ISSN 2007-9737

# Cross-Platform Performance Evaluation of Matrix Multiplication: Insights from MKL, cuBLAS, and SYCL

Luis A. Torres[1,*], Carlos J. Barrios H.[1], Yves Denneulin[2]

[1] Universidad Industrial de Santander,
Colombia

[2] Université Grenoble Alpes,
France

{luis.torres, cbarrios}@uis.edu.co, yves.denneulin@grenoble-inp.fr

**Abstract.** Matrix multiplication is a fundamental operation in deep neural network training and scientific computing, optimized through libraries such as Intel MKL [4] and NVIDIA cuBLAS [5]. MKL enhances CPU execution using multithreading and AVX-based vectorization, improving memory bandwidth utilization and computational throughput. Conversely, cuBLAS leverages CUDA's massive parallelism, employing thousands of GPU cores and Tensor Cores to accelerate matrix computations, though Tensor Core usage introduces numerical precision loss. SYCL [23] extends heterogeneous computing capabilities, enabling efficient workload distribution across CPUs and GPUs. This study analyzes execution time, computational efficiency, and power consumption, utilizing PAPI [20] and PERF [9] to evaluate third- and fourth-generation Intel CPUs and selected NVIDIA GPUs. Results indicate that MKL delivers high CPU performance, while SYCL offers an alternative approach with distinct efficiency characteristics. GPU-based benchmarks show that cuBLAS with Tensor Cores achieves maximum throughput but at the cost of precision, whereas cuBLAS without Tensor Cores preserves accuracy with minimal performance trade-offs. These differences highlight the importance of optimization strategies in artificial intelligence and scientific computing, where scaling models and simulations demand efficient, high-performance, and sustainable computation.

**Keywords.** Matrix multiplication, performance evaluation, power consumption, CUDA, MKL, SYCL.

## 1 Introduction

Heterogeneous computing systems have become essential for modern computational workloads, as specialized hardware can significantly enhance performance compared to traditional CPUs. This shift is particularly evident in artificial intelligence (AI), deep learning (DL), and machine learning (ML), where complex mathematical operations demand efficient execution. These systems integrate diverse hardware architectures—CPUs, GPUs, TPUs, and FPGAs—to optimize high-performance workloads while managing power consumption, a critical factor in modern computing trends. Beyond AI, disciplines such as computational fluid dynamics and multimedia processing rely on substantial computational power, requiring advanced mathematical operations and efficient coordination of heterogeneous resources to balance speed and energy efficiency [7].

Among the fundamental operations in scientific computing, matrix multiplication plays a crucial role in high-performance computing (HPC) and ML applications. In deep learning, it is central to model training, particularly within the backpropagation algorithm, which iteratively adjusts neural network weights to minimize learning error [24]. The backpropagation process consists of two key phases: forward propagation, where input data flows through the network to generate predictions, and backward propagation, where errors are computed and distributed across network layers. During backward propagation, matrix multiplication is extensively used to compute gradients, which determine the necessary weight adjustments. These computations involve multiplying weight matrices by error matrices, ensuring efficient gradient updates across large-scale neural networks. Consequently, optimizing matrix multiplication in backpropagation is essential, as it directly impacts training time and computational efficiency.

Since the 1950s, researchers have sought to reduce the complexity of $N \times N$ matrix multiplication beyond the conventional $O(n^{2.81})$, with the best-known algorithms achieving $O(n^{2.37})$ [25]. In HPC environments, matrix multiplication falls under general matrix multiplication (GEMM) routines [2], widely utilized in deep neural networks, often involving matrices exceeding 10,000 elements [21]. To enhance efficiency, modern numerical libraries employ memory optimization techniques, improving data locality and execution speed [17]. However, key challenges persist, including the need for architecture-specific implementations, low-level assembly coding, and delays in optimizing libraries for emerging hardware platforms.

Historically, CPUs were the primary architecture for scientific computing, with algorithms designed to parallelize execution across multiple cores. However, the introduction of GPUs revolutionized the field, significantly outperforming CPUs in highly parallel workloads [3]. To enhance matrix computations, major vendors like Intel and NVIDIA developed optimized numerical libraries. Intel's oneAPI Math Kernel Library (MKL) and NVIDIA's CUDA Basic Linear Algebra Subroutine (cuBLAS) are widely used for efficient numerical computation [14, 16]. Moreover, NVIDIA Tensor Cores further accelerated matrix operations, enabling rapid $4 \times 4$ multiplications per clock cycle, with early implementations featuring 640 Tensor Cores, achieving 125 TFlops/s in mixed precision computing [28]. While mixed precision enhances computational throughput, it may reduce numerical accuracy in HPC applications requiring high precision. Currently, matrix multiplication using Tensor Cores is performed through three primary methods: the CUDA Warp Matrix Multiply Accumulate (WMMA) API, the WMMA-based CUTLASS template library, and cuBLAS GEMM [19].

The increasing complexity of HPC architectures has driven efforts to simplify programming models and improve accessibility. OpenCL[1] was introduced in 2009 as a framework for heterogeneous computing, but its low-level programming complexity limited widespread adoption. In 2014, SYCL[2] emerged as a more accessible alternative, offering an open programming model that facilitates heterogeneous system development while reducing implementation complexity. The SYCL runtime handles critical tasks such as memory allocation and synchronization, utilizing the SMPC (single-source multiple compiler-passes) approach to compile host and device code into a unified executable for CPUs and GPUs [22, 8].

This study compares matrix multiplication algorithms across multiple hardware architectures, measuring execution times, computational performance, and power consumption. It examines the potential for distributed matrix computation in neural network training, emphasizing workload distribution in forward and backward propagation. The evaluation encompasses OpenMP, intrinsic functions (AVX2 and AVX512), CUDA, Intel MKL, and NVIDIA cuBLAS, with and without Tensor Cores, alongside SYCL implementations on both CPUs and GPUs. The paper is structured as follows: Section II discusses related work, Section III outlines the methodology, Section IV presents experimental results, Section V provides a scientific discussion, and Section VI summarizes conclusions and future directions.

## 2 Related Work

Matrix multiplication is a fundamental operation in high-performance computing (HPC), widely used in applications ranging from scientific simulations to advanced deep learning algorithms. Due to its high computational demand, especially when working with large matrices, optimizing its performance and energy efficiency is a crucial challenge in improving modern computing systems. This aspect is particularly relevant in large-scale infrastructures, such as data centers, where power consumption and heat dissipation directly impact operational costs and long-term sustainability. Various optimization strategies have been developed to enhance matrix multiplication execution, addressing aspects from

[1] https://www.khronos.org/opencl/

[2] https://www.khronos.org/sycl/

instruction-level improvements to advanced energy management techniques.

One of the most effective approaches to improving matrix multiplication efficiency is vectorization, which leverages data-level parallelism in modern hardware architectures. By using SIMD (Single Instruction, Multiple Data) instruction sets such as AVX-512 [12] and AVX2 [15], it is possible to execute the same operation simultaneously across multiple data elements, significantly reducing the total number of required instructions [1]. Vectorization can be implemented through two primary methods:

1. Automatic vectorization, where compilers analyze the source code and introduce SIMD instructions when optimization opportunities are detected [13].

2. Manual vectorization, where developers use processor-specific intrinsic functions to exert finer control over instruction generation, potentially leading to greater performance and energy savings [13].

In addition to vectorization, loop-level optimizations play a crucial role in enhancing computational performance. Techniques such as loop unrolling reduce the overhead associated with loop control by expanding the loop body, exposing additional optimization opportunities for compilers [13]. Another essential technique is blocking (tiling), which improves data locality by partitioning matrices into smaller blocks that can fit within higher levels of the cache hierarchy. Working with these blocks increases data reusability in cache, minimizing access to main memory, which is significantly more expensive in terms of latency and energy consumption. The optimal block size is determined by the capacities of L1 and L2 caches [18].

Parallelization is another key approach for accelerating matrix multiplication, distributing computational workloads among multiple cores in modern processors to reduce execution times. OpenMP is widely used for task-level parallelization, enabling workloads to be divided across multiple threads within multi-core architectures. Some hybrid implementations combine OpenMP for task-level parallelism with SIMD for data-level parallelism, achieving optimal performance on multi-core systems. Additionally, OpenCL facilitates parallel execution on both CPUs and GPUs, offering cross-platform portability [1].

In the context of heterogeneous computing, SYCL has evolved into a flexible programming standard for high-performance applications. It provides a high-level C++ abstraction, enabling developers to write portable parallel programs across multiple hardware backends, including NVIDIA, AMD, and Intel GPUs [6]. Recent updates in SYCL 2020 have expanded its adaptability, integrating vendor-specific backends such as CUDA, ROCm, and Level Zero. Additionally, Unified Shared Memory (USM) in SYCL simplifies memory management between host and device, reducing explicit data transfer overhead and improving execution efficiency [6].

Specialized hardware accelerators, such as NVIDIA Tensor Cores, introduce dedicated matrix multiplication units optimized for deep learning workloads. Mixed-precision arithmetic, particularly using fp16, can significantly increase computational throughput [11]. Although lower precision may reduce numerical accuracy, it often provides substantial speed and energy benefits in deep learning applications. Multiword arithmetic has been explored as a technique to extend numerical precision, representing matrices as a sum of multiple lower-precision components, balancing accuracy and performance [11].

Energy-efficient computation is another crucial aspect of matrix multiplication optimization. Dynamic Voltage and Frequency Scaling (DVFS) improves processor performance and energy efficiency by adjusting clock speeds based on workload demands. Running processors at slightly lower frequencies can yield substantial energy savings while incurring minimal performance degradation [27].

Understanding performance metrics is essential for optimizing matrix multiplication in high-performance computing systems. The Performance API (PAPI) framework plays a crucial role in this domain by providing access to hardware performance counters that offer insights into computational efficiency, execution behavior, and bottlenecks. PAPI enables developers to monitor low-level processor events such as cache misses, floating-point operations, memory bandwidth utilization, and instruction throughput, facilitating performance profiling and optimization in both serial and parallel applications [26].

A key extension of PAPI is PowerPAPI, which enhances the original API by enabling power and energy consumption measurement. This integration allows researchers and system architects to correlate performance metrics with power efficiency, offering a comprehensive view of computational trade-offs [26]. PowerPAPI supports data collection from external power meters and built-in telemetry interfaces of modern processors and GPUs, such as Intel's Running Average Power Limit (RAPL) technology [6]. By linking power measurements with computational workload data, PowerPAPI aids in identifying energy-efficient execution patterns and optimizing workload distribution strategies. PAPI's significance extends beyond individual process profiling; its metrics are widely used in performance modeling and dynamic tuning of applications. By leveraging PAPI counters, frameworks such as MuMMI (Multiple Metrics Modeling Infrastructure) perform systematic analysis of execution time, power consumption, and efficiency trends. These analyses contribute to the development of adaptive optimization techniques, where applications dynamically adjust execution parameters—such as scheduling policies, memory management strategies, and frequency scaling—to balance performance and energy constraints effectively [27].

Within heterogeneous computing, PAPI plays an instrumental role in benchmarking CPU-GPU workloads, helping evaluate and compare performance across different architectures. This capability is particularly relevant when assessing SYCL's portability across multiple hardware backends, as PAPI metrics provide quantitative insights into how diverse architectures handle matrix multiplication workloads [6]. Additionally, in large-scale parallel systems, PAPI's multi-threaded profiling capabilities enable developers to fine-tune task distribution, ensuring efficient resource utilization and minimizing execution bottlenecks.

By integrating PAPI with modern optimization frameworks, researchers can quantify computational efficiency, refine matrix multiplication algorithms, and design more energy-conscious execution models. As computing architectures evolve, PAPI remains a cornerstone for performance measurement, guiding advancements in parallel execution strategies and sustainable high-performance computing solutions [26].

Optimizing matrix multiplication requires a multifaceted approach, integrating instruction-level enhancements, memory hierarchy optimizations, parallelization frameworks, specialized hardware accelerators, and power management strategies. Performance modeling tools such as MuMMI and PowerPAPI play a crucial role in refining computational efficiency. Within heterogeneous computing, SYCL continues to evolve as a versatile programming model, enabling portable and high-performance matrix computations. The synergy between hardware and software optimizations will drive future advancements in matrix multiplication performance and energy efficiency, reinforcing its critical role in high-performance computing and AI-driven applications.

## 3 Methodology

Matrix multiplication is a fundamental operation in computational science, playing a key role in numerical simulations, machine learning, and high-performance computing (HPC). Due to its computational intensity and impact on execution efficiency, optimizing matrix multiplication requires a systematic evaluation of its implementations across heterogeneous hardware architectures.

This study analyzes different matrix multiplication techniques, evaluating their execution time, power consumption, and numerical accuracy under controlled experimental conditions. The source code and experimental results are publicly available in the repository: GitHub – MatMul[3], fostering transparency and enabling further research and validation.

[3]https://github.com/alejandrotorresn/MatMul

### 3.1 Matrix Multiplication

Evaluating the efficiency of matrix multiplication requires analyzing how different implementations perform across various hardware architectures. This study examines multiple approaches, including optimized numerical libraries and intrinsic functions, to determine their impact on computational performance. By systematically assessing execution time, power consumption, and numerical accuracy, this methodology provides a comparative perspective on how these techniques behave under different conditions.

This study evaluates both optimization techniques and specialized numerical libraries for matrix multiplication across different computational architectures. For CPUs, the analysis includes vectorization using AVX2 and AVX512, where intrinsics enable precise control over instruction-level execution, optimizing register usage and minimizing memory access latency. Additionally, parallelization using OpenMP is examined, evaluating its scalability in multi-core architectures. On GPUs, the study focuses on high-performance libraries such as cuBLAS, specifically its SGEMM (Single-precision General Matrix Multiplication) implementation. Meanwhile, for CPUs, MKL is evaluated, also utilizing SGEMM to accelerate single-precision matrix multiplication operations.

To ensure statistically stable and reproducible results, each $N \times N$ matrix multiplication was executed ten times, averaging execution times to reduce variability. Power consumption measurements were conducted over twenty executions, compensating for incomplete PAPI metrics on certain platforms.

The study evaluates matrices ranging from $32 \times 32$ to $8192 \times 8192$, covering both small-scale computations, commonly used in embedded systems, and large-scale SGEMM operations, essential for deep neural network training and scientific simulations. This range provides a comprehensive analysis of performance trends across varying matrix sizes and architectures, offering insights into computational scalability and efficiency.

### 3.2 Experimental Environment

This study was conducted across three distinct environments featuring heterogeneous architectures, ensuring a comprehensive and reliable evaluation of matrix multiplication methods across diverse computing systems. Each platform presents a unique configuration, influencing computational efficiency, parallelism, scalability, and power consumption behavior.

#### 3.2.1 Environment 1: Grid5000 Infrastructure

Grid5000[4] is a large-scale testbed designed for high-performance computing (HPC) and experimental research. To ensure diversity in CPU architectures and memory subsystems, three distinct servers with varying hardware configurations were selected. The specific hardware details are provided in Tables 1 and 2.

To maintain a controlled and reproducible software environment, a custom Rocky Linux 9.2 image was deployed using Kadeploy, an automated large-scale system provisioning tool[5]. The kernel version across these servers was Linux 5.14.0, ensuring compatibility with low-level performance counters and power measurement tools. Additionally, the experimental workload was executed in isolated runtime sessions to minimize external system interference, ensuring reliable benchmarking and performance evaluation.

#### 3.2.2 Environment 2: PACCA Cluster (University of Cartagena, Colombia)

The PACCA cluster is an advanced computing facility designed for parallel and GPU-intensive workloads. This platform integrates modern multi-core processors and high-performance GPUs, enabling comparative analysis between CPU and GPU execution models. The detailed hardware specifications for this system are provided in Tables 3 and 4.

[4]https://www.grid5000.fr
[5]https://kadeploy.imag.fr/

### 3.2.3 Environment 3: Intel Developer Cloud

Final validation and comparative evaluations were conducted on the Intel Developer Cloud, which provides access to 4th-generation Intel Xeon processors. The system specifications for this architecture are detailed in Table 5.

Due to PAPI's incompatibility with the Intel Xeon Platinum 8480+ processor, power measurements on this platform were performed exclusively using PERF. Unlike PAPI, which records energy consumption specific to matrix computation kernels, PERF captures total system energy usage, including runtime overhead and background services. This distinction is critical for interpreting power efficiency metrics across different measurement tools.

## 3.3 Software Configuration

To ensure consistency across all evaluations, the same software stack was installed on all three platforms, including:

— Rocky Linux 9.2 as the operating system

— CUDA Toolkit 12.4 for NVIDIA GPU computations

— Intel oneAPI 2023 for CPU-based workloads

— SYCL plugin for NVIDIA support [6].

However, on the Intel Developer Cloud, CUDA was not installed, as this platform does not include NVIDIA GPUs, and thus GPU-accelerated CUDA computations were not performed.

[6]https://developer.codeplay.com/products/oneapi/nvidia/home/

## 3.4 Power Measurement Methodology

Energy consumption was assessed using two monitoring tools, PAPI (Performance API) and PERF, each utilizing distinct methodologies for power measurement:

— PAPI (Performance API) provides access to processor-level power counters via RAPL (Running Average Power Limit) events, as well as GPU energy metrics through NVML (NVIDIA Management Library). RAPL enables detailed energy tracking at the package, core, and DRAM levels on supported CPU architectures, while NVML provides precise GPU power monitoring, capturing consumption at the core, memory, and PCIe levels. However, RAPL is limited to architectures that natively support energy monitoring, meaning it cannot provide data for all processor models (e.g., Intel Xeon Platinum 8480+).

— PERF (Performance Monitoring Tool) operates differently, aggregating total system power consumption, including CPU, memory, and I/O activity. Unlike PAPI, PERF does not exclusively track the energy used by computation kernels, but instead captures broader system-wide power metrics. This approach accounts for additional overhead factors, such as operating system background tasks, process scheduling overhead, and memory allocations. While this method provides a more holistic view of system efficiency, it lacks the granularity needed to isolate computation-specific power usage.

Despite their differences, PAPI and PERF share several fundamental similarities:

1. Performance Monitoring – Both tools track execution performance, integrating power metrics into broader profiling capabilities.
2. Hardware-Based Energy Metrics–Each tool retrieves energy consumption data directly from hardware sensors, ensuring high measurement accuracy.

**Table 1.** GRID5K Hardware Description

| # | CPU Type | CPU | Cores | Freq(GHz) | Mem(GiB) | Freq(MHz) | PCI | TFLOPs | TDP |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Intel Xeon Gold 6126 | 2 | 24 | 2.60 | 192 | 2666 | 3.0 | 3.6864 | 125 |
| 2 | Intel Xeon Gold 6254 | 2 | 36 | 3.10 | 384 | 2933 | 3.0 | 7.1424 | 200 |
| 3 | Intel Xeon Silver 4314 | 2 | 32 | 2.40 | 256 | 3200 | 4.0 | 4.9152 | 135 |

**Table 2.** GPU description for server No. 1

| GPU Type | Cores | T. Cores | freq(MHz) | Mem(GB) | Freq(MHz) | TFLOPs | TDP |
|---|---|---|---|---|---|---|---|
| NVIDIA Tesla V100 | 5120 | 640 | 1230 | 32 | 876 | 14 | 300 |

**Table 3.** PACCA Hardware Description

| # | CPU Type | CPU | Cores | Freq(GHz) | Mem(GiB) | Freq(MHz) | PCI | TFLOPs | TDP |
|---|---|---|---|---|---|---|---|---|---|
| 4 | Intel Xeon Gold 5320 | 2 | 26 | 2.20 | 256 | 3200 | 4.0 | 7.3216 | 185 |
| 5 | Intel Xeon Gold 5315Y | 2 | 16 | 3.20 | 256 | 3200 | 4.0 | 3.2768 | 140 |

**Table 4.** GPU description for server No. 5

| GPU Type | Cores | T. Cores | freq(MHz) | Mem(GB) | Freq(MHz) | TFLOPs | TDP |
|---|---|---|---|---|---|---|---|
| NVIDIA A100 | 6912 | 432 | 765 | 40 | 1215 | 19.5 | 250 |

**Table 5.** Intel Cloud Hardware Description

| # | CPU Type |
|---|---|
| 6 | Xeon Platinum 8480+ |

3. Integration with Benchmarking Workflows – These tools are widely used in HPC benchmarking studies, allowing researchers to correlate energy usage with computational performance across different architectures.

4. Their complementary roles define their relationship:

5. PAPI provides detailed kernel-specific energy measurements for both CPU and GPU workloads (with GPU measurements obtained via NVML).

6. PERF captures total system power consumption, including background processes.

By leveraging both tools in combination, this study ensures a comprehensive analysis of energy efficiency, balancing fine-grained kernel-level measurements with broader workload power profiling across CPUs and GPUs.

## 4 Evaluation and Results

The results of this study are categorized into execution time, power consumption, and computational performance, providing a comprehensive evaluation of matrix multiplication efficiency across different architectures.

Execution time measurements strictly correspond to the matrix multiplication operation itself, except for CUDA and cuBLAS, where the reported time includes both computation and memory transfer overhead. Since GPU implementations require explicit data movement between system memory (RAM) and device memory (VRAM), these transfers contribute to total execution time. Consequently, performance comparisons between CPU and GPU implementations must account for these differences to ensure an accurate evaluation of computational efficiency.

Computational performance was assessed based on matrix size, rather than time-dependent throughput metrics. The metric was computed by determining the number of floating-point operations required for each matrix multiplication, providing a direct comparison of efficiency across varying matrix dimensions. This approach enables a detailed assessment of how different architectures handle computations as matrix sizes increase.

Power consumption metrics were obtained using both PAPI and PERF, each offering distinct perspectives on energy usage:

— PAPI measures power consumption exclusively from the matrix multiplication process, ensuring a direct correlation between computation and energy usage. Additionally, for GPU power measurements, PAPI retrieves energy consumption data via NVML (NVIDIA Management Library), providing detailed metrics on GPU core power draw, memory subsystem consumption, and PCIe interface energy usage.

— In contrast, PERF captures total system-wide energy consumption, including background processes, memory operations, and data loading functions, offering a holistic view of power utilization across all executed code

This distinction is crucial for ensuring that energy efficiency comparisons properly differentiate between computation-specific power usage and overall system consumption.

### 4.1 Performance and Precision

The efficiency of matrix multiplication was evaluated across various computational architectures, comparing CPU and GPU performance, optimization strategies, and numerical accuracy. This section systematically examines execution time, scalability with matrix size, and precision variations across different implementations.

#### 4.1.1 Computational Performance Analysis

This study evaluates matrix multiplication across six Intel processors spanning two generations, assessing the performance of MKL and SYCL for CPU-based computations, alongside cuBLAS and SYCL for GPU executions, tested on two distinct GPU architectures. Additionally, implementations using OpenMP and intrinsic functions leveraging AVX2 and AVX512 were analyzed to assess the effectiveness of high- and low-level parallelization techniques.

Performance metrics are presented in Figures 1 and 2, where: The left section of the image displays computational throughput in TFLOPS/s. The central section shows execution time in milliseconds. The right section quantifies numerical precision using Mean Square Error (MSE), calculated by comparing results against the output from serial matrix multiplication execution.

— CPU Performance: Among CPU-based implementations, MKL and SYCL exhibited optimal execution times, benefiting from efficient multithreading and optimized memory access patterns. OpenMP-based approaches demonstrated significantly higher execution times, approximately twice that of SYCL, leading to their exclusion. Intrinsic functions with AVX2 and AVX512 consistently resulted in the lowest performance, indicating that raw vectorization alone is insufficient without proper memory management.

— GPU Performance: CUDA-based implementations demonstrated superior computational efficiency, with cuBLAS outperforming SYCL due to its tile-based memory optimizations, which enhance parallel execution by reducing global memory access overhead. The Tiled Matrix Multiplication approach further improved performance, yielding lower execution times compared to SYCL, as illustrated in Figure 2. However, SYCL's lambda function was not optimized for GPU execution, as it remained unchanged for both CPU and GPU computations.

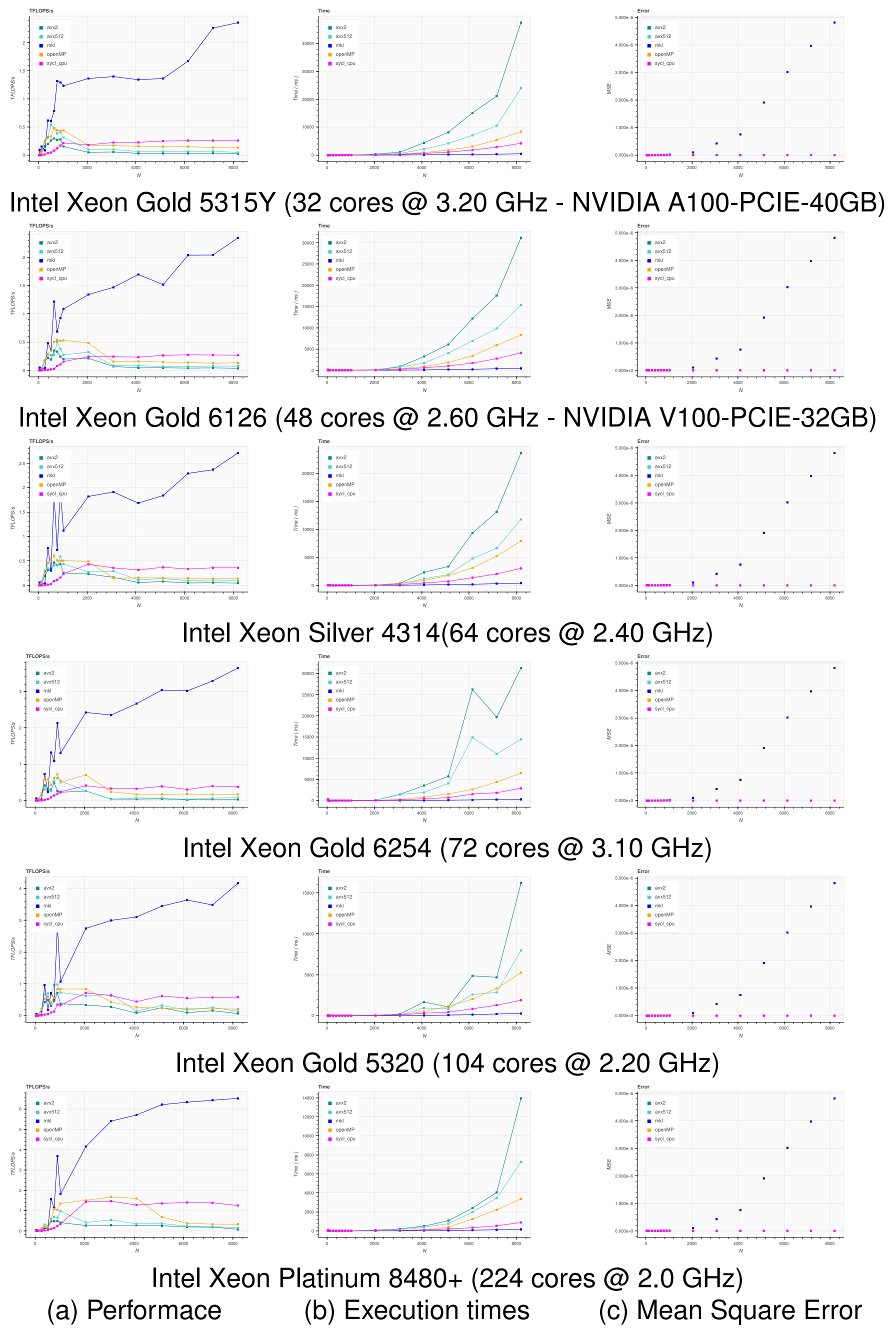


**Fig. 1.** Comparison of (a) Performace (b) Execution times (c) and Error (MSE)

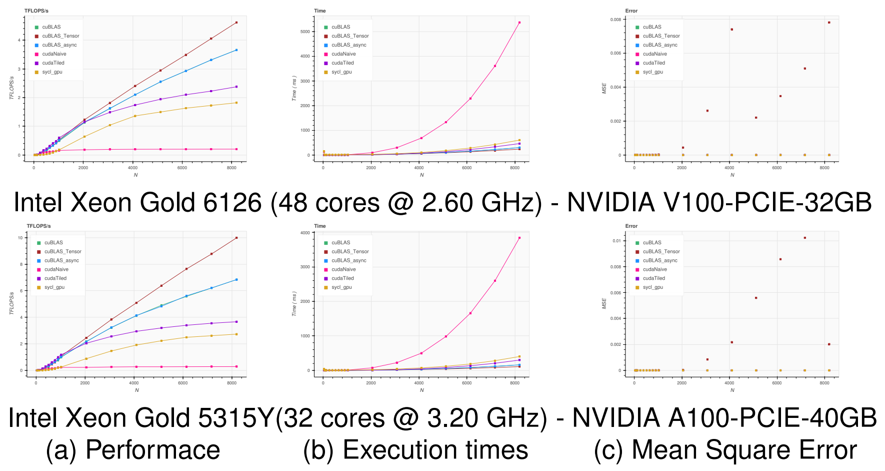


**Fig. 2.** Comparison between (a) Performace (b) Execution times (c) and Error (MSE) of the two NVIDIA architectures evaluated)

### 4.1.2 Portability of SYCL Across CPU and GPU Architectures

SYCL provides a unified programming model, enabling a single kernel to run across CPUs, GPUs, and other accelerators. In this study, no architecture-specific optimizations were applied, ensuring a direct comparison of its portability across different processing units without modifying the lambda function.

While SYCL can be optimized for GPUs, such modifications may reduce efficiency on CPUs, necessitating separate tuning for each architecture. Similarly, optimizing SYCL for CPU execution can negatively impact its performance on GPUs, as CPU-friendly memory layouts, execution ordering, and threading models may not efficiently translate to GPU parallelism.

Unlike cuBLAS, which is tailored for NVIDIA GPUs, SYCL maintains cross-platform compatibility, though it exhibits lower peak performance on specialized hardware. As shown in Figures 3 and 4, cuBLAS achieves higher computational throughput, whereas SYCL provides consistent execution across CPU and GPU environments, emphasizing its flexibility despite performance trade-offs.

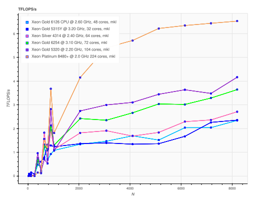
### 4.1.3 Comparison of CUDA Implementations Across GPUs

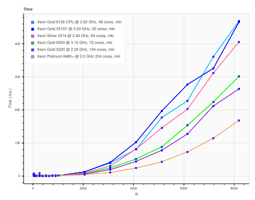
CUDA performance was evaluated across two distinct GPU architectures, comparing execution efficiency under different hardware configurations. Figure 4 illustrates processing times for NVIDIA A100 and NVIDIA V100, demonstrating that A100 consistently outperforms V100, benefiting from advanced memory optimizations and increased computational density. These results highlight the importance of hardware-specific optimizations, as the differences between these two architectures directly impact performance scaling and numerical stability.

### 4.1.4 Numerical Precision Analysis

Numerical accuracy was assessed using Mean Squared Error (MSE), comparing each implementation against a serial reference model. While all tested methods comply with floating-point computation standards, variations in memory representation, execution order, and accumulation precision led to minor numerical discrepancies.

MSE was chosen as the primary metric due to its sensitivity to large errors, its ability to evaluate numerical stability in high-performance computations, and its usefulness in comparing computational efficiency across different architectures. Unlike


(a) Performace


(b) Execution times

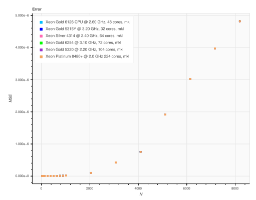
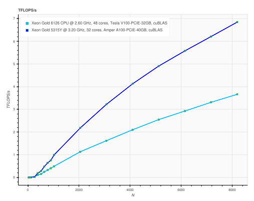
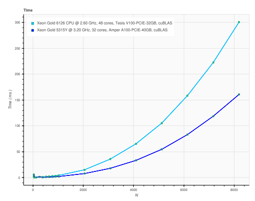

(c) Mean Square Error

**Fig. 3.** Comparison of the execution times of the different processors evaluated using the MKL library


(a) Performace


(b) Execution times

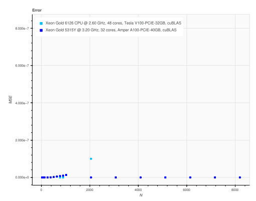

(c) Mean Square Error

**Fig. 4.** Comparison of the execution times of the different GPUs evaluated using the cuBLAS library

metrics such as Mean Absolute Error (MAE), MSE emphasizes significant deviations, providing a more detailed analysis of how accumulated errors affect final results.

Additionally, individual performance across implementations showed:

— MKL exhibited an increasing error trend, though it remained within an acceptable $10^{-6}$ MSE threshold, primarily due to accumulated rounding effects in large matrix computations.

— cuBLAS achieved lower numerical errors, benefiting from tile-based memory optimizations, which reduce rounding inconsistencies in large-scale computations.

— SYCL displayed greater MSE variability, likely due to architecture-dependent rounding mechanisms, affecting cross-platform numerical consistency

### 4.1.5 Impact of Floating-Point Computation on Precision

Precision degradation in floating-point operations occurs due to several factors:

— Loss of significant bits in computations involving high-magnitude values, amplifying accumulated rounding errors.

— Inconsistencies in parallel execution, where the order of floating-point summation influences final results.

— Differences in memory representation, particularly in GPU environments, where storage formats may introduce additional numerical variations.

These findings highlight the trade-off between efficient execution and numerical stability, reinforcing the importance of precision-aware optimizations in high-performance computing applications.

### 4.1.6 Comparison of CPU and GPU Performance

Figure 5 compares Intel Xeon 5320 (104 cores, 2.2 GHz), Intel Xeon Platinum 8489+ (224 cores, 2.0 GHz), and NVIDIA A100, highlighting the best computational results obtained across architectures. While the Xeon 5320 exhibits lower performance, the Platinum 8489+ surpasses NVIDIA A100 when executing cuBLAS, achieving higher computational throughput, though at the cost of increased numerical error due to accumulated floating-point rounding effects.

Figure 6 further illustrates the impact of Tensor Cores in cuBLAS, which significantly enhances NVIDIA A100's execution efficiency, allowing it to exceed Intel Platinum's processing performance. Despite this improvement, the trade-off between execution speed and numerical precision remains, as parallel computations in CPUs and GPUs introduce differences in numerical accumulation and data representation.

These results highlight the impact of architecture-specific optimizations on floating-point stability, emphasizing the balance between computational efficiency and numerical accuracy in high-performance computing environments.

### 4.2 Power Consumption

Energy efficiency is a critical factor in high-performance computing (HPC) and artificial intelligence (AI) applications, where optimizing computational workloads must be balanced with power constraints. This section provides a detailed examination of the power consumption metrics collected using PAPI and PERF for the CPUs and GPUs analyzed in this study.

#### 4.2.1 Measurement Framework and Limitations

The power consumption of each system was recorded using PAPI and PERF, each employing different approaches to energy measurement:

— PAPI: The power consumption data for the CPU and GPU were obtained directly using PAPI, except for the Intel Xeon Platinum 8480+ processor, where RAPL could not be used due to compatibility limitations with this processor generation.

— PERF: All processors were evaluated using PERF, which measured power consumption considering both CPU and RAM usage during matrix multiplication execution. For the GPU, consumption was estimated by summing the CPU and RAM power data obtained from PERF with the GPU power data extracted via PAPI through NVML, providing an overall system energy consumption estimate.

#### 4.2.2 Comparative Power Consumption Analysis

Figure 7 presents a visual comparison of TFLOPs vs. Watts measurements obtained using PAPI (left) and PERF (right) for the CPUs and GPUs analyzed. The observed trends highlight distinct differences between the two measurement tools:

— According to PAPI, MKL exhibited the highest power consumption among CPU-based methods. However, it also achieved the best computational performance compared to other CPU-based approaches, despite not reaching the processor's peak theoretical performance.

— PERF, in contrast, measures total system power consumption, including additional tasks such as memory allocation, array initialization, and data loading, leading to higher aggregated values.

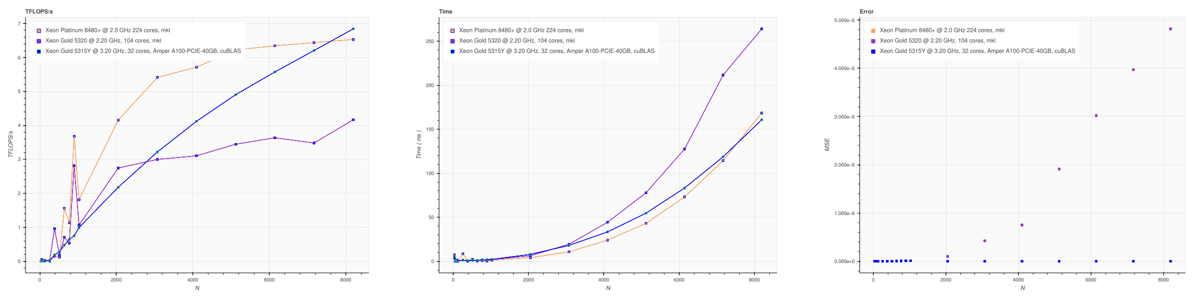
A notable discrepancy appears in the SYCL vs. MKL comparison:

— According to PAPI, SYCL demonstrates lower power consumption than MKL, suggesting a more energy-efficient execution pattern.

— According to PERF, SYCL registers higher overall power consumption, potentially due to additional computational overhead and kernel scheduling inefficiencies

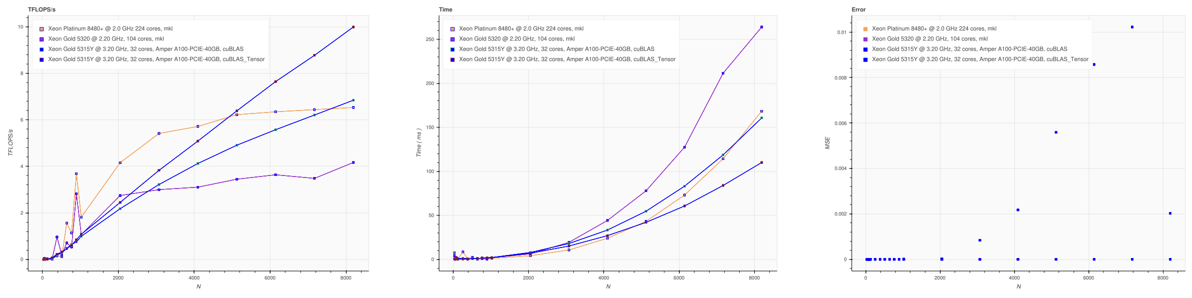
#### 4.2.3 Power Consumption Trends Across Architectures

Among the processors analyzed, the Intel Xeon Platinum 8480+ exhibited the highest power consumption according to PERF metrics, reinforcing the idea that high-core-count architectures require significant energy resources for intensive workloads. However, due to PAPI's incompatibility with this processor, a detailed breakdown of power consumption was not possible, limiting direct comparisons of MKL's efficiency. Figure 7 specifically displays TFLOPs vs. Watts results



(a) Performace (b) Execution times (c) Mean Square Error

**Fig. 5.** Comparison between the best CPU and GPU execution times. Intel Xeon Platinum 8480+, Intel Xeon Gold 5320, and NVIDIA A100-PCIE-40GB GPU. MKL Vs. cuBLAS



(a) Performance (b) Execution times (c) Mean Square Error

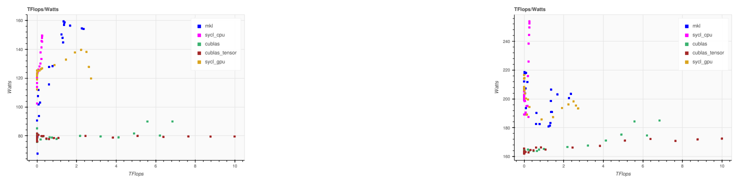
**Fig. 6.** Comparison between the best CPU and GPU execution times. Intel Xeon Platinum 8480+, Intel Xeon Gold 5320, and NVIDIA A100-PCIE-40GB GPU. MKL Vs. cuBLAS with Tensor Cores

obtained via PERF for the Intel Xeon Platinum 8480+, representing the system's total energy consumption, as PAPI RAPL does not support this architecture, preventing a more detailed component-level analysis.

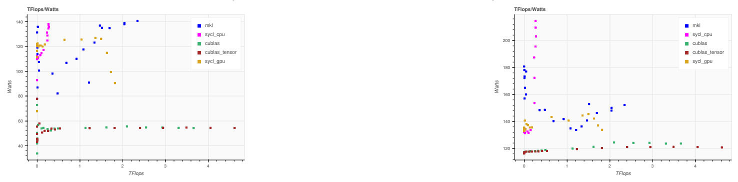
Given these findings, future research should integrate cross-architecture power normalization techniques to ensure fair comparisons between CPU and GPU platforms. The adoption of per-component power tracking methods would enhance the accuracy of benchmarking methodologies, refining efficiency analyses for next-generation HPC and AI workloads.

Figure 8 presents a detailed analysis of the power consumption associated with cuBLAS and SYCL executions on GPUs, considering energy usage across the CPU, memory, and GPU components. According to official NVIDIA documentation, the recorded power consumption for cuBLAS on the V100 and A100 GPUs exhibits comparable values, indicating consistent energy behavior across these architectures. In contrast, SYCL demonstrates higher overall power consumption under typical execution conditions. However, for matrix sizes where $N > 5120$, a notable reduction in energy usage is observed. This decline is attributed to SYCL's architectural design, which prioritizes energy efficiency and portability, as highlighted in the work of Faqir-Rhazoui [10].

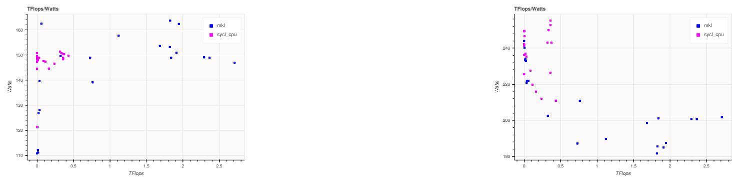
Figure 9 illustrates the power consumption trends observed in MKL executions on the Intel Xeon Platinum 8480+, as well as cuBLAS on the NVIDIA A100 GPU. The results indicate that increasing the number of active cores and operating frequency in the Intel Xeon Platinum 8480+ leads to a significant rise in power consumption, reflecting the direct correlation between computational intensity and energy demand in



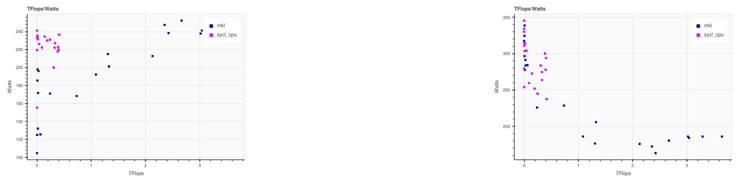
Intel Xeon Gold 5315Y (32 cores @ 3.20 GHz) - NVIDIA A100-PCIE-40GB GPU



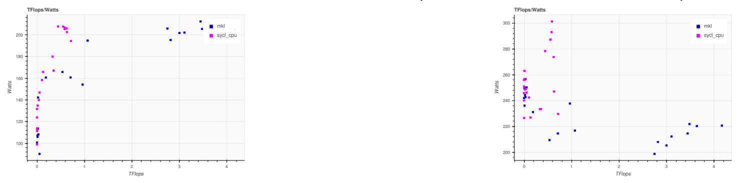
Intel Xeon Gold 6126 (48 cores @ 2.60 GHz) - NVIDIA V100-PCIE-32GB0



Intel Xeon Silver 4314(64 cores @ 2.40 GHz)

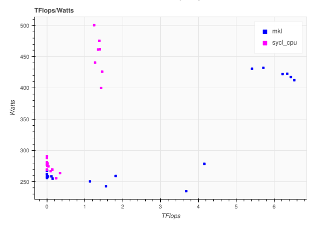


Intel Xeon Gold 6254 (72 cores @ 3.10 GHz)



Intel Xeon Gold 5320 (104 cores @ 2.20 GHz)

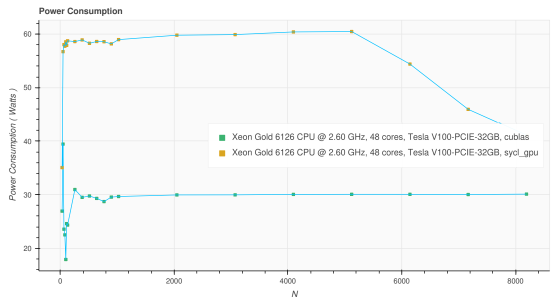
(a) PAPI - Power Consumption (b) PERF - Power Consumption



Intel Xeon Platinum 8480+ (224 cores @ 2.0 GHz)

(c) PERF - Power Consumption

**Fig. 7.** The power consumption results obtained with PAPI are shown on the left, while the PERF results are on the right. The Intel Xeon Platinum 8480+ processor couldn't measure power consumption because PAPI does not have counters available to support this processor



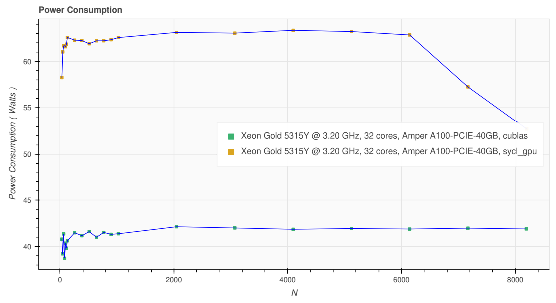


**Fig. 8.** Power consumption of NVIDIA V100 (top) and NVIDIA A100 (bottom). Comparison between SYCL, cuBLAS and cuBLAS with Tensor Cores

CPU architectures. In contrast, the NVIDIA A100 maintains a nearly stable power profile across both small and large matrix computations. This stability suggests that GPU workloads do not generate the same power fluctuations as CPU-intensive tasks, as computational demands are efficiently distributed across the GPU's parallelized architecture.

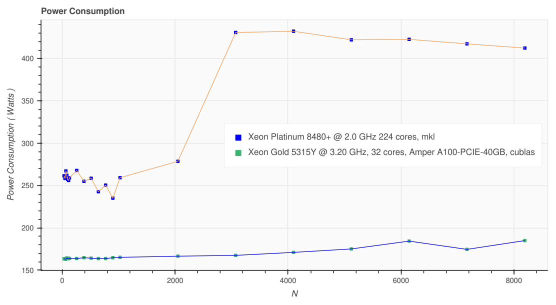


**Fig. 9.** Comparison of power consumption between the Intel Xeon Gold 5320, Intel Xeon Gold 5315Y processors with the MKL library and the NVIDIA A100 GPU with the cuBLAS library

## 5 Discussion

Efficient matrix multiplication in heterogeneous computing systems requires an optimal balance between performance and energy consumption. The experimental findings in this study reinforce the complex interaction between software optimizations, hardware architecture, and resource utilization, emphasizing the deviation between theoretical peak performance and real-world execution metrics.

### 5.1 Impact of Software Optimization and Execution Frameworks

The results presented for MKL, cuBLAS, and SYCL reveal the substantial influence of execution frameworks on computational efficiency and power consumption. While MKL demonstrated high power demand, it also achieved the best computational performance among CPU-based methods, reinforcing the importance of low-level optimizations and vectorization techniques in numerical computing libraries. However, MKL did not reach the theoretical peak performance of the Intel Xeon Platinum 8480+, suggesting the presence of bottlenecks related to memory bandwidth and thread scheduling overhead.

Conversely, SYCL exhibited higher overall energy consumption compared to cuBLAS across GPU architectures. However, a reduction in power demand was observed for large matrix sizes $(N > 5120)$, aligning with SYCL's design principles, which emphasize energy efficiency and portability, as noted in Faqir-Rhazoui's work [10]. These findings highlight the necessity of adaptive workload optimization strategies to mitigate energy overhead while preserving computational throughput.

### 5.2 Hardware Architecture and Computational Efficiency

The analysis of Intel Xeon Platinum 8480+ and NVIDIA A100 provides key insights into the energy-performance trade-offs between CPU and GPU execution. Increasing the number of active cores and operating frequency in the Intel Xeon Platinum 8480+ led to a significant rise in power consumption, reinforcing the direct relationship between computational intensity and energy demand in CPU architectures.

On the other hand, NVIDIA A100 exhibited a nearly constant power profile across matrix sizes, suggesting that GPU workloads distribute computational demands efficiently across parallel cores, preventing excessive energy fluctuations. This behavior contrasts with CPU-based processing, where increasing core utilization introduces nonlinear power scaling effects due to thermal constraints and cache hierarchy bottlenecks.

### 5.3 Future Directions in Energy-Efficient Computing

These findings highlight the necessity for comprehensive performance-energy evaluations to establish optimal execution strategies in scientific computing, artificial intelligence, and large-scale simulations. Future research should address the following key areas:

— Energy-efficient computing models: Developing architectures that minimize power overhead while maximizing computational performance.

— Workload-specific optimization techniques: Implementing dynamic scheduling and power-aware execution strategies tailored to diverse HPC applications.

— Hybrid execution strategies: Integrating CPU-GPU heterogeneous computing paradigms to balance power efficiency and performance scalability.

Achieving maximum computational efficiency without compromising energy sustainability remains a fundamental challenge in high-performance computing research. These results underscore the importance of software-hardware co-design, ensuring that future architectures and execution frameworks achieve the desired equilibrium between speed, scalability, and sustainability.

## 6 Conclusion and Future Work

### 6.1 Conclusion

This study evaluated the comparative efficiency of matrix multiplication on CPUs and GPUs, analyzing the balance between performance, numerical precision, and power consumption across different architectures. The findings indicate that Intel Xeon Platinum 8480+ with MKL outperforms NVIDIA A100 when cuBLAS does not utilize Tensor Cores, as shown in Figure 5, despite exhibiting a higher Mean Square Error (MSE). However, when Tensor Cores are enabled, NVIDIA A100 surpasses CPU performance for matrices larger than 5120×5120, as illustrated in Figure 6, although at the cost of significantly reduced numerical accuracy compared to MKL. These results suggest that fourth-generation Intel CPUs can achieve performance levels comparable to or exceeding high-performance GPUs in matrix multiplication, depending on precision requirements.

Regarding power consumption, obtaining measurements using PAPI on Intel Xeon Platinum 8480+ was not feasible, due to the lack of support for RAPL event monitoring in this processor generation. However, Figure 7 shows that PERF-recorded power values were significantly higher than those obtained with PAPI in MKL executions. This is because PERF accounts for additional factors, such as memory management, array initialization, and data loading, leading to higher overall values. Nonetheless, comparing both platforms, MKL on CPU exhibits higher energy consumption than cuBLAS on GPU, highlighting the difference in energy efficiency between these computational approaches.

The findings indicate that the execution of the backpropagation algorithm on CPU, specifically the matrix multiplication stage, can achieve computational performance comparable to GPUs, and even surpass them under certain conditions, albeit at a higher energy cost. Furthermore, the execution of matrix operations in heterogeneous computing systems and their optimization for diverse workloads remain fundamental research areas in high-performance computing.

This study reinforces the importance of maximizing computational efficiency while maintaining energy sustainability, contributing to the development of energy-efficient computing architectures for hybrid applications, including scientific computing and artificial intelligence.

### 6.2 Future Work

The results obtained open new research avenues that can significantly contribute to advancing high-performance computing. Some key areas to explore include:

— Evaluation of next-generation computing architectures: As new hardware technologies continue to evolve, it will be essential to reevaluate computational efficiency and power consumption, assessing their impact on matrix multiplication across diverse architectures and execution environments.

— Impact of numerical precision on neural network training: A key focus will be investigating how accumulated errors from numerical precision affect execution time in sequential multiplications, particularly in neural network training, where error propagation influences convergence rates and model stability.

— Development of energy-efficient computing models: Future research should explore methods to reduce power consumption while maintaining high computational performance.

— Implementation of dynamic task allocation strategies: Integrating energy-aware libraries and adaptive parallel processing techniques to enhance efficiency in heterogeneous computing environments.

— Advancements in AI-driven optimization techniques: Refining deep learning acceleration frameworks will be essential for driving high-performance energy-efficient computing paradigms.

By addressing these challenges, future research can contribute to the development of scalable, power-efficient, and high-performance computing models, ensuring optimal computational speed and sustainability across diverse applications.

## Acknowledgments

The experiments presented in this study were conducted using the Grid'5000 testbed, supported by a scientific interest group hosted by INRIA, with the collaboration of CNRS, RENATER, multiple universities, and research institutions across France[3]. Additionally, computational resources were provided by the High Performance and Scientific Computing Center at the Universidad Industrial de Santander (SC3UIS)[4] and the Universidad de Cartagena in Colombia, whose advanced computing infrastructure significantly contributed to the execution of this research.

## References

1. **Akoushideh, A., Shahbahrami, A. (2022).** Performance evaluation of matrix-matrix multiplication using parallel programming models on cpu platforms. Research Square. DOI: `10.21203/rs.3.rs-2135830/v1`.

2. **Alman, J., Williams, V. V. (2021).** A refined laser method and faster matrix multiplication. Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, USA, pp. 522–539.

3. **Baratta, I., Richardson, C., Wells, G. (2022).** Performance analysis of matrix-free conjugate gradient kernels using sycl. Proceedings of the 10th International Workshop on OpenCL, Association for Computing Machinery, New York, NY, USA, pp. 1–10. DOI: `10.1145/3529538.3529993`.

4. **Corporation, I. (2023).** Intel math kernel library (intel mkl). `https://software.intel.com/en-us/intel-mkl`.

5. **Corporation, N. (2025).** cublas: Cuda basic linear algebra subroutine library. `https://developer.nvidia.com/cublas`.

[3]https://www.grid5000.fr
[4]http://www.sc3.uis.edu.co

6. **Crisci, L., Carpentieri, L., Thoman, P., Alpay, A., Heuveline, V., Cosenza, B. (2024).** Sycl-bench 2020: Benchmarking sycl 2020 on amd, intel, and nvidia gpus. Proceedings of the 12th International Workshop on OpenCL and SYCL, Association for Computing Machinery, New York, NY, USA, pp. 1–12. DOI: 10.1145/3648115.3648120.

7. **Cussen, D., Ullman, J. D. (2023).** Matrix multiplication using only addition.

8. **da Silva, H. C., Pisani, F., Borin, E. (2016).** A comparative study of sycl, opencl, and openmp. 2016 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), pp. 61–66. DOI: 10.1109/SBAC-PADW.2016.19.

9. **Dresden, T. (2018).** Perf - system and application tracing on linux. http://tu-dresden.de/zih/perf/.

10. **Faqir-Rhazoui, Y., García, C. (2024).** Sycl in the edge: performance and energy evaluation for heterogeneous acceleration. J. Supercomput., Vol. 80, No. 10, pp. 14203–14223. DOI: 10.1007/s11227-024-05957-6.

11. **Fasi, M., Higham, N. J., Lopez, F., Mary, T., Mikaitis, M. (2023).** Matrix multiplication in multiword arithmetic: Error analysis and application to gpu tensor cores. SIAM Journal on Scientific Computing, Vol. 45, No. 1, pp. C1–C19. DOI: 10.1137/21M1465032.

12. **Hemeida, A., Hassan, S., Alkhalaf, S., Mahmoud, M., Saber, M., Bahaa Eldin, A. M., Senjyu, T., Alayed, A. H. (2020).** Optimizing matrix-matrix multiplication on intel's advanced vector extensions multicore processor. Ain Shams Engineering Journal, Vol. 11, No. 4, pp. 1179–1190. DOI: https://doi.org/10.1016/j.asej.2020.01.003.

13. **Jakobs, T., Hofmann, M., Rünger, G. (2016).** Reducing the power consumption of matrix multiplications by vectorization. 2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES), pp. 213–220. DOI: 10.1109/CSE-EUC-DCABES.2016.187.

14. **Khalilov, M., Timoveev, A. (2021).** Performance analysis of cuda, openacc and openmp programming models on tesla v100 gpu. Journal of Physics: Conference Series, Vol. 1740, No. 1, pp. 012056. DOI: 10.1088/1742-6596/1740/1/012056.

15. **Kouya, T. (2021).** Acceleration of multiple precision matrix multiplication based on multi-component floating-point arithmetic using avx2. **Gervasi, O., Murgante, B., Misra, S., Garau, C., Blečić, I., Taniar, D., Apduhan, B. O., Rocha, A. M. A. C., Tarantino, E., Torre, C. M.**, editors, Computational Science and Its Applications – ICCSA 2021, Springer International Publishing, Cham, pp. 202–217.

16. **Krainiuk, M., Goli, M., Pascuzzi, V. R. (2021).** oneapi open-source math library interface. 2021 International Workshop on Performance, Portability and Productivity in HPC (P3HPC), pp. 22–32. DOI: 10.1109/P3HPC54578.2021.00006.

17. **Kuzma, B., Korostelev, I., de Carvalho, J. P. L., Moreira, J. E., Barton, C., Araujo, G., Amaral, J. N. (2023).** Fast matrix multiplication via compiler-only layered data reorganization and intrinsic lowering. Software: Practice and Experience, Vol. 53, No. 9, pp. 1793–1814. DOI: https://doi.org/10.1002/spe.3214.

18. **Lim, R., Lee, Y., Kim, R., Choi, J. (2018).** An implementation of matrix—matrix multiplication on the intel knl processor with avx-512. Cluster Computing, Vol. 21, No. 4, pp. 1785–1795. DOI: 10.1007/s10586-018-2810-y.

19. **Markidis, S., Chien, S. W. D., Laure, E., Peng, I. B., Vetter, J. S. (2018).** NVIDIA Tensor Core Programmability, Performance & Precision . 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE Computer Society, Los Alamitos, CA, USA, pp. 522–531. DOI: 10.1109/IPDPSW.2018.00091.

20. **PAPI (2024).** Papi: A portable interface to hardware performance counters. `https://api.semanticscholar.org/CorpusID:12673152`.

21. **Qin, E., Samajdar, A., Kwon, H., Nadella, V., Srinivasan, S., Das, D., Kaul, B., Krishna, T. (2020).** Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 58–70. DOI: `10.1109/HPCA47549.2020.00015`.

22. **Reddy Kuncham, G. K., Vaidya, R., Barve, M. (2021).** Performance study of gpu applications using sycl and cuda on tesla v100 gpu. 2021 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7. DOI: `10.1109/HPEC49654.2021.9622813`.

23. **Reyes, R., Lomüller, V. (2015).** Sycl: Single-source c++ accelerator programming. International Conference on Parallel Computing, pp. .

24. **Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986).** Learning representations by back-propagating errors. Nature, Vol. 323, pp. 533–536.

25. **Strassen, V. (1969).** Gaussian elimination is not optimal. Numerische Mathematik, Vol. 13, pp. 354–356.

26. **Weaver, V. M., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., Moore, S. (2012).** Measuring energy and power with papi. 2012 41st International Conference on Parallel Processing Workshops, pp. 262–268. DOI: `10.1109/ICPPW.2012.39`.

27. **Wu, X., Lively, C., Taylor, V., Chang, H.-C., Su, C.-Y., Cameron, K., Moore, S., Terpstra, D., Weaver, V. (2013).** Mummi: Multiple metrics modeling infrastructure. 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 289–295. DOI: `10.1109/SNPD.2013.73`.

28. **Yan, D., Wang, W., Chu, X. (2020).** Demystifying tensor cores to optimize half-precision matrix multiply. 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 634–643. DOI: `10.1109/IPDPS47924.2020.00071`.

*Article received on 26/03/2025; accepted on 03/07/2025.*
**Corresponding author is Luis A. Torres.*