 for mismatch, -5 for gap open, and -2 for gap extension.

Last, each test was executed 20 times and the performance was determined based on the average of these multiple runs (to minimize deviations).

4.2. Single-GPU Performance and Portability Results

A comprehensive analysis of SYCL and CUDA implementations was conducted on various NVIDIA, AMD, and Intel GPUs to evaluate the performance and portability of SYCL across different GPU architectures.

4.2.1. Protein Database Search

Table 4 provides a detailed comparison of the performance and the architectural efficiency of CUDA and SYCL implementations on each platform for protein database search. On NVIDIA GPUs, CUDA and SYCL demonstrated comparable performance and efficiency values. As anticipated, more powerful GPUs achieved higher GCUPS values. The architectural efficiency values ranged from 37% to 52%. In particular, while the RTX 3090 GPU exhibited the highest GCUPS value, the RTX 2070 GPU proved to be the most efficient considering architectural usage.

For AMD and Intel GPUs, only SYCL results are presented, as CUDA exclusively supports NVIDIA hardware. This limitation underscores the superior portability of SYCL compared to CUDA. The performance of the SYCL implementation on AMD and Intel GPUs presents a mixed picture. While AMD's RX 6700 XT dGPU achieved efficiency rates comparable to NVIDIA GPUs (51.7%), the AMD Vega 6 iGPU showed considerably lower performance with only 21.3% efficiency rate. For Intel GPUs, a notable dichotomy emerged between their graphic architectures. Intel's Gen-based iGPUs (UHD 630 and UHD 770) demonstrated impressive architectural efficiency, reaching up to 75.7%, significantly surpassing the baseline established by NVIDIA GPUs. However, Intel's Xe-based GPUs (Arc A770 dGPU and Xe-LPG 128EU

[3] `https://intel.github.io/llvm-docs/GetStartedGuide.html`

[4] `https://codeplay.com/portal/blogs/2022/12/16/bringing-nvidia-and-amd-support-to-oneapi.html`

[5] Swiss-Prot: `https://www.uniprot.org/downloads`

Vendor	NVIDIA						Intel				AMD	
Model	GTX 980	GTX 1080	RTX 2070	V100	RTX 3070	RTX 3090	Arc A770	UHD 630	UHD 770	Xe-LPG 128EU	RX 6700 XT	Vega 6
Type	Discrete						Discrete	Integrated			Discrete	Integrated
Architecture	Maxwell (CC 5.2)	Pascal (CC 6.1)	Turing (CC 7.5)	Volta (CC 7.0)	Ampere (CC 8.6)	Ampere (CC 8.6)	Xe	Gen 9.5	Gen 12	Xe	RDNA 2.0	GCN5
# Cores	16	20	36	80	46	82	32	3	4	8	40	6
# Lanes	32							16	8	16	32	
Inst. throu.	4/2	4/2	2				8				2	
Clock (MHz)	1216	1733	1620	1380	1725	1695	2400	1200	1650	2250	2581	1100
Peak (GCUPS)	155.6	277.3	311	588.8	423.2	741.2	819.2	19.2	35.2	192	550.6	35.2

The instruction throughput for GTX 980 and GTX 1080 is 4 for add/subtract and 2 for max/min. The core instructions include 5 add/subtract and 6 max. Thus, the equivalent throughput is 3.
The core instruction count for each cell update is 12.

Table 1: GPU specifications and their theoretical peak performance in terms of GCUPS

Vendor	Intel								AMD
Model	Core i5-7400	Core i5-10400F	Xeon E5-1620 v3	Xeon E5-2695 v3	Xeon Gold 6138	Core i9-9900K	Core i9-13900K	Core Ultra 9-185H	Ryzen 3 5300U
Segment	Desktop		Server			Desktop		Mobile	
Architecture	Kaby Lake	Comet Lake	Sandy Bridge-E	Haswell	Skylake	Coffee Lake-R	Raptor Lake-S	Meteor Lake-S	Lucienne
# Cores	4	6	4	14	40	8	8/16	6/8/2	4
SIMD set	AVX2	AVX2	AVX2	AVX2	AVX-512	AVX2	AVX2	AVX2	AVX2
# Lanes	8				16	8			
Inst. throu.	1								
Clock (MHz)	3300	4000	3500	1900	1900	4700	5500/4300	5100/3800/2500	3600
Peak (GCUPS)	8.8	16	9.3	35.5	101.3	25.1	75.2	44	9.6

The Intel Core i9-13900K CPU features 8 P-cores and 16 E-cores.
The Intel Core Ultra 9-185H CPU features 6 P-cores, 8 E-cores, and 2 LP E-cores.

Table 2: CPU specifications and their theoretical peak performance in terms of GCUPS

Query sequence		Database sequence		Matrix Size (cells)
Accession	Size	Accession	Size	
NC_000898	162K	NC_007605	172K	28M
NC_003064.2	543K	NC_000914.1	536K	291M
CP000051.1	1M	AE002160.2	1M	1G
BA000035.2	3M	BX927147.1	3M	9G
NC_005027.1	7M	NC_003997.3	5M	35G

Table 3: DNA sequence information used in the tests

	Platform		**CUDA**		**SYCL**	
Vendor	**GPU**	**GCUPS peak**	**GCUPS ach.**	**Arch. eff.**	**GCUPS ach.**	**Arch. eff.**
NVIDIA	GTX 980	155.5	70.6	45.3	67.7	43.5%
	GTX 1080	277.2	104.5	37.7	103.8	37.4%
	RTX 2070	311.0	162.6	52.2	163.1	52.4%
	Tesla V100	588.8	225.0	38.2	233.0	39.6%
	RTX 3070	423.2	173.2	40.9	174.4	41.2%
	RTX 3090	741.3	280.2	37.8	288.6	38.9%
Intel	Arc A770	819.2	x	NA	191.4	23.4%
	UHD 630	19.2	x	NA	13.1	68.4%
	UHD 770	35.2	x	NA	26.6	75.7%
	Xe-LPG 128EU	192.0	x	NA	53.5	27.9%
AMD	RX 6700 XT	550.6	x	NA	284.4	51.7%
	RX Vega 6	35.2	x	NA	7.5	21.3%

Table 4: GCUPS and architectural efficiencies of CUDA and SYCL codes on single GPUs for protein database search.

iGPU) exhibited markedly lower performance, with architectural efficiencies down to 23.4%, respectively. This suboptimal performance can be attributed to ongoing driver optimization challenges in Intel's Arc generation, a factor that has contributed to Intel's dGPU market share dropping below 1% in 2024 [3]. These performance limitations with Xe-based GPUs will likely impact the overall performance portability metrics for Intel GPUs in this study. Further investigation, including detailed code profiling, is planned to understand these performance discrepancies and their relationship to hardware architecture and driver maturity.

Table 5 presents the performance portability of both CUDA and SYCL implementations. The aggregated results are consistent with the individual observations previously noted. For NVIDIA GPUs, the performance portability of CUDA and SYCL is remarkably similar, with values of 42% and 42.2%, respectively. In the case of Intel GPUs, SYCL shows distinct performance patterns: excellent architectural efficiency (up to 75.7%) on Gen-based iGPUs, but significantly lower efficiency (23.4%) on the Arc-based dGPU. As a consequence, SYCL presents 48.9% of performance portability on Intel GPUs. The combination of AMD and Intel GPUs yields performance porta-

Platform set (H)	$\bar{\Phi}(\alpha, p, H)$ CUDA	SYCL
NVIDIA	42%	42.2%
AMD	NA	36.5%
Intel	NA	48.9%
NVIDIA ∪ AMD	NA	40.8%
NVIDIA ∪ Intel	NA	44.9%
Intel ∪ AMD	NA	44.7%
NVIDIA ∪ AMD ∪ Intel	NA	43.5%

Table 5: Performance portability of both CUDA and SYCL codes on single GPUs for protein database search.

Vendor	GPU	Platform GCUPS peak	CUDA GCUPS ach.	CUDA Arch. eff.	SYCL GCUPS ach.	SYCL Arch eff.
NVIDIA	GTX 980	155.5	60.3	38.7	60.4	38.8%
	GTX 1080	277.2	125.0	45.1	125.2	45.1%
	RTX 2070	311.0	197.9	63.6	218.8	70.3%
	Tesla V100	588.8	257.6	43.8	278.9	47.4%
	RTX 3090	741.3	487.5	65.8	496.3	67.0%
Intel	Arc A770	819.2	x	NA	193.6	23.6%
	UHD 630	19.2	x	NA	7.9	41.3%
	UHD 770	35.2	x	NA	14.79	42.0%
	Xe-LPG 128EU	192.0	x	NA	83.30	43.4%

Table 6: GCUPS and architectural efficiencies of CUDA and SYCL codes on single GPUs for pairwise alignment

bility of 44.7%, higher than other vendor combinations. However, including NVIDIA GPUs in the analysis reduces overall performance portability to 43.5%, primarily due to the lower efficiency rates observed in Intel's Xe-based GPUs and AMD's integrated solutions.

4.2.2. Pairwise Alignment

Table 6 presents the performance and the architectural efficiency of CUDA and SYCL implementations on each platform for pairwise alignment. As in the protein database search case, CUDA and SYCL demonstrated comparable performance and efficiency values on NVIDIA GPUs. Once again, the highest GCUPS value was achieved by the RTX 3090 while the most efficient GPU considering architectural usage was the RTX 2070.

The results on the Intel GPUs are mixed. The Arc A770 achieved almost the same architectural efficiency as the protein benchmark. However, the two Gen-based iGPUs showed lower performance while the XE-LPG 128 EU achieved a higher architectural efficiency than the previous case. This behavior could be related to the datatype used in the pairwise alignment kernel (*long kernel*), as discussed below in Section 4.4.3.

The performance portability of both CUDA and SYCL implementations for pairwise alignment is presented in Table 7. Once again, the performance portability of CUDA and SYCL is remarkably similar for NVIDIA GPUs. In particular, CUDA and SYCL present values of 51.4% and 53.7%, respectively. This parity observed for both protein and DNA alignment indicates that both programming models can deliver consistent performance across the diverse range of NVIDIA GPUs utilized in this study.

Platform set (H)	$\bar{\Phi}(\alpha, p, H)$ CUDA	SYCL
NVIDIA	51.4%	53.7%
Intel	NA	37.6%
NVIDIA ∪ Intel	NA	45.7%

Table 7: Performance portability of both CUDA and SYCL codes on single GPUs for pairwise alignment.

In the case of Intel GPUs, while the iGPUs exhibited an acceptable performance, the discrete GPU showed low architectural efficiency values. This fact explains why SYCL presents 37.6% of performance portability for pairwise alignment. Last, the combination of NVIDIA and Intel GPUs yields performance portability of 45.7%, a slightly higher value than the one achieved by the same platform set for the protein benchmark.

4.2.3. Discussion

Building upon this analysis, SYCL demonstrates comparable performance portability to CUDA throughout this study. Specifically, SYCL achieves nearly equivalent architectural efficiency to CUDA across six NVIDIA GPUs representing five distinct microarchitectures. SYCL's key advantage lies in its cross-vendor compatibility, successfully executing on both AMD and Intel GPUs, with particularly strong performance on Intel iGPUs and AMD dGPUs, despite the lower efficiency observed on Intel Arc and AMD integrated solutions.

This comprehensive evaluation underscores SYCL's broad compatibility and its ability to enhance performance across a diverse range of GPU architectures for the application under consideration. The results highlight SYCL's potential as a versatile solution for heterogeneous computing environments, offering a balance between performance and portability that extends beyond the capabilities of CUDA[26, 27, 28, 29, 30, 31].

4.3. Multi-GPU Performance and Portability Results

Experimental work was conducted on various NVIDIA and Intel multi-GPU setups to further investigate the performance and portability of SYCL when exploiting multiple GPUs [6]. Table 8 presents a detailed comparison of the performance and architectural efficiency of CUDA and SYCL implementations across five distinct multi-GPU configurations for protein database search.

For NVIDIA multi-GPU configurations, the efficiency rates achieved when utilizing two GPUs in combination are generally slightly lower than those observed with single-GPU configurations. This phenomenon is evident in three of the four tested

[6] The pairwise alignment benchmark was not considered for this analysis because SW# only support multi-device configuration for protein database search

Platform			CUDA		SYCL	
Vendor	GPUs	GCUPS peak	GCUPS ach.	Arch. eff.	GCUPS ach.	Arch. eff.
NVIDIA	2× GTX 1080	554.6	189.8	34.2%	187.8	33.9%
	2× Tesla V100	1177.6	306.5	36.2%	308.9	36.5%
	2× RTX 3070	846.4	318.1	27.0%	336.5	28.6%
	Tesla V100 + RTX 3090	1330.1	450.5	33.8%	460.7	34.6%
Intel	Arc A770 + UHD 770	854.4	×	NA	126.8	14.8%

Table 8: GCUPS and architectural efficiencies of both CUDA and SYCL codes on multiple GPUs for protein database search.

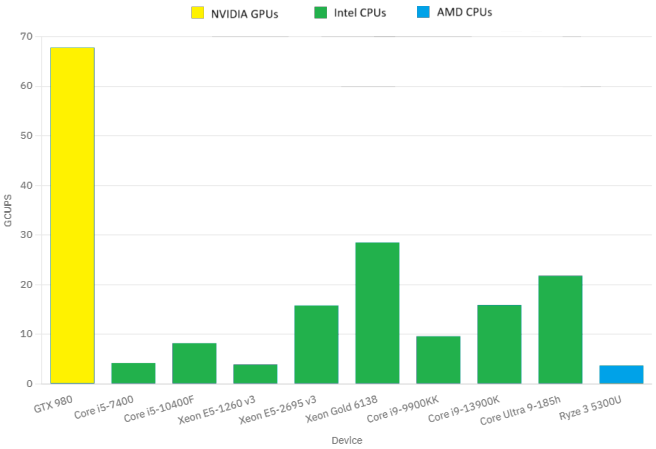

Figure 2: Performance comparison between Intel CPUs and AMD CPU for protein database search, using an NVIDIA GPU as reference.

configurations (except for 2×Tesla V100) and can be attributed to two primary reasons. On the one hand, efficiency usually decreases when the problem size remains fixed while computational resources are increased. On the other hand, the workload distribution strategy employed by *SW#* is rather simple: it distributes query sequences among GPUs without considering sequence length and individual GPU computing power. Given that these sequences vary in length, this approach can lead to load imbalance between GPUs, potentially reducing overall performance. This situation worsens in heterogeneous multi-GPU environments.

4.3.1. Discussion

SYCL once again demonstrates its enhanced functional portability in the case of Intel's multi-GPU configuration. While the performance in this scenario is suboptimal for the previously mentioned reasons, it is noteworthy that SYCL enables the simultaneous utilization of two Intel GPUs of different types: an integrated GPU (iGPU) and a discrete GPU (dGPU). This capability underscores the potential for SYCL to leverage heterogeneous computing resources within a single system.

These findings have significant implications for HPC in heterogeneous environments. The ability of SYCL to maintain performance parity with CUDA in NVIDIA multi-GPU configurations, while also enabling cross-vendor GPU utilization [26, 28, 29, 30, 32], positions it as a versatile solution for developers seeking to maximize computational resources across diverse hardware landscapes[33, 34, 35, 36, 37]. However, the observed efficiency variations in multi-GPU setups highlight the need for more sophisticated workload distribution strategies to fully capitalize on the potential of heterogeneous computing environments, as it was considered in [38].

4.4. CPU Performance and Portability Results

A comprehensive analysis was conducted on various Intel and AMD processors to evaluate the portability and performance of SYCL on different CPU architectures,

4.4.1. Protein Database Search

Fig. 2 presents a performance comparison between Intel and AMD CPUs, comparing them with an NVIDIA GPU as a reference. It is important to remark that the main objective of this figure is to showcase the adaptability of SYCL code across diverse CPU architectures. The SYCL code demonstrates remarkable adaptability by successfully executing on a diverse range of CPUs from different manufacturers, including the Intel Core i5-7400 and AMD Ryzen 3 5300U models. Notably, the code also ran effectively on the Core i9-13900K and Core Ultra 9-185H CPUs, which employ a hybrid architecture.

Even though the original, CUDA code was designed for NVIDIA GPUs, the migrated SYCL code could be adapted and executed on CPUs from various manufacturers with minimal modifications. Two key points deserve emphasis: first, running these tests only required changing the backend during compilation; second, since the ported DPC++ version is pure SYCL code, executing this version on other architectures will only necessitate a compatible compiler.

While functional portability to CPUs has been established, examining the corresponding performance portability is important. Table 9 presents the GCUPS performance and architectural efficiency of the SYCL code running on CPUs for protein database search. The Core i5-10400F emerges as the top performer among all CPUs, achieving a maximum efficiency of 51.2%. Conversely, the Core i9-13900K exhibits the lowest architectural efficiency despite boasting the highest theoretical peak. It's noteworthy that while CPUs generally deliver lower GCUPS compared to GPUs, architectural efficiency surpasses 38% in 7 out of 9 instances.

To provide a more comprehensive analysis, Table 10 presents the performance portability on CPUs, categorizing results by manufacturers and segments [7]. Among Intel CPUs, the desktop segment demonstrates a slight advantage over the server segment. A similar trend is observed when comparing

[7] Single-member groups are discarded

Vendor	Platform: CPU	Platform: GCUPS peak	SYCL: GCUPS ach.	SYCL: Arch. eff.
Intel	Core i5-7400	8.8	4.2	47.6%
	Core i5-10400F	16.0	8.2	51.2%
	Xeon E5-1620 v3	9.3	3.9	42.1%
	Xeon E5-2695 v3	35.5	15.8	44.6%
	Xeon Gold 6138	101.3	28.5	28.2%
	Core i9-9900K	25.1	9.6	38.4%
	Core i9-13900K	75.2	15.9	21.1%
	Core U9-185H	44	21.8	49.5%
AMD	Ryzen 3 5300U	9.6	3.7	38.8%

Table 9: GCUPS and architectural efficiencies of SYCL code on individual CPUs for protein database search.

Platform set (H)	$\bar{\Phi}(\alpha, p, H)$ SYCL
Intel CPUs (desktop)	39.6%
Intel CPUs (server)	38.3%
Intel CPUs	40.3%
Intel ∪ AMD	40.3%

Table 10: Performance portability of SYCL code on individual CPUs for protein database search.

Vendor	Platform: CPU	Platform: GCUPS peak	SYCL: GCUPS ach.	SYCL: Arch. eff.
Intel	Core i5-10400F	16.0	2.0	12.2%
	Xeon E5-1620 v3	0.9	3.9	9.7%
	Xeon Gold 6138	101.3	17.38	17.2%
	Core i9-13900K	75.2	3.8	5.1%

Table 11: GCUPS and architectural efficiencies of SYCL code on individual CPUs for pairwise alignment

different manufacturers, with Intel CPUs showing marginally better performance than the selected AMD CPU. It is important to note that this discrepancy could be attributed to the fact that the oneAPI ecosystem is developed by Intel, which may result in more efficient code generation for their processors. However, caution should be taken in drawing broad conclusions, given the limited number of CPUs within each platform set.

The $\bar{\Phi}$ metric offers a means to assess the adaptability of an application when transitioning between architectures. It's crucial to recognize that *SW#* is a CUDA code specifically optimized for NVIDIA GPUs, while its SYCL counterpart achieved a $\bar{\Phi}$ value of 42.2% during testing on 6 distinct GPUs with 5 different microarchitectures. According to Table 10, the performance portability across all CPU segments and manufacturers stands at 40.3%, exhibiting a negligible difference of less than 2%. This observation underscores the effectiveness of SYCL code across various platforms and highlights its advantage in terms of portability, a critical factor in environments characterized by diverse hardware availability and flexible hardware selection.

4.4.2. Pairwise Alignment

Table 11 presents the GCUPS performance and architectural efficiency of the SYCL code running on CPUs for pairwise alignment. Unlike the protein database search case, SYCL achieves significantly lower architectural efficiency values for CPUs than GPUs. Through code review and profiling, we attribute this difference to the fact that the kernel for long sequences uses scalar data instead of vector data. For protein benchmark, *SW#* uses the *short kernel* for almost all alignments, which leads to explicit instruction vectorization. This code change effectively impacts performance on the CPUs, as the compiler is not able to vectorize operations, achieving lower GCUPS. Moreover, the computation of performance portability reflects the previous behavior. On the one hand, SYCL presents practically the same values of $\bar{\Phi}$ for the protein benchmark (43.5% on GPUs vs 40.2% on CPUs). On the other hand, when pairwise alignment is considered, SYCL achieves $\bar{\Phi}$ values of 45.7% and 11.5% on GPUs and CPUs, respectively. Although the platform sets are similar but not identical, the difference is large enough to notice the performance losses.

4.4.3. Discussion

These findings have significant implications for the development of high-performance, portable applications in fields such as bioinformatics. SYCL demonstrated some ability to maintain consistent performance across diverse CPU architectures [26, 33, 39, 40]. However, kernel programming plays an important role in achieving that goal and programmers should consider this issue if they plan to execute their code on both CPUs and GPUs. In addition, as time goes by, compilers will improve their vectorization capabilities to cope with the current limitations. In that sense, SYCL could serve as a powerful tool for researchers and developers seeking to optimize

computational performance while maintaining code portability [31, 37, 40] in heterogeneous computing environments.

4.5. CPU-GPU Performance and Portability Results

To investigate the performance and portability of SYCL in hybrid CPU-GPU configurations, experiments were conducted on various combinations of NVIDIA, Intel, and AMD devices [8]. Table 12 provides a comprehensive overview of GCUPS and architectural efficiencies for codes executed on various CPU-GPU combinations using SYCL for protein database search [9]. As previously noted, CUDA's unsuitability for CPUs further accentuates the significance of SYCL in hybrid computing scenarios. Analysis of the table reveals that combinations involving the GTX 980 (Maxwell) and RTX 3070 (Ampere) GPUs demonstrate the highest performance, highlighting the code's scalability across different NVIDIA architectures. Nevertheless, even configurations with lower GCUPS, such as Vega 6 and Ryzen 3 5300U, showcase functional portability across a broader spectrum of hardware, including those with more constrained resources.

The architectural efficiency of SYCL exhibits variability across different CPU-GPU combinations, illustrating the code's adaptability to the unique characteristics of each architecture. Configurations featuring NVIDIA GPUs and Intel CPUs demonstrate higher architectural efficiency at the upper end of the table, suggesting a more seamless integration between these technologies. In contrast, pairings of Intel GPUs and CPUs, as well as AMD CPUs and GPUs, display lower architectural efficiencies, highlighting potential areas for enhancing the synergy between these architectures.

Table 13 presents an analysis of performance portability across various hybrid combinations for protein database search. Optimal performance with SYCL is observed in configurations that combine an Intel CPU and an NVIDIA GPU, reaching 27.3%, suggesting potentially more efficient optimization or enhanced compatibility between these specific devices to execute the given codes. Conversely, the least favorable performance is noted with an Intel CPU and an AMD dGPU combination, achieving only 6%.

Configurations incorporating AMD and Intel iGPUs demonstrate moderate performance levels, ranging from 14.03% to 17.8%, respectively. These findings underscore the significant impact of the selection of the CPU and GPU models on the performance of the SYCL code, highlighting the importance of application optimization in integrated CPU+GPU systems. To enhance the validity of these findings, future studies would benefit from an expanded collection of GPUs, particularly from Intel and AMD, as well as an increased number of CPUs, especially from AMD.

4.5.1. Discussion

In this context, incorporating additional devices that present different computing power generally results in performance degradation, even below individual rates. Similar to multi-GPU experiments, outcomes are predominantly influenced by the *SW#* workload distribution strategy, which fails to account for individual device capabilities. Consequently, the *slow* device may be burdened with aligning longer sequences compared to the *fast* one, which will remain idle the rest of the time. This situation is independent of which device is fast or slow. For example, when using the Core i9-13900K+UHD770 configuration, the CPU is the fast device and the GPU is the slow one. However, the opposite case occurs when using the Ryzen 3 5300U+RX Vega 6 system. In any case, the load imbalance detrimentally impacts overall performance. Furthermore, although not equivalent to utilizing two GPUs simultaneously, CPU-GPU combinations are also affected when the problem size remains constant while computational resources increase. This scenario presents potential avenues for future optimizations, including the development of more sophisticated workload distribution algorithms.

5. Related Works

Several studies have compared the performance of SYCL and CUDA implementations across various domains and architectures. These studies can be categorized based on the focus of their comparisons and the architectures used:

- Performance comparison on NVIDIA GPUs: In [26, 28, 29, 30, 32, 33, 34, 35, 36, 37, 27], the authors compared SYCL and CUDA implementations on NVIDIA GPUs, finding that SYCL performance is generally comparable to CUDA, with some cases showing a performance gap attributed to differences in compiler optimizations and memory management.

- Performance comparison on Intel and AMD GPUs: Studies in [26, 30, 33, 39, 40, 27] evaluated SYCL performance on Intel and AMD GPUs, demonstrating that SYCL can achieve equivalent or superior performance compared to native programming approaches on these architectures.

- Performance comparison on CPUs and FPGAs: The works in [39, 40, 41, 42] investigated SYCL performance on various CPU architectures, finding that SYCL is generally less efficient on CPUs compared to native approaches, highlighting the need for additional optimizations.

- Performance portability and migration experiences: Several papers [28, 40, 41, 43, 44, 45, 46, 47] focused on the process of migrating applications from CUDA to SYCL and the performance portability achieved across different architectures. These studies emphasize the importance of device-specific optimizations and parallelization approaches to maximize SYCL performance.

- Domain-specific applications: Some studies focused on specific application domains, such as bioinformatics [26,

[8] The pairwise alignment benchmark was not considered for this analysis because *SW#* only supports multi-device configuration for protein database search

[9] Single-member groups are discarded

Conf.	Platform: CPU + GPU	Platform: GCUPS peak	SYCL: GCUPS ach.	SYCL: Arch. eff.
Intel CPU + NVIDIA GPU	Xeon E5-2695 v3 + GTX980	191	73	38.1%
	Xeon E5-2695 v3 + GTX1080	287	84	29.5%
	Core i5-7400 + RTX2070	320	49	15.3%
	Core i5-10400F + RTX3070	439	149	33.9%
	Xeon Gold 6138 + RTX3090	843	210	24.9%
	Xeon Gold 6138 + V100	690	156	22.6%
Intel CPU + Intel GPU	Core i9-13900K + Arc A770	894	124	13.9%
	Core i9-13900K + UHD770	110	13	12%
	Core i9-13900K + Arc A770 + UHD770	930	102	11%
	Core U9-185H + Xe-LPG 128EU	236	25.8	10.9%
Intel CPU + AMD GPU	Xeon E5-1620 v3 + RX6700	560	33	6%
AMD CPU + AMD GPU	Ryzen 3 5300U + RX Vega 6	45.0	6	14.0%

Table 12: GCUPS and architectural efficiencies of CPU+GPU codes on multiple combinations for protein database search.

Platform set (H)	$\bar{\Phi}(\alpha, p, H)$ SYCL
Intel CPU + NVIDIA GPU	27.4%
Intel CPU + Intel GPU	12%
Intel CPU + GPU(s)	19.8%
CPU + GPU(s)	19.3%

Table 13: Performance portability of SYCL code on CPU+GPU for protein database search.

28, 29, 34], cosmology [40], iterative methods [43], FTLE calculations [44], and molecular dynamics simulations [35]. These articles provide insights into the performance and portability of SYCL in these particular fields.

The studies mentioned above have significantly contributed to understanding the performance and portability of SYCL across various architectures and application domains. However, most of these studies focused on comparing SYCL with native programming models, such as CUDA, on a limited set of platforms, primarily GPUs. Building upon our previous works [5, 6, 7], which evaluated SYCL's performance for the SW# application on GPUs and multi-GPU configurations, this study presents the most comprehensive assessment of SYCL's functional and performance portability to date, encompassing an unprecedented range of computing architectures. Our analysis extends across 12 different GPUs (6 NVIDIA, 4 Intel, and 2 AMD), 9 distinct CPU models (8 Intel and 1 AMD) from various market segments (desktop, server, and mobile), and multiple hybrid CPU-GPU configurations. This extensive hardware coverage, which significantly surpasses previous studies in the field, allowed us to thoroughly investigate SYCL's capabilities across diverse architectural paradigms, from traditional single-device setups to complex heterogeneous systems combining CPUs and GPUs from different vendors.

6. Conclusions and Future Work

This study addresses the challenges of functional and performance portability in heterogeneous computing environments by evaluating SYCL and CUDA for Smith-Waterman biological sequence alignment across diverse GPU and CPU architectures, including multi-GPU and hybrid configurations. The experimental results demonstrate that CUDA and SYCL deliver comparable performance on NVIDIA GPUs in both single and multi-GPU scenarios, while SYCL exhibits superior architectural efficiency on AMD and Intel GPUs in the majority of cases tested. In multi-GPU configurations, both programming models showed similar scaling behavior. While the performance improved when using additional GPUs, the current workload distribution strategy somewhat limited the associated benefits. These findings underscore SYCL's broad compatibility and its potential to enhance performance across a diverse range of GPU configurations for this specific application, from single accelerators to more complex multi-GPU setups.

SYCL also demonstrated remarkable versatility and effectiveness across CPUs from various manufacturers, including the latest hybrid architectures like Intel's Meteor Lake, which combines up to three different types of cores within a single

processor. This successful adaptation to both traditional and modern heterogeneous CPU designs is particularly noteworthy, as it showcases SYCL's ability to handle increasingly complex processor architectures effectively. Even so, achieving performance portability to CPUs can remain challenging in some cases, particularly when instruction vectorization cannot be guaranteed. However, the success stories are promising, as the protein benchmark case. The performance consistency across all CPU categories and manufacturers was marginally lower than that achieved on NVIDIA GPUs, highlighting SYCL's potential as a unified programming model for heterogeneous computing environments, regardless of the underlying architectural complexity.

Furthermore, SYCL showed exceptional functional portability in various CPU-GPU hybrid configurations. However, performance in these heterogeneous scenarios was constrained by limitations inherent in the original code base rather than SYCL-specific issues. This highlights the importance of developing more sophisticated work distribution mechanisms that consider the distinct characteristics and capabilities of each computing device in heterogeneous environments. Such optimizations would be crucial for maximizing performance in CPU-GPU configurations, regardless of the programming model used.

Future work will focus on:

- Optimizing the SYCL code to achieve maximum performance. Specifically, the original SW# suite does not incorporate certain known optimizations for SW alignment [18], such as instruction reordering to reduce the instruction count and the use of lower-precision integers to increase parallelism [10]. Additionally, our objective is to improve the workload distribution strategy when utilizing multiple devices. These improvements are expected to lead to higher efficiency rates.
- Executing the SYCL code on other FPGA architectures and considering alternative programming models, such as Kokkos and RAJA, to strengthen the current performance portability study. This expansion of scope will provide a more comprehensive understanding of SYCL's capabilities across diverse hardware and software ecosystems, further informing its potential as a unified programming model for heterogeneous computing environments.

CRediT authorship contribution statement

Manuel Costanzo: Conceptualization, Software, Validation, Writing - Original Draft, Investigation, Visualization, Data Curation. **Enzo Rucci**: Conceptualization, Methodology, Investigation, Supervision, Writing - Original Draft, Project administration. **Carlos García-Sanchez**: Conceptualization, Investigation, Writing - Review & Editing, Resources. **Marcelo Naiouf**: Writing - Review & Editing, Funding acquisition. **Manuel Prieto-Matías**: Writing - Review & Editing, Funding acquisition.

[10] It is important to note that at the time of SW#'s development, most CUDA-enabled GPUs did not support efficient arithmetic operations on 8-bit vector data types.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve language and readbility. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Data availability

The SW# CUDA software used in this study is available at `https://github.com/mkorpar/swsharp`, while the migrated SW# software is available at `https://github.com/ManuelCostanzo/swsharp_sycl`. The protein data utilized for this research are sourced from the UniProtKB/Swiss-Prot (Swiss-Prot) database (release 2022_07) (available at `https://www.uniprot.org/downloads`), and the Environmental Non-Redundant (Env. NR) database (release 2021_04) (available at `https://ftp.ncbi.nlm.nih.gov/blast/db/`).

Funding

Grant PID2021-126576NB-I00 funded by MCIN/AEI/10.13039/501100011033 and, as appropriate, by "ERDF A way of making Europe", by the "European Union" or by the "European Union Next Generation EU/PRTR".

References

[1] P. Navaux, A. Lorenzon, M. Serpa, Challenges in high-performance computing, Journal of the Brazilian Computer Society 29 (2023) 51–62. doi:10.5753/jbcs.2023.2219.

[2] A. Shilov, Discrete GPU sales increase as Intel's share drops to 0, Tom's Hardware (9 2024).
URL https://www.tomshardware.com/pc-components/gpus/discrete-gpu-sales-increase-as-intels-share-drops-to-0

[3] J. Norem, Intel has reportedly lost all its discrete GPU market share, ExtremeTech (9 2024).
URL https://www.extremetech.com/gaming/intel-has-reportedly-lost-all-its-discrete-gpu-market-share

[4] Khronos SYCL working group, Sycl 2020 specification (2023).
URL https://registry.khronos.org/SYCL/specs/sycl-2020/html/sycl-2020.html

[5] M. Costanzo, E. Rucci, C. García-Sánchez, M. Naiouf, M. Prieto-Matías, Migrating cuda to oneapi: A smith-waterman case study, in: I. Rojas, O. Valenzuela, F. Rojas, L. J. Herrera, F. Ortuño (Eds.), Bioinformatics and Biomedical Engineering, Springer International Publishing, Cham, 2022, pp. 103–116. doi:10.1007/978-3-031-07802-6_9.

[6] M. Costanzo, E. Rucci, C. Garcia-Sanchez, M. Naiouf, M. Prieto-Matias, Comparing performance and portability between cuda and sycl for protein database search on nvidia, amd, and intel gpus, in: 2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 141–148. doi:10.1109/SBAC-PAD59825.2023.00023.

[7] M. Costanzo, E. Rucci, C. G. Sánchez, M. R. Naiouf, M. Prieto, Assessing opportunities of sycl for biological sequence alignment on gpu-based systems, J. Supercomput. 80 (2022) 12599–12622. doi:10.1007/s11227-024-05907-2.

[8] H. Lan, W. Liu, Y. Liu, B. Schmidt, SWhybrid: A hybrid-parallel framework for large-scale protein sequence database search, in: Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International, IEEE, 2017, pp. 42–51. doi:10.1109/IPDPS.2017.42.

[9] D. B. Kirk, W.-m. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. doi:10.1016/C2015-0-02431-5.

[10] J. E. Stone, D. Gohara, G. Shi, Opencl: A parallel programming standard for heterogeneous computing systems, Computing in Science and Engg. 12 (3) (2010) 66–73. doi:10.1109/MCSE.2010.69.

[11] OpenMP ARB, The OpenMP Specification, https://www.openmp.org/.

[12] OpenACC organization, The OpenACC Specification, https://www.openacc.org/.

[13] R. Farber, Parallel Programming with OpenACC, 1st Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016.

[14] B. Ashbaugh, A. Bader, J. Brodman, J. Hammond, M. Kinsner, J. Pennycook, R. Schulz, J. Sewall, Data parallel c++ enhancing sycl through extensions for productivity and performance, in: Proceedings of the International Workshop on OpenCL, 2020, pp. 1–2. doi:10.1145/3388333.3388653.

[15] E. F. D. O. Sandes, A. Boukerche, A. C. M. A. D. Melo, Parallel optimal pairwise biological sequence comparison: Algorithms, platforms, and classification, ACM Comput. Surv. 48 (4) (Mar. 2016). doi:10.1145/2893488.
URL https://doi.org/10.1145/2893488

[16] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, Journal of Molecular Biology 147 (1) (1981) 195–197. doi:10.1016/0022-2836(81)90087-5.

[17] O. Gotoh, An improved algorithm for matching biological sequences, in: Journal of Molecular Biology, Vol. 162, 1981, pp. 705–708. doi:10.1016/0022-2836(82)90398-9.

[18] T. Rognes, Faster Smith-Waterman database searches with inter-sequence SIMD parallelization, BMC Bioinformatics 12:221 (2011). doi:10.1186/1471-2105-12-221.

[19] S. Pennycook, J. Sewall, V. Lee, Implications of a metric for performance portability, Future Generation Computer Systems 92 (2019) 947–958. doi:10.1016/j.future.2017.08.007.

[20] A. Marowka, Reformulation of the performance portability metric, Software: Practice and Experience 52 (1) (2022) 154–171. doi:10.1002/spe.3002.

[21] M. Korpar, M. Sikic, SW# - GPU-enabled exact alignments on genome scale., Bioinformatics 29 (19) (2013) 2494–2495. doi:10.1093/bioinformatics/btt410.

[22] M. Korpar, M. Sosic, D. Blazeka, M. Sikic, SWdb: GPU-Accelerated Exact Sequence Similarity Database Search, PLOS ONE 10 (12) (2016) 1–11. doi:10.1371/journal.pone.0145857.

[23] E. Rucci, C. García, G. Botella, A. De Giusti, M. Naiouf, M. Prieto-Matías, State-of-the-Art in Smith-Waterman Protein Database Search on HPC Platforms, Springer International Publishing, Cham, 2016, pp. 197–223. doi:10.1007/978-3-319-41279-5_6.
URL https://doi.org/10.1007/978-3-319-41279-5_6

[24] MJ Rutter, Intel's Variable Clock Speeds and Benchmarking, https://www.mjr19.org.uk/IT/clocks.html (2023).

[25] Intel Corporation, Alder Lake S (2023).
URL https://www.intel.la/content/www/xl/es/products/\platforms/details/alder-lake-s.html

[26] M. Haseeb, N. Ding, J. Deslippe, M. Awan, Evaluating performance and portability of a core bioinformatics kernel on multiple vendor gpus, in: 2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC), 2021, pp. 68–78. doi:10.1109/P3HPC54578.2021.00010.

[27] I. Vasileska, P. Tomšič, L. Kos, L. Bogdanović, Unveiling performance insights and portability achievements between cuda and sycl for particle-in-cell codes on different gpu architectures, in: 2024 47th MIPRO ICT and Electronics Convention (MIPRO), 2024, pp. 1115–1120. doi:10.1109/MIPRO60963.2024.10569866.

[28] Z. Jin, J. 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, Association for Computing Machinery, New York, NY, USA, 2022. doi:10.1145/3535508.3545591.

[29] Z. Jin, J. S. Vetter, Understanding performance portability of bioinformatics applications in sycl on an nvidia gpu, in: 2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2022, pp. 2190–2195. doi:10.1109/BIBM55620.2022.9995222.

[30] L. Solis-Vasquez, E. Mascarenhas, A. Koch, Experiences migrating cuda to sycl: A molecular docking case study, in: Proceedings of the 2023 International Workshop on OpenCL, IWOCL '23, Association for Computing Machinery, New York, NY, USA, 2023. doi:10.1145/3585341.3585372.

[31] G. Castaño, Y. Faqir-Rhazoui, C. García, M. Prieto-Matías, Evaluation of intel's dpc++ compatibility tool in heterogeneous computing, Journal of Parallel and Distributed Computing 165 (2022) 120–129.

[32] B. Homerding, J. Tramm, Evaluating the performance of the hipsycl toolchain for hpc kernels on nvidia v100 gpus, in: Proceedings of the International Workshop on OpenCL, IWOCL '20, Association for Computing Machinery, New York, NY, USA, 2020. doi:10.1145/3388333.3388660.

[33] Y. Faqir-Rhazoui, C. García, Exploring the performance and portability of the k-means algorithm on sycl across cpu and gpu architectures, J. Supercomput. 79 (16) (2023) 18480–18506. doi:10.1007/s11227-023-05373-2.

[34] Z. Jin, J. S. Vetter, Understanding performance portability of sycl kernels: A case study with the all-pairs distance calculation in bioinformatics on gpus, in: 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2023, pp. 366–372. doi:10.1109/IPDPSW59300.2023.00067.

[35] L. Apanasevich, Y. Kale, H. Sharma, A. M. Sokovic, A comparison of the performance of the molecular dynamics simulation package gromacs implemented in the sycl and cuda programming models (2023). doi:10.48550/arXiv.2406.10362.

[36] L. A. Torres, C. J. Barrios, Y. Denneulin, C. Científico, G. de Investigación, C. A. y a Gran, Evaluation of computational and energy performance in matrix multiplication algorithms on cpu and gpu using mkl, cublas and sycl, 2024.
URL https://api.semanticscholar.org/CorpusID:270063378

[37] Y. Faqir-Rhazoui, C. Garcia, Sycl in the edge: performance and energy evaluation for heterogeneous acceleration, The Journal of Supercomputing 80 (2024) 1–21. doi:10.1007/s11227-024-05957-6.

[38] B. Schmidt, F. Kallenborn, A. Chacon, C. Hundt, Cudasw++ 4.0: ultra-fast gpu-based smith–waterman protein sequence database search, BMC bioinformatics 25 (1) (2024) 342.

[39] I. Z. Reguly, Evaluating the performance portability of sycl across cpus and gpus on bandwidth-bound applications, in: Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023, pp. 1038–1047. doi:10.1145/3624062.3624180.

[40] E. M. Rangel, S. J. Pennycook, A. Pope, N. Frontiere, Z. Ma, V. Madananth, A performance-portable sycl implementation of crk-hacc for exascale, in: Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023, pp. 1114–1125. doi:10.1145/3624062.3624187.

[41] J. Franquinet, Performance portability analysis of sycl with a classical cg on cpu, gpu, and fpga, Master's thesis (2023).

[42] M. de Castro, F. J. Andujar, R. Osorio, R. Carratalá-Sáez, D. Llanos, Challenging portability paradigms: Fpga acceleration using sycl and opencl (09 2024). doi:10.48550/arXiv.2409.03391.

[43] P. Nguyen, P. Nayak, H. Anzt, Porting batched iterative solvers onto intel gpus with sycl, SC-W '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 1048–1058. doi:10.1145/3624062.

3624181.

[44] R. Carratalá-Sáez, Y. Torres, A. Gonzalez-Escribano, D. R. Llanos, et al., Open sycl on heterogeneous gpu systems: A case of study, arXiv preprint arXiv:2310.06947 (2023). doi:10.13140/RG.2.2.32379.73769.

[45] Z. Jin, J. S. Vetter, Understanding sycl portability for pseudorandom number generation: a case study with gene-expression connectivity mapping, in: 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2023, pp. 295–298.

[46] C. Weckert, L. Solis-Vasquez, J. Oppermann, A. Koch, O. Sinnen, Altissycl: Migrating altis benchmarking suite from cuda to sycl for gpus and fpgas, in: Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023, pp. 547–555. doi:10.1145/3624062.3624542.

[47] R. Mueller-Albrecht, Syclomatic: Sycl adoption for everyone - moving from cuda to sycl gets progressively easier: Advanced migration considerations, in: Proceedings of the 12th International Workshop on OpenCL and SYCL, IWOCL '24, Association for Computing Machinery, New York, NY, USA, 2024. doi:10.1145/3648115.3648124.