ISSN 2007-9737

HAMP: A Hardware Accelerator Multi-Platform Benchmark for High Performance Heterogeneous Computing

Rogelio Valdez*, Yazmin Maldonado

Tecnológico Nacional de México/Instituto Tecnológico de Tijuana,
Posgrado en Ciencias de la Ingeniería Tijuana,
Mexico

{rogelio.valdez17, yaz.maldonado}@tectijuana.edu.mx

Abstract. This paper introduces a Hardware Accelerator Multi-Platform Benchmark for high performance heterogeneous computing called HAMP, a multi-platform tool designed to analyze the performance of hardware accelerators such as CPUs, GPUs and FPGAs. HAMP benchmark stands out for its ability to evaluate different hardware architectures within a unified environment using a common programming language, C++ with the OpenCL framework. The multi-platform tool integrates a selection of state-of-the-art kernels for the evaluation of hardware characteristics and arithmetic operations, employing two performance metrics (speed and bandwidth). Developed in Python, a graphical user interface (GUI) simplifies interaction with the tool, allowing users to configure and execute kernels on the hardware accelerator without needing technical knowledge. Experiments were performed on three hardware accelerators with the five kernels comprising the HAMP benchmark. The results obtained indicate that HAMP is an efficient and reliable multi-platform tool, facilitating the comparison of diverse hardware architectures within a unified design environment, a capability not previously available in the state-of-the-art.

Keywords. Hardware accelerator, benchmark, heterogeneous computing, FPGA, GPU.

1 Introduction

Heterogeneous computing is an emerging paradigm due to the high demand for faster and more efficient computer systems [37]. Heterogeneous computing employs a variety of accelerators, including CPUs, GPUs, DSPs, APUs (Accelerated Processing Units), TPUs (Tensor Processing Units), and, more recently, FPGAs (Field Programmable Gate Arrays) [6, 16, 36, 37, 38]. However, each accelerator has a specific architecture, combining them in a heterogeneous computing system allows them to leverage their respective strengths, resulting in more efficient application execution, reduced processing times, and increased energy efficiency.

This enables the development of more powerful systems that open new possibilities for innovation in areas such as artificial intelligence [12, 35, 48], big data analysis [52, 24], machine learning [19, 39], among others. Combining different accelerators presents several challenges and opportunities.

One key challenge is the difference in programming paradigms. While CPUs and GPUs are programmed using software languages, FPGAs are programmed using hardware description languages (HDLs) such as VHDL (Very High Speed Integrated Circuit HDL), which require specialized knowledge of electronic design tools. This disparity necessitates a programming language capable of communicating with diverse accelerators.

Another challenge is designing and updating benchmarks, which are critical for assessing accelerator performance, especially given the diverse applications implemented on FPGAs. Benchmarks must be updated regularly to evaluate new capabilities and accommodate this application diversity.

Table 1. Taxonomy of Rodinia's benchmarks

Kernel	Area
K-means	Linear algebra
Needleman-Wunsch	Dynamic programming
HotSpot	Structured grid
Back propagation	Unstructured grid
SRAD	Structured grid
Leukocyte tracking	Structured grid
Breadth-First Search	Graph transversal
Stream cluster	Linear algebra
Similarity scores	MapReduce

The BenchCouncil defines a benchmark as a tool for quantitatively measuring solutions to a defined problem. This can involve explicitly defining the problem, providing a specific example, using a current best-practice solution as a reference, or establishing a measurement standard [51].

This paper presents HAMP (Hardware Accelerator Multi-Platform), a novel benchmark that, unlike existing benchmarks, enables direct performance comparisons between CPUs, GPUs, and FPGAs in a single environment. While previous benchmarks focused on a limited set of accelerators, HAMP integrates all three into a single testing enviroment. This is achieved through OpenCL, providing portability across architectures and eliminating dependency on vendor-specific solutions such as CUDA. By providing a common execution and evaluation structure, HAMP enables impartial comparisons between different hardware accelerators. The main contribution of this work is summarized as follows:

— The development of HAMP, a novel multi-platform benchmark compatible with three distinct hardware accelerators: CPUs, GPUs, and FPGAs.

— The development of the first benchmark in this domain to feature a user-friendly Graphical User Interface (GUI).

The paper is organized as follows: Section 2 presents a literature review of benchmarks, features, and evaluation metrics; in Section 3, the HAMP benchmark overview is presented, while in Section 4 the experimental evaluation, performance comparison and discussion are analyzed, finally, conclusions and future work are presented in Section 5.

2 Benchmarking and State-of-the-Art in Heterogeneous Computing

This section provides a comprehensive review of the state-of-the-art in hardware accelerator benchmarking. The review examines the benchmarks' characteristics, target devices, application areas, and the problems they address. This body of research has been essential for establishing standardized practices and fostering the development of new benchmarks in this rapidly evolving field.

2.1 Rodinia

Rodinia is a benchmark suite specializing in heterogeneous computing for multi-core CPUs and GPUs [3]. It includes a collection of kernels from the literature. Rodinia has been used to evaluate various parallelization approaches, such as OpenMP [33] for CPUs and CUDA [31] for GPUs.

Its main application areas include: (1) linear algebra, where algorithms based on matrix and vector operations are used (e.g., k-means and stream clustering); (2) dynamic programming, such as the Needleman-Wunsch algorithm; (3) structured grids, where algorithms based on structured matrices are applied, such as SRAD and Hotspot; (4) unstructured grid; (5) graph traversal; and (6) MapReduce [3].

Rodinia offers a diverse set of tools for evaluating heterogeneous computing, including algorithms, kernels, and examples of database-driven applications, which are classified in Table 1.

Table 2. Complexity classification of SHOC benchmark

Level	Kernel
0	Bus speed Download Bus speed readback Device memory bandwidth Kernel compilation
1	Peak flops Queuing delay Resource contention
2	Sorting problems Matrix operations Fast Fourier Transform

Table 3. Kernels and domains of CHO benchmark

Kernel	Domain
Sum Division Multiplication Sine	Arithmetic
Adaptive differential pulse code encoder and decoder Prediction coding JPEG image decoder Motion vector decoding	Media
Advanced encryption standard Blowfish algorithm Secure Hash algorithm	Cryptography
Simplified MIPS processor	Other

Table 4. Taxonomy of Mirovia's benchmarks

Level	Kernel	Area	Domain
0	Bus speed download Bus speed readback Device memory bandwidth Peak flops	Hardware characteristics	Engineering
1	Sorting Multiplication	Arithmetic Linear algebra	Sorting Arithmetic
2	Needleman-Wunsh CFDSolver	Dynamic programming Unstructured grid	Bioinformatics Fluid dynamics
3	Neural networks	Dense algebra linear	Deep learning

2.2 SHOC

SHOC is a Scalable Heterogeneous Computing benchmark for GPUs and CPUs [6]. SHOC was developed to standardize performance and stability measurement of heterogeneous computing using OpenCL. SHOC provides a suite of computational benchmarks organized by complexity level.

Table 2 lists several of kernels, with level 2 indicating the highest complexity.

2.3 CHO

CHO is a suite of computational benchmarks [29] for Altera (Intel) FPGAs, programmed in OpenCL. Table 3 shows the twelve CHO Kernels, classified by domain.

2.4 Mirovia

Mirovia is an advanced benchmark suite for evaluating modern hardware accelerators [16]. Unlike previous benchmarks like Rodinia [3] and SHOC [6], Mirovia is specifically optimized for heterogeneous computing on GPUs, prioritizing workload efficiency, problem size, and the unique characteristics of each task. Furthermore, Mirovia leverages new CUDA features, such as unified memory management and improved parallel kernel execution. Table 4 presents an analysis of Mirovia's complexity levels by kernel, application area, and problem domain.

2.5 Evaluation Metrics

This section describes evaluation metrics for modern hardware accelerators. Benchmarking is essential for comparing hardware accelerators under standardized conditions. The key metrics are described below:

Speed: Among the most common metrics is speed, which refers to task completion time (also often referred to as execution time or latency). It is measured in units of time (seconds, milliseconds, nanoseconds), where lower values signify better performance [34, 16, 28].

Bandwidth: This metric measures the amount of data that can be transferred per unit of time between various components, including memory and processor, and FPGA and processor. Typical units are bits per second (bps), bytes per second (B/s), and gigabytes per second (GB/s). Higher bandwidth generally means better performance. Bandwidth is defined as:

$$bandwidth = (B_r + B_w)/T, \tag{1}$$

where B_r is defined as the number of bytes read from memory, B_w represents the number of bytes

written to memory, and T represents the execution time of the algorithm in seconds [15, 21, 34, 14].

Speedup: This measures the improvement in performance achieved by using a hardware accelerator or a parallel processing technique compared to a baseline (usually a sequential implementation or a less powerful hardware configuration). It's calculated as the ratio of the execution time of the baseline to the execution time of the accelerated version. A higher speedup value indicates better performance improvement. The speedup is calculated as follows:

$$S_p = \frac{T_s(n,1)}{T(n,p)}, \tag{2}$$

where $T_s(n,1)$ represents the execution time of the optimal sequential algorithm and $T(n,p)$ represents the execution time of the parallel algorithm for the same input n [14, 42].

Floating Point Operations Per Second (FLOPS): This measures the number of floating-point operations that a system can perform per second [27]. It's a common metric for evaluating the performance of hardware accelerators that heavily rely on floating-point arithmetic. Higher FLOPS generally indicate better performance for these types of workloads.

2.6 Principal Kernels in the State-of-the-Art Benchmarks

In this section, we present some of the most widely used kernels for evaluating CPU, GPUs and FPGAs. These core kernels provide standardized datasets and evaluation metrics, enabling researchers to compare the performance of different accelerators in terms of speed and power efficiency across a variety of workloads.

— Bus speed download and bus speed readback kernels measure the speed at which data is transferred between the computer and the hardware accelerator, essential for evaluating PCI bus efficiency [16, 6, 27, 15].

— The sum kernel measures the speed and bandwidth of the hardware accelerator. This benchmark assesses performance during vector sum [16, 6, 27, 50, 21, 22, 20, 41, 9, 4, 46].

— The multiplication measures the speed and bandwidth of hardware accelerators when performing matrix multiplication on matrices of different sizes, offering insights into their performance [16, 6, 27, 15, 21, 22, 20, 9, 4, 46, 30, 5, 44, 42, 49, 43, 26, 2, 40, 1, 34, 47, 10, 18, 28].

— The sorting kernel evaluates the performance of the hardware accelerator during character vector sorting, measuring both speed and bandwidth [16, 6, 15, 42, 26, 1, 47, 10, 13].

2.7 Discusion

A review of the state-of-the-art reveals that kernels in benchmarks are executed on one or two hardware accelerators, with CPUs, GPUs, and occasionally FPGAs either individually or in combination.

Another relevant aspect is that commonly evaluated kernels include vector operations, matrix multiplication, and sorting problems.

It is important to note that the benchmarks discussed in the state-of-the-art are typically executed through the command line, potentially limiting their accessibility to users unfamiliar with command-line interfaces. Furthermore, the most commonly used metrics in these studies include speed, bandwidth, speedup, and FLOPS.

Table 5 presents a comparison between the HAMP benchmark (our proposal) and benchmarks from the state-of-the-art. Typical benchmarks include hardware characteristics, vector operations, matrix problems, and sorting problems. Furthermore, speed and bandwidth are the most common performance metrics. A key distinction of HAMP compared to existing benchmarks is its ability to evaluate CPUs, GPUs, and FPGAs under a unified execution environment, eliminating the need for separate benchmarking tools for each architecture. Current benchmarks, such as Rodinia, SHOC, and Mirovia, typically focus on one or two accelerators

Table 5. Features of state-of-the-art benchmarks versus HAMP

Benchmark suite		Rodinia	SHOC	CHO	Mirovia	HAMP
Kernel	Hardware characteristics		✓	✓	✓	✓
	Vector operations	✓		✓	✓	✓
	Matrix operations	✓	✓		✓	✓
	Sorting problems	✓	✓	✓	✓	✓
	Image compression	✓		✓		
	Audio de encoding			✓		
	Encryption algorithms			✓		
	Computer vision				✓	
	Data science				✓	
Metrics	Speed	✓	✓	✓	✓	✓
	Bandwidth		✓		✓	✓
	Speed up	✓			✓	
	FLOPS		✓		✓	
Accel.	CPU	✓	✓		✓	✓
	GPU	✓	✓		✓	✓
	FPGA			✓		✓
Multi-platform						✓
GUI						✓

and rely on vendor specific frameworks like CUDA for GPUs. In contrast, HAMP uses OpenCL to establish a standardized benchmarking methodology across multiple accelerator types, enabling direct performance comparisons under the same execution conditions. Furthermore, HAMP integrates a GUI that simplifies the benchmarking process, making it more accessible for researchers and developers while ensuring consistency in experimental evaluation.

Figure 1 shows the relational graph illustrates the relationships between hardware accelerators (hexagonal nodes), kernels (circular nodes), and scientific publications (Che et al. [3])(Coronado et al. [5])(Danalis et al. [6])(Dinelli G. et al. [7])(Dongarra et al. [8])(Hu B. et al. [16])(Jiang et al. [21])(Khoould et al. [22])(Meyer et al. [27])(Navarro [42])(Ndu et al. [29])(Taylor Lloyd et al. [41])(Verman et al. [45])(Wei et al. [49])(Zeke et al. [50]).

Each hexagonal node represents a specific accelerator (CPU, GPU, FPGA), while each circular node represents a benchmark used to evaluate its performance. The rectangular nodes represent scientific papers where these benchmarks have been used or designed for the accelerators.

To highlight the proposed HAMP, its kernels are shown in red, along with the three hardware accelerators it supports. Unlike other benchmarks, HAMP not only encompasses a diverse set of kernels but also offers compatibility with three specific hardware accelerators, enabling more comprehensive and accurate performance evaluations.

3 HAMP Benchmark Overview

HAMP benchmark is a multi-platform tool, multi-platform refers to the capability of HAMP to execute across diverse hardware accelerator architectures, specifically CPUs, GPUs, and FPGAs, regardless of the underlying operating system. Designed in C++, OpenCL, and Python to evaluate the performance of a wide range of hardware accelerators, including CPUs, GPUs, and FPGAs. Developed using C++, OpenCL, and Python, HAMP leverages the strengths of each language to provide a powerful and flexible benchmarking solution. C++ delivers the necessary performance for evaluation, OpenCL enables parallel programming on different accelerators including Intel FPGAs, and Python facilitates interaction with the tool and analysis of results. HAMP not only allows for performance evaluation but also offers additional features such as an intuitive graphical user interface that simplifies experiment configuration, results visualization, and comparative analysis. The ability to load or randomly generate data for experiments adds flexibility and allows users to tailor HAMP to their specific needs.

The required libraries, drivers, and compilers are essential for HAMP to effectively utilize the heterogeneous hardware. For instance, OpenCL libraries are necessary for multi-platform execution of kernels, while specific drivers ensure optimal communication with each hardware accelerator. The following is a list of the specific libraries and drivers required for each hardware accelerator:

ISSN 2007-9737

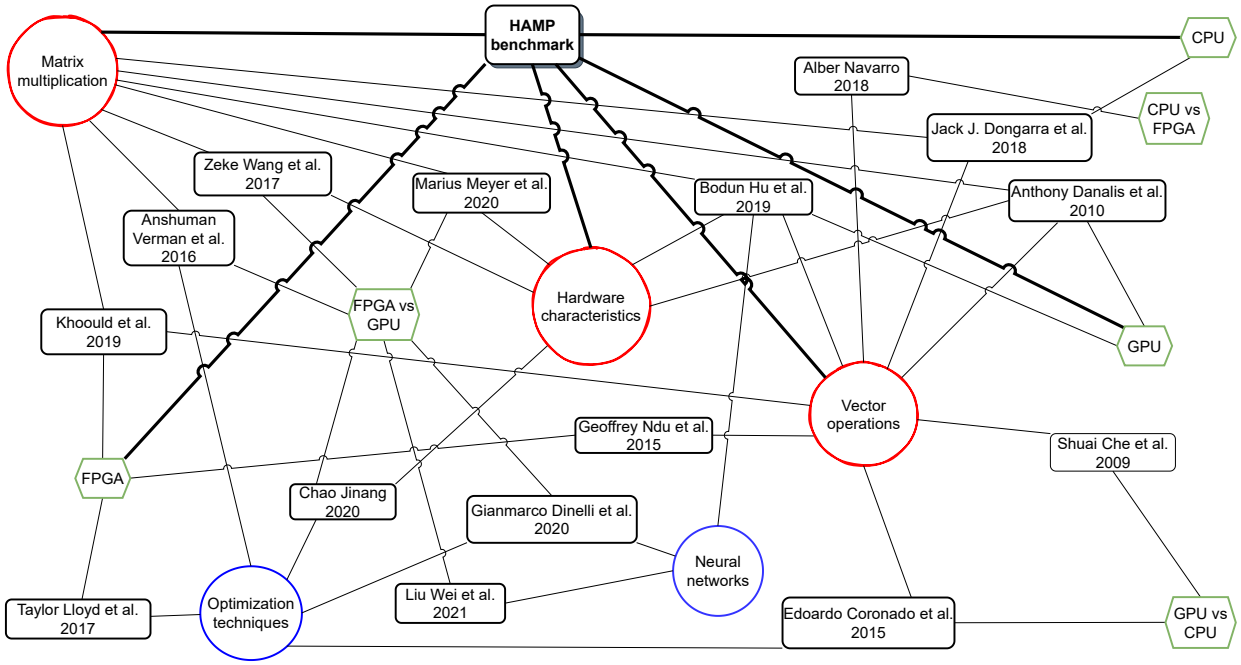

Fig. 1. Relational graph of state-of-the-art benchmarks, including HAMP

— **CPUs:** Requires the installation of OpenCL drivers, such as the Intel OpenCL CPU runtime, which enables CPU-based execution of OpenCL kernels.

— **GPUs (Nvidia & Intel):** The system must have the appropriate vendor-specific OpenCL runtime installed, such as, the Nvidia CUDA Toolkit with OpenCL support for Nvidia GPUs, and the Intel Compute Runtime (ICD) for Intel GPUs.

— **FPGAs (Intel/Terasic):** The Intel FPGA SDK for OpenCL (AOCL) is required, which provides the necessary Board Support Package (BSP) and compilation tools to generate bitstreams compatible with the target FPGA device.

While HAMP requires standard C++ compilers such as GCC or G++. For FPGA execution, the AOCL compiler is used to compile OpenCL kernels into FPGA-optimized hardware descriptions. The above requirements are necessary to ensure that each hardware accelerator operates under optimal conditions, avoiding performance inconsistencies caused by missing drivers or incorrect runtime configurations.

HAMP incorporates five widely used and recognized kernels from the state-of-the-art benchmarks: Bus speed download, bus speed readback, vector sum, matrix multiplication, and sorting problem. These kernels were selected for their relevance in evaluating the performance of hardware accelerators and their ability to represent a variety of workloads and memory access patterns. To evaluate performance, HAMP uses two key metrics: bandwidth (Eq. 1) and speed (Section 2.5). These metrics measure the efficiency of data transfer and the execution speed of the kernels, respectively, providing a complete view of the performance of the accelerators. The combination of these kernels and metrics ensures the relevance and validity of the HAMP proposal, and allows for comparable results with state-of-the-art. Figure 2 shows the HAMP benchmark data flow.

OpenCL programming for HAMP consists of six main steps: (1) Kernel loading, (2) Data input, (3)

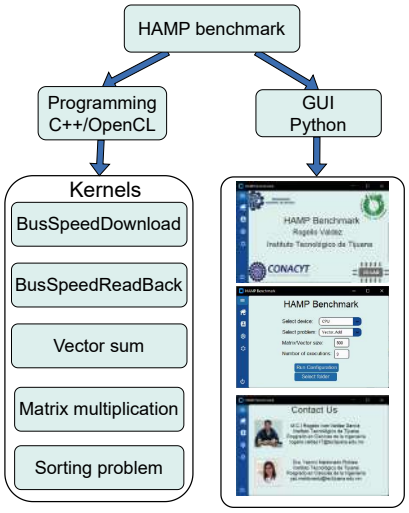

Fig. 2. Overview of HAMP benchmark

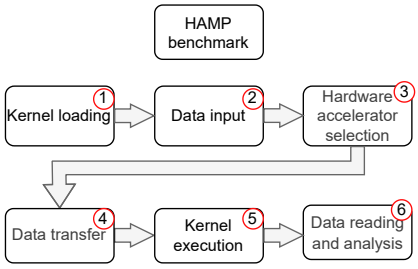

Fig. 3. Flowchart of OpenCL programming for HAMP

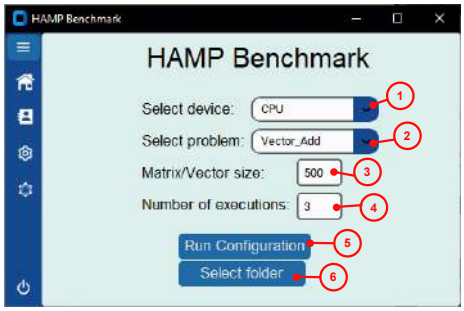

Fig. 4. HAMP GUI configuration options

Hardware accelerator selection, (4) Data transfer, (5) Kernel execution, and (6) Data reading and analysis. Figure 3 shows a flowchart representing these six main steps.

In the first step, the OpenCL kernel code is loaded from a file or string into the host program. Next, the input data to be processed on the accelerator is prepared. Subsequently, the specific accelerator (CPU, GPU, or FPGA) on which the kernel will be executed is chosen. The input data is then transferred from the host memory to the accelerator memory. Once the data resides on the accelerator, the OpenCL kernel is executed, processing the input data. Finally, the results are transferred from the accelerator memory back to the host memory, where they are analyzed and presented to the user.

Unlike other benchmarking tools, the GUI provides a wide range of options for adjusting input data, allowing users to customize experiments. Figure 4 shows the HAMP GUI configuration options.

This GUI allows users to configure six key aspects for running experiments. In (1), the user selects the available hardware accelerator on their computer (CPU, GPU, or FPGA); (2) allows for the selection of one of the five available kernels: Bus speed download, bus speed readback, vector sum, matrix multiplication, and sorting algorithms; (3) specifies the size of the input data to be processed by the selected kernel, for example, if the kernel is selected for Matrix multiplication, a value of 500 would indicate that the kernel will multiply two matrices of 500x500 or if the selected kernel performs Vector sum, a value of 500 would indicate that the kernel will add two vectors of 500 elements each. This parameter allows adjusting the workload and execution time of the experiment, as well as evaluating the performance of the kernel under different input sizes; Option (4) defines the number of times the experiment will be run, this allows obtaining statistically significant results and evaluating performance variability; In option (5), the user initiates the experiment execution, HAMP proceeds to load the data, transfer it to the accelerator, execute the kernel, and collect the data; Option (6) allows the user to load pre-existing data from a file instead of generating random data, this is useful for experiments requiring specific data or for replicating results from previous studies.

Table 6. Specifications of the hardware accelerators for the experiments.

Device	Manufacturer	Clock speed	Memory
CPU i7 9750H	Intel	2.60 GHz	24 GB
GPU UHD Graphics 630	Intel	1.15 GHz	24 GB
GPU GeForce GTX 1650	Nvidia	930 MHz	4 GB
Cyclone V FPGA OpenVINO™ Toolkit	Intel/Terasic	50 MHz	1 GB

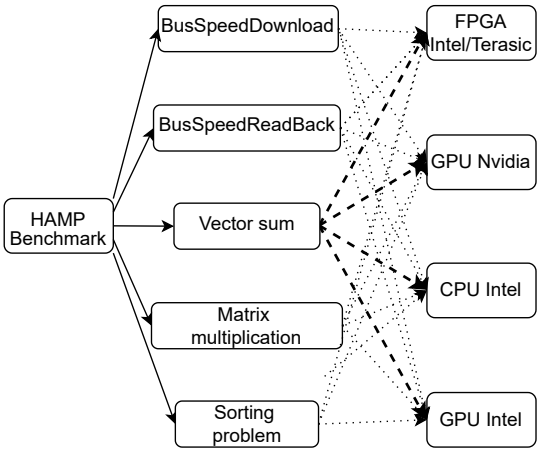

Fig. 5. Experimental evaluation methodology for HAMP benchmark.

4 Experimental Evaluation

To assess the effectiveness of HAMP benchmark in heterogeneous computing, this section presents an evaluation of its performance on three hardware accelerators, focusing on the measurement of bandwidth and kernel speed. Experiments were conducted using five representative parallel kernels - bus speed download, bus speed readback, matrix multiplication, vector sum, and sorting problem- executed on three hardware accelerators (CPU, GPU, and FPGA). Performance was evaluated in terms of bandwidth and kernel speed. The following subsections detail the experimental methodology, present the performance results for each accelerator and kernel combination, and analyze the observed improvements.

4.1 Experimental Setup

Each of the five benchmark kernels (bus speed download, bus speed readback, matrix multiplication, vector sum, and sorting problem) was executed on four devices (an Intel i7 CPU, an Intel Graphics 630 GPU, an Nnvidia GeForce GTX GPU, and an Intel/Terasic FPGA), these devices represent three distinct hardware accelerators: CPU, GPU, and FPGA. For each kernel-accelerator combination, 30 independent runs were performed to obtain robust performance measurements of bandwidth and speed.

Figure 5 shows the experimental evaluation methodology for the HAMP benchmark, and Table 6 presents the specifications of the hardware accelerators used in this study.

4.2 Benchmark Performance

Performance is evaluated in terms of bandwidth and speed, two key metrics for assessing the efficiency of heterogeneous computing. The results, presented by kernel, compare the performance achieved on each of the four devices.

To evaluate the **Vector sum** kernel, the size of the input vectors was systematically varied from 16 elements up to a maximum of 8,388,608 elements, using 20 logarithmically spaced vector sizes. Performance measurements for vector sizes smaller than 16 elements were omitted due to their negligible impact on the results. The overhead of kernel launch and data transfer dominated the execution time for small vectors, making the performance variations negligible.

Thirty independent runs were conducted for each kernel-accelerator combination to ensure statistically significant measurements of bandwidth and speed. The speed results are presented in Table 7.

Figure 6 presents the speed performance results for the Vector sum kernel across the four devices, plotted against vector size. The Intel GPU exhibits significantly higher speed than the other devices, particularly for larger vector sizes, this is due to its architecture, which is optimized for parallel operations. The Nvidia GPU demonstrates lower performance, suggesting lower performance for this specific kernel. Notably, the Intel

Table 7. Speed performance of the Vector sum on four devices

Vector size	CPU Intel (ms)	GPU Nvidia (ms)	GPU Intel (ms)	FPGA Intel/Terasic (ms)
16	0.012	0.015	0.003	0.094
32	0.033	0.016	0.003	0.094
64	0.035	0.016	0.004	0.099
128	0.036	0.016	0.004	0.104
256	0.039	0.017	0.004	0.098
512	0.039	0.017	0.004	0.095
1,024	0.040	0.017	0.004	0.101
2,048	0.042	0.018	0.004	0.103
4,096	0.044	0.021	0.006	0.113
8,192	0.051	0.025	0.008	0.149
16,384	0.057	0.039	0.012	0.200
32,768	0.074	0.074	0.020	0.348
65,536	0.096	0.136	0.037	0.581
131,072	0.172	0.264	0.072	1.101
262,144	0.257	0.516	0.133	2.117
524,288	0.471	1.020	0.283	4.152
1,048,576	0.795	2.027	0.570	8.249
2,097,152	1.650	3.997	1.261	16.418
4,194,304	3.686	8.008	2.420	32.716
8,388,608	6.736	16.017	5.247	65.339

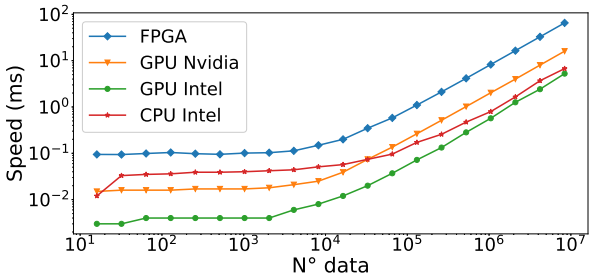

Fig. 6. Speed performance of the Vector sum on four devices

CPU outperforms the Nvidia GPU, especially for larger vector sizes, suggests that for certain configurations, the CPU's processing capacity can be more effective than the Nvidia GPU. The Intel/Terasic FPGA achieves consistent, but lower, performance compared to the other three devices. This may indicate a bottleneck in data transfer or limitations in leveraging the hardware's full potential when using a framework like OpenCL, as opposed to a HDL. These results suggest that the Intel GPU is the most efficient device for the Vector sum kernel when high performance is required, although further analysis is necessary to fully understand the observed performance differences and to determine if this trend holds for other kernels.

Table 8. Speed performance of the Sorting kernel on four devices

Vector size	CPU Intel (ms)	GPU Nvidia (ms)	GPU Intel (ms)	FPGA Intel/Terasic (ms)
4	0.074	0.004	0.005	0.056
8	0.096	0.004	0.006	0.062
16	0.105	0.005	0.009	0.066
32	0.115	0.005	0.011	0.076
64	0.130	0.006	0.019	0.080
128	0.207	0.047	0.027	0.086
256	0.620	0.097	0.098	0.098
512	2.119	0.291	0.400	0.141
1,024	8.212	1.028	1.271	0.431
2,048	32.324	3.950	4.963	1.133
4,096	129.078	15.390	16.312	10.630
8,192	512.248	60.959	61.814	17.251
16,384	2,072.590	255.620	253.887	54.191
32,768	8,255.089	1,027.264	1,043.079	132.394

To evaluate the Vector sum kernel's performance, bandwidth was measured on four different devices. The results are shown in Figure 7. The trend observed in Figure 6 is confirmed, the Intel GPU exhibits significantly higher bandwidth than the other devices, suggesting a greater capacity for efficient data transfer. While the other devices demonstrate adequate performance, they do not achieve the same level of bandwidth. This difference may be attributed to variations in their architecture, memory, or processing capabilities.

To evaluate the performance of the Sorting kernel, a Python script was implemented to generate sets of randomly data for each execution, creating files with vector sizes between 4 and 32,768 elements. Each execution of the kernel on each device used a unique, randomly generated dataset. This approach allowed for more representative performance measurements and avoided the bias that could be introduced by repeatedly using the same, potentially pre-sorted, dataset. Table 8 shows the speed performance of the Sorting kernel on four devices.

Figure 9 presents the speed performance results for the Sorting kernel across the four devices, plotted against input data size (N). For smaller dataset sizes ($N < 10^2$), the Nvidia GPU achieves the highest speed, suggesting greater parallel processing efficiency for smaller data volumes. However, with increasing dataset size ($N > 10^2$), the Intel CPU emerges as the fastest device, suggesting a greater capacity for handling larger

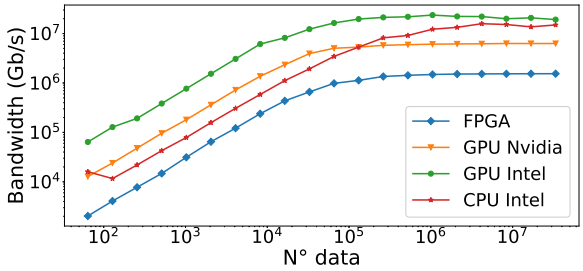

Fig. 7. Bandwidth performance of the Vector sum on four devices

Table 9. Speed performance of the Matrix multiplication on four devices

Matrix size	CPU Intel (ms)	GPU Nvidia (ms)	GPU Intel (ms)	FPGA Intel/Terasic (ms)
2	0.120	0.005	0.050	0.033
4	0.116	0.005	0.050	0.037
8	0.109	0.006	0.060	0.052
16	0.135	0.010	0.017	0.058
32	0.350	0.028	0.089	0.072
64	2.171	0.118	0.627	0.174
128	16.529	1.022	4.900	3.800
256	230	11.981	39.200	9.900
512	2,865	114.766	314	59
1,024	27,877	1,215	2,584	617
2,048	288,863	9,524	22,043	7,304
4,096	2,985,782	76,932	–	216,102

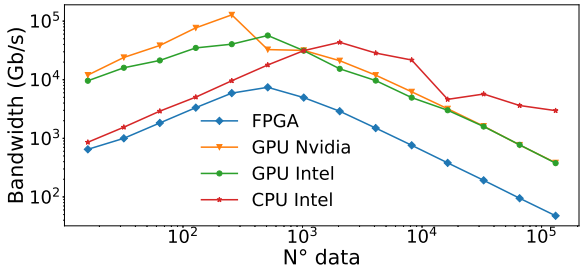

Fig. 8. Bandwidth performance of the Sorting kernel on four devices.

datasets. The Intel GPU and Nvidia GPU exhibit comparable performance, indicating their suitability for sorting tasks, although their speed is slightly lower than the Intel CPU for larger datasets. The Intel/Terasic FPGA initially performs on par with the Intel CPU, but its performance deteriorates with increasing dataset size, possibly due to a bottleneck in data transfer or limitations in

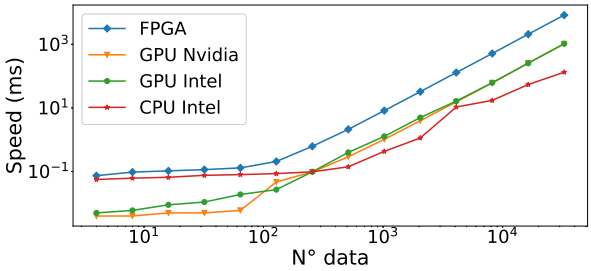

Fig. 9. Speed performance of the Sorting kernel on four devices

maximizing hardware utilization when employing OpenCL instead of HDL.

The trend observed in Figure 9 is confirmed by the bandwidth analysis presented in Figure 8. For smaller datasets ($N < 10^2$), the Nvidia GPU exhibits the highest bandwidth, suggesting greater efficiency in data transfer for smaller data volumes. However, as the dataset size increases ($N > 10^2$), the Intel CPU outperforms other devices and achieves the highest bandwidth. This may indicate a greater capacity of the Intel CPU to handle larger data volumes or a greater efficiency of its architecture for this type of operation.

For the **Matrix multiplication** kernel, random integer values were generated for the elements of each square matrix. The dimensions of the matrices varied from 2x2 up to 4,096x4,096, incremented logarithmically by powers of two. The speed results obtained on the four devices are presented in Table 9.

To evaluate the speed performance of the Matrix multiplication kernel, measurements were conducted on four different devices, Figure 10 illustrates the speed performance achieved, the Nvidia GPU achieves the highest speed across all tested data sizes, suggesting greater parallel processing efficiency. The Intel CPU follows in terms of speed, potentially indicating a greater capacity for handling large datasets. The Intel GPU performs very similarly to the Intel CPU, indicating their suitability for Matrix multiplication operations. The Intel/Terasic FPGA consistently exhibits low performance, which, as discussed earlier, may be

Table 10. Comparison of Mirovia and HAMP compatibility with hardware accelerators for Bus speed download and Bus speed readback kernels

Benchmark	Kernel	FPGA Intel/Terasic	GPU Nvidia	GPU Intel	CPU Intel
Mirovia	Bus speed download		✓		
	Bus speed readback		✓		
HAMP	Bus speed download	✓	✓	✓	✓
	Bus speed readback	✓	✓	✓	✓

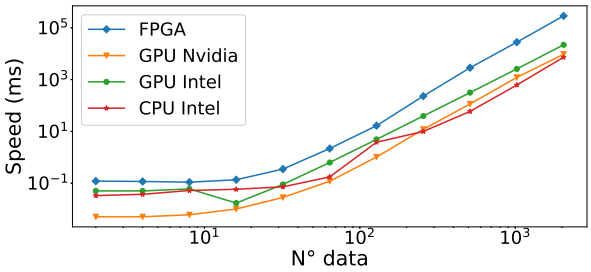

Fig. 10. Speed performance of the Matrix multiplication on four devices

attributed to limitations in maximizing hardware utilization with OpenCL compared to HDL.

Figure 11 complements the results presented in Figure 10, showing the bandwidth achieved by the Matrix multiplication kernel on the four devices. Figure 11 confirms the trend from Figure 10, the Intel/Terasic FPGA exhibits the lowest bandwidth, while the Nvidia GPU is the best option for Matrix multiplication, regardless of matrix size. This reinforces the hypothesis that the Nvidia GPU is particularly well-suited for Matrix multiplication and intensive data transfer operations.

4.3 Performance Comparison and Discussion

For performance comparison, Mirovia was selected as a benchmark due to its recent updates and focus on GPU architectures, this benchmark is widely used in the literature for evaluating GPU performance [16]. The hardware characteristics kernels Bus speed download and Bus speed readback were chosen for comparison. These kernels were successfully executed with HAMP on four devices of three hardware accelerators (CPU, GPU, and FPGA). The bus speed download and readback measurements for

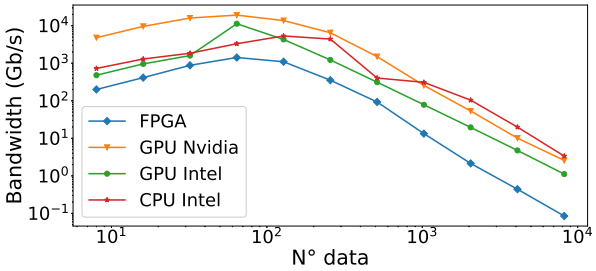

Fig. 11. Bandwidth performance of the Matrix multiplication on four devices

HAMP were performed using OpenCL's *cl_event* objects. A *cl_event* was created for each transfer, and *clWaitForEvents* was used to ensure the transfer's completion before measuring the elapsed time. The elapsed time was then obtained using *clGetEventProfilingInfo* with the *cl_Profiling_Command_End* query. The time taken to transfer a 1 GB data buffer in each direction (download and readback) was measured. Each measurement was repeated 30 times, and the average time was recorded. Mirovia execution was limited to the Nvidia GPU as the benchmark's reliance on CUDA prevents its direct execution on FPGA and CPU architectures. Table 10 presents a comparison of the bus speed download and readback bandwidth achieved by HAMP and Mirovia on the specified hardware.

In summary, HAMP demonstrates its effectiveness as a multi-platform benchmark, leveraging the strengths of diverse hardware architectures. Results show significant improvements in compatibility, supporting four devices of three hardware accelerators, compared to existing benchmarks that are primarily limited to one or two hardware accelerator. Specifically, the Nvidia GPU delivers the highest performance for Matrix multiplication and Sorting problem, while the Intel GPU is ideally suited for Vector sum. The Intel CPU offers competitive performance across all evaluated operations. Although the Intel/Terasic FPGAs performance is lower than expected for a dedicated hardware device, this is attributed to the limitations of OpenCL for hardware description.

5 Conclusion and Future Work

This paper introduces HAMP, a multi-platform benchmark for evaluating application performance across diverse hardware accelerators, including CPUs, GPUs, and FPGAs. HAMP comprises a suite of five kernels (Bus speed download, Bus speed readback, Vector sum, Matrix multiplication, and Sorting problem) and employs two evaluation metrics (speed and bandwidth).

The results demonstrate HAMP's significant improvements in compatibility, supporting four devices (Intel CPU, Intel GPU, Nvidia GPU, and Intel/Terasic FPGA) of three accelerators (CPU, GPU, and FPGA), compared to existing benchmarks that are typically limited to one or two accelerator. Specifically, the Nvidia GPU achieves the highest performance for Matrix multiplication and Sorting problem, while the Intel GPU is ideally suited for Vector sum. The Intel CPU also delivers competitive performance across all five evaluated kernels.

Although the Intel/Terasic FPGA did not achieve the expected performance, this is attributed to the limitations of OpenCL for hardware description compared to HDLs. This suggests the need to explore alternatives such as high-level synthesis tools or optimization techniques to fully realize the potential of these dedicated hardware devices. Collectively, these findings suggest that HAMP can serve as a valuable tool for comparative application benchmarking across a wide range of hardware accelerators, paving the way for the development of high-performance heterogeneous computing.

To extend the HAMP benchmark, future work will focus on several key areas. The plan is to incorporate a wider range of kernels and applications, including, for example, neural networks for machine learning, and image processing algorithms such as Sobel filters. Integration of additional performance metrics relevant to application performance is also planned, including FLOPs, speedup, and energy efficiency. These metrics will provide a more holistic view of performance characteristics.

An investigation of alternative programming frameworks, such as CUDA, and high-level synthesis languages for FPGAs, is also planned to broaden HAMP's applicability and explore different programming paradigms. Furthermore, various software and hardware optimization techniques will be incorporated, including unrolling, pipelining, and floating-point optimization (both single and double precision), to maximize the performance and energy efficiency of the target hardware accelerators.

To facilitate performance analysis and optimization, integration of HAMP with performance analysis and profiling tools like Intel VTune profiler [17] and Nvidia Nsight systems [32] will be explored. This integration will enable developers to easily identify performance bottlenecks and guide the optimization process, providing deeper insights into application behavior and informing optimization decisions.

Finally, validation of HAMP across an even wider range of hardware architectures is a key goal, extending support to diverse heterogeneous and specialized architectures. This includes emerging platforms like quantum accelerators [23, 11] and neuromorphic devices [25]. This broader validation effort aims to ensure HAMP's effectiveness and relevance across the evolving landscape of high-performance computing.

Acknowledgments

This work was supported by TecNM (Mexico) project 22056.25-P. The first author received a CONACYT scholarship (692786) for this research.

References

1. **Asri, M., Malhotra, D., Wang, J., Biros, G., John, L., Gerstlauer, A. (2021).** Hardware accelerator integration tradeoffs for high-performance computing: A case study of gemm acceleration in n-body methods. IEEE Transactions on Parallel and Distributed Systems, Vol. 32, No. 8, pp. 2035–2048.

2. **Castañares, C. (2016).** Desarrollo de aceleradores hardware con técnicas de High-Level Synthesis: exploración de alternativas con paralelismo explícito y ejecución escalable. Master's thesis, Escuela Técnica Superior de Ingenieros Industriales UPM.

3. **Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaer, J., Lee, S., Skadron, K. (2009).** Rodinia: A benchmark suite for heterogeneous computing. IEEE international symposium on workload characterization (IISWC), pp. 44–54.

4. **Chong, X., Ning, X. (2020).** Performance comparison of blas on cpu, gpu and fpga. 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Vol. 9, pp. 193–197.

5. **Coronado-Barrientos, E., Indalecio, G., Garcia, A. (2015).** Study of basic vector operations on intel xeon phi and nvidia tesla using opencl. Annals of Multicore and GPU Programming, Vol. 2, pp. 66.

6. **Danalis, A., Marin, G., McCurdy, C., Jeremy, S., Roth, P., Spafford, K., Tipparaju, V., Vetter, J. (2010).** The scalable heterogeneous computing (shoc) benchmark suite. Association for Computing Machinery, pp. 63–74.

7. **Dinelli, G., Meoni, G., Rapuano, E., Fanucci, L. (2020).** Advantages and limitations of fully on-chip cnn fpga-based hardware accelerator. 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5.

8. **Dongarra, J., Getov, V., Walsh, K. (2018).** The 30th anniversary of the supercomputing conference: bringing the future closer - supercomputing history and the immortality of now. Computer, pp. 74–85.

9. **Escobar, F., Chang, X., Valderrama, C. (2016).** Suitability analysis of fpgas for heterogeneous platforms in hpc. Parallel and Distributed Systems, IEEE Transactions on, Vol. 27, pp. 600–612.

10. **Gautier, Q., Althoff, A., Pingfan, M., Kastner, R. (2016).** Spector: An opencl fpga benchmark suite. 2016 International Conference on Field-Programmable Technology (FPT), pp. 141–148.

11. **Gerlach, T., Knipp, S., Biesner, D., Emmanouilidis, S., Hauber, K., Piatkowski, N. (2024).** Quantum optimization for fpga-placement. 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), Vol. 01, pp. 637–647.

12. **Hagleitner, C., Diamantopoulos, D., Ringlein, B., Evangelinos, C., Johns, C., Chang, R., D'Amora, B., Kahle, J., Sexton, J., Johnston, M., Pyzer-Knapp, E., Ward, C. (2021).** Heterogeneous computing systems for complex scientific discovery workflows. 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 13–18.

13. **HajiRassouliha, A., Taberner, A., Nash, M., Nielsen, P. (2018).** Suitability of recent hardware accelerators (dsps, fpgas, and gpus) for computer vision and image processing algorithms. Signal Processing: Image Communication, Vol. 68, pp. 101–119.

14. **Hoefler, T., Belli, R. (2015).** Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Association for Computing Machinery, pp. 12.

15. **Hu, B., Rossbach, C. (2020).** Altis: Modernizing gpgpu benchmarks. 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 1–11.

16. **Hu, B., Rossbach, C. J. (2019).** Mirovia: A benchmarking suite for modern heterogeneous computing. University of Texas at Austin Austin, USA, pp. 1–10.

17. **Intel (2025).** Intel vtune profiler.

18. **Janik, I., Tang, Q., Khalid, M. (2015).** An overview of altera sdk for opencl: A user perspective. 2015 IEEE 28th Canadian

Conference on Electrical and Computer Engineering (CCECE), pp. 559–564.

19. **Ji, L., Zhihua, W., Danlei, F., Minxu, Z., Xinxuan, W., Xuefeng, Y., Dianhai, Y., Yanjun, M., Feng, Z., Dejing, D. (2023).** Heterps: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. Future Generation Computer Systems, Vol. 148, pp. 106–117.
20. **Jialiang, Z., Jing, L. (2017).** Improving the performance of opencl-based fpga accelerator for convolutional neural network. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 25–34.
21. **Jiang, C. (2020).** Heterogeneous system design and optimisation for embedded vision systems. Master's thesis, School of Electrical and Electronic Engineering.
22. **Kholoud, S., Marwa, E., Adel, E.-Z. (2019).** Optimized implementation of opencl kernels on fpgas. Journal of Systems Architecture, Vol. 97, pp. 491–505.
23. **Li, H., Pang, Y. (2021).** Fpga-accelerated quantum computing emulation and quantum key distillation. IEEE Micro, Vol. 41, No. 4, pp. 49–57.
24. **Lidong, W. (2017).** Heterogeneous data and big data analytics. Automatic Control and Information Sciences, Vol. 1, pp. 8–15.
25. **Lopez, S. (2024).** Fpga-based acceleration for emerging neuromorphic computing paradigms. Ph.D thesis, Universidad Politecnica de Madrid.
26. **Madeira, P. A. (2011).** Evaluation de precision et vitesse de simulation pour des systemes de calcul distribue a large echelle. Ph.D. thesis, Universite de Grenoble.
27. **Meyer, M., Kenter, T., Plessl, C. (2020).** Evaluating fpga accelerator performance with a parameterized opencl adaptation of selected benchmarks of the hpcchallenge benchmark suite. 2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC), pp. 10–18.
28. **Minhas, U., Bayliss, S., Constantinides, G. (2014).** GPU vs FPGA: A Comparative Analysis for Non-standard Precision, Vol. 8405. Springer.
29. **Ndu, G., Lujan, M., Navaridas, J. (2014).** Cho: A benchmark suite for opencl-based fpga accelerators. Technical Report, School of Computer Science, University of Manchester, pp. 1–10.
30. **Nguyen, T., MacLean, C., Siracusa, M., Doerfler, D., Wrigh, N., Williams, S. (2022).** Fpga-based hpc accelerators: An evaluation on performance and energy efficiency. Concurrency and Computation: Practice and Experience, Vol. 34, No. 20, pp. 23.
31. **Nvidia (2020).** Cuda.
32. **Nvidia (2025).** Nvidia nsight systems.
33. **OpenMP (2020).** The openmp api specification for parallel programming.
34. **Peccerillo, B., Mannino, M., Mondelli, A., Bartolini, S. (2022).** A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. Journal of Systems Architecture, Vol. 129, pp. 102561.
35. **Pei-Kuei, T., Tung-Chien, C., Chien-Hung, L., Chih-Yu, C., Jih-Ming, H. (2019).** Heterogeneous computing for edge ai. 2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 1–12.
36. **Petitet, A., Whaley, R., Dongarra, J., Cleary, A. (2018).** Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers.
37. **Reichenbach, M., Holzinger, P., Haublein, K., Lieske, T., Blinzer, P., D., D. F. (2017).** Heterogeneous computing utilizing fpgas. Journal of signal processing systems, pp. 1–13.
38. **Rodriguez, A., Navarro, A., Asenjo, R., Corbera, F., Gran, R., Suarez, D., Nunez-Yanez, J. (2019).** Exploring heterogeneous

scheduling for edge computing with cpu and fpga mpsocs. Journal of systems architecture, pp. 27–40.

39. **Silvano, C., Ielmini, D., Ferrandi, F., Fiorin, L., Curzel, S., Benini, L., Conti, F., Garofalo, A., Zambelli, C., Calore, E., Schifano, S., Palesi, M., Ascia, G., Patti, D., Petra, N., Caro, D. D., Lavagno, L., Urso, T., Cardellini, V., Cardarilli, G., Birke, R., Perri, S. (2024).** A survey on deep learning hardware accelerators for heterogeneous hpc platforms. arXiv:2306.15552, pp. 1–58.
40. **Sung, E., Tong-Wook, S., Tadao, T. (2014).** A faster parallel algorithm for matrix multiplication on a mesh array. Procedia Computer Science, Vol. 29, pp. 2230–2240.
41. **Taylor, L., Artem, C., Erick, O., Karim, A., Jose, N. A. (2017).** A case for better integration of host and target compilation when using opencl for fpgas. FSP 2017; Fourth International Workshop on FPGAs for Software Programmers.
42. **Torrento, A. N. (2018).** Optimization of OpenCL applications on FPGA. Master's thesis, Universitat Politecnica de Catalunya.
43. **Usman, A., Jerry, L., Gautam, S. (2022).** A ml-based resource utilization opencl gpu-kernel fusion model. Sustainable Computing: Informatics and Systems, Vol. 35, pp. 100683. DOI: 10.1016/j.suscom.2022.100683.
44. **Varbanescu, A., Shen, J. (2016).** Heterogeneous computing with accelerators: an overview with examples. 2016 Forum on Specification and Design Languages (FDL), pp. 1–8.
45. **Verma, A., Helal, A., Krommydas, K., Feng, W. (2016).** Accelerating workloads on fpgas via opencl: A case study with opendwarfs. Technical Report, Department of Computer Science, Virginia Polytechnic Institute & State University, pp. 1–9.
46. **Vestias, M., Neto, H. (2014).** Trends of cpu, gpu and fpga for high-performance computing. 2014 24th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–6.
47. **Viet, D., Kamyar, M., Kris, G. (2023).** High-speed hardware architectures and fpga benchmarking of crystals-kyber, ntru, and saber. IEEE Transactions on Computers, Vol. 72, No. 2, pp. 306–320.
48. **Wang, X., Li, X., Leung, V. (2015).** Artificial intelligence-based techniques for emerging heterogeneous network: State of the arts, opportunities, and challenges. IEEE Access, Vol. 3, pp. 1379–1391.
49. **Wei, L., Peng, L. (2021).** An efficient opencl-based fpga accelerator for mobilenet. Journal of Physics: Conference Series, Vol. 1883, pp. 012086.
50. **Zeke, W., Johns, P., Bingsheng, H., Wei, Z. (2017).** Multikernel data partitioning with channel on opencl-based fpgas. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 25, No. 6, pp. 1906–1918.
51. **Zhan, J. (2022).** A benchcouncil view on benchmarking emerging and future computing. BenchCouncil Transactions on Benchmarks, Standards and Evaluations, pp. 1–9.
52. **Ziliang, Z., Rong, G., Qijun, G. (2017).** Marcher: A heterogeneous system supporting energy-aware high performance computing and big data analytics. Big Data Research, Vol. 8, pp. 27–38.

Article received on 12/02/2025; accepted on 16/06/2025.
Corresponding author is Rogelio Valdez.