Experimental Study of Power Consumption of Basic Parallel Programs

Roblex Nana Tchakoute, Claude Tadonki

Experimental Study of Power Consumption of Basic Parallel Programs

Roblex Nana Tchakoute
Centre de Recherche en Informatique (CRI)
Mines Paris - PSL, Fontainebleau, France
roblex.nana_tchakoute@minesparis.psl.eu

Claude Tadonki
Centre de Recherche en Informatique (CRI)
Mines Paris - PSL, Fontainebleau, France
claude.tadonki@minesparis.psl.eu

***Abstract*—Parallelism is now a standard from the hardware standpoint considering the multicore nature of current processors and their vector processing feature. Beside somehow following Moore's Law and/or running faster, the design of multicore processors was also driven by energy concern. With an increasing number of cores per chip and wider SIMD capabilities, the question of how the corresponding parallelism is related to power consumption is important for energy-aware parallel implementation. The goal of this work is to provide from experimentation some basic insights on the power consumption pattern related to the aforementioned levels of parallelism including accelerated-computing with GPUs.**

***Index Terms*—Energy-Aware Computing, benchmarking, power consumption, parallel computing, multicore processors**

I. Introduction

The rapid evolution of processor architectures has opened in a new era of innovation in high-performance computing (HPC). This era, often referred to as the HPC "Cambrian" explosion [1], has led to a diverse range of processor and micro-architecture designs, each offering specific capabilities for parallel computing. Considering traditional multicore CPUs with vectorized SIMD units and massively parallel GPUs, modern computing systems now can handle the major parallel computing paradigms [2]. This dynamic can be seen from the standpoint of both opportunities and challenges. In fact, while these modern architectures promise impressive computational performance, they also make it more complex to select the optimal configuration for heterogeneous workloads, particularly for large-scale supercomputing facilities.

As computational workloads continue to grow in charge, the demand for energy-efficient computing has become a critical concern [3]. Power consumption in computing systems is now a primary factor in determining the overall efficiency and sustainability of HPC solutions. Considering the increasing about energy and sustainability together with the noteworthy consumption of high-class data centers, reducing the energy consumption of computing systems is crucial for both for operational/maintenance budget and environmental concerns. IEA estimates that by 2026, data centers could use up to 1,050 TWh of electricity, which is equivalent to the total electricity demand of a country like Sweden or Germany [4]. In 2020, ICT activities accounted for approximately 1.8% to 2.8% of greenhouse gas (GHG) emission, which is more than the emissions from the aviation sector [5]. Nowadays, researchers and engineers seriously seek balanced design/implementation approaches that address both computational performance and power efficiency.

Serious advances have been made in the field of parallel computing at all levels. However, traditional benchmarking approaches still mainly focus on execution time, thus without an explicit consideration of energy consumption. Seeking such twofold efficiency in modern HPC systems requires a holistic approach that accounts for both time/space performance and energy. This shift in focus is essential as energy consumption has become a limiting factor in scaling on supercomputers [6], and minimizing power usage without compromising performance is crucial for future developments in HPC [7].

The goal of this work is to investigate the energy efficiency related to the main shared-memory parallel computing paradigms: *SIMD vectorization, multicore/multithreaded*, and *GPU-based parallelism*. Each of these paradigms presents some specific characteristics in terms of how they leverage parallelism and utilize system resources.

About the vector processing (also referred as SIMD), the energy associated with activating wide vector lanes and the power drawn by using vector units might be substantial. Regarding multicore architectures, they offer the ability to concurrently run multiple tasks across several cores, but the overhead of synchronization and contention on shared resources can affect both time performance and power efficiency. GPUs, with their thousands of lightweight cores, excel at tasks that require massive (data) parallelism, but their high power consumption raises challenges for energy-aware computing. These complexities necessitate a thorough examination of how each of these paradigms behaves under different workloads and specific settings, not only about time performance but also about energy consumption.

To perform our energy efficiency evaluation, we consider a set of kernel benchmarks designed to stress the aspects of parallelism we want to investigate w.r.t energy. Memory access patterns, in particular, are known to have a critical impact on both performance and power consumption [8].

The contributions of this work are:

- A comprehensive experimental evaluation of power consumption across three parallel computing paradigms.
- A set of benchmarking kernels to investigate the power consumption model of different hardware aspects.

- Insights about the trade-offs between time performance and power consumption.

The remainder of this paper is organized as follows. In section II, a technical background on parallel models and related work about power consumption analysis. The description of our experimental methodology with measurement approach in provided in section III. Experimental power consumption measurements and related analyses are provided for CPU in section IV and for GPU in section V. Section VI concludes the paper.

II. Parallel models and power consumption

Parallel computing models have evolved to address the growing demand for HPC [9] and large-scale data processing. These models, restricted to a single compute node, include *Single Instruction, Multiple Data* (SIMD) vectorization, multicore/multithreaded architectures, and accelerator-based computing (mainly with GPUs), and they offer different trade-offs in terms of computational speed and power consumption. We examine each of them from the energy perspective.

A. SIMD Vectorization

The *SIMD* model allows the same operation to be performed on multiple data points simultaneously. It is commonly implemented on modern processors through instruction sets such as Intel's AVX or ARM's NEON [2]. While SIMD vectorization can significantly improve computational efficiency by processing multiple data in a single instruction cycle, the associated power consumption is likely to increase due to a simultaneous utilization of multiple functional units within the processor. However, since there a single instruction (no explicit instruction dispatch), there is a potential power saving compared to scalar execution [10]. SIMD power consumption scales with the vector width and the complexity of instructions. For example, using 256-bit or 512-bit wide SIMD instructions may lead to higher power draw compared to narrower 128-bit instructions [11]. Additionally, as idle lanes still consume power, an under-utilized of the vector lanes (e.g., due to data misalignment or irregular data sizes) reduces the overall energy efficiency benefit.

B. Multicore Architectures

Multicore architectures, in which multiple CPU-cores are integrated onto a single chip, coupled with multithreading, where multiple threads are concurrently executed on the cores, represent a more general-purpose form of shared-memory parallelism. Each core or thread can handle independent tasks, or work on different parts of the same problem in parallel, which is highly advantageous for tasks with parallelizable workloads such as *scientific simulations, data analytics*, and *machine learning training* [12].

While this parallelism can improve overall performance, there might be a power consumption increase due to the overhead of the activity of additional cores. In fact, each active core contributes to the total power consumption. However, despite the higher power draw, multi-threading can improve the overall energy efficiency because it reduces idle times and allows tasks to complete faster, thereby lowering the total energy following its correlation with the execution time [13].

C. GPU-Based Computing

GPU (Graphics Processing Units) is the major accelerator unit tailored for massively parallel computation [14]. GPUs are highly optimized for data-parallel tasks through a model known as *Single Instruction, Multiple Threads* (SIMT), where many threads execute the same instruction across different data elements.

While GPUs are really faster on tasks with adequate structure, energy efficiency is still a genuine concern. For example, power consumption of Nvidia H100 is about 700W (400W for A100) and servers like DGX-H100 have eight H100 [15]. With this kind of systems, power consumption is clearly a crucial aspect. Some studies analyze the energy consumption and runtime performance of GPU libraries (like TensorFlow and Pytorch) [16] and programming models (likes OpenCL and CUDA) [17]. These studies conclude that *tuning for time performance generally improves energy efficiency*, specific optimizations are necessary to achieve optimal energy efficiency. These insights are crucial for developers aiming to optimize both performance and energy consumption in GPU-accelerated HPC applications.

III. Measurement approach and tools

The main performance metrics we have evaluated are *GFLOPS* (for floating point operations) and *GB/s* (for memory bandwidth). Regarding energy, CPU and GPU measurements are recorded separately using an appropriate tool for each platform following the testbed described in Table I.

For CPU (resp. GPU) benchmarking, we use C (resp. Python) codes. Each experiment is repeated five times to ensure consistency and to account for variability introduced by the operating system (OS) activity, then the mean value is calculated and considered.

A. Benchmarking selection

For benchmarking, we selected six common computing kernels with different structure *(compute-bound, memory-bound, or mixed-bound operations)*.

Generalized Matrix Multiplication (GEMM): a classical linear algebra kernel that is compute-bound, it is widely used in many scientific/technical applications.

Sparse Matrix-Vector Multiplication (SPMV): also a linear algebra kernel likely memory-bound because of its sparse implementation (irregular memory access pattern).

Streaming Vector TRIAD (TRIAD): commonly used as memory bandwidth benchmark to measure the memory throughput through operations on large vectors, thus another memory-bound kernel.

7-point 3D stencil computation (Stencil): a mixed kernel that combines commonly used in the modeling of physical phenomena like *heat diffusion* and *fluid dynamics*.

Monte Carlo Pi Estimation (Monte Carlo): also a compute-bound kernel that is widely used in *stochastic simulations* and *financial modeling*.
Generalized Euclidean Distance from Origin (DIST): a mixed workload that combines vectorized memory access and floating-point operations, widely used in AI for *clustering* and *k-nearest neighbors* algorithms.

B. CPU Benchmarking and Instrumentation

For CPU-based parallel benchmarking, we used the *LIKWID* tool suite [18] for code instrumentation and performance profiling. LIKWID offers a comprehensive set of features for monitoring performance counters, memory traffic, and energy consumption in modern processors, making it an ideal choice for benchmarking SIMD and multithreaded programs. Its MakerAPI feature allows to profile a specific section of the input program, thereby enabling fine-grained collection of energy-related data.

With a given code, we analyse three versions:

- **Scalar (Sequential)**: Baseline version, scalar and single-threaded.
- **Multithreaded (OpenMP)**: Multi-threaded version derived from the baseline using OpenMP (mainly on loops).
- **Vectorized (SIMD)**: SIMD version implemented from the baseline using SSE and AVX intrinsics.

We use gcc compiler with `-O0` flag so as to prevent any optimization that would hinder our observations. We use the `likwid-perfctr` module to capture performance counters and energy consumption at runtime. For memory traffic and cache behavior, we rely on event groups such as `MEM_SP` (on Intel) and `MEM1`, `MEM2` (on AMD). Energy consumption is recorded via the `ENERGY` group, providing detailed power metrics for the CPU cores. Thread affinity is controlled using `likwid-pin`, ensuring that threads are properly pinned to the CPU-cores in a one-to-one binding.

C. GPU Benchmarking with EA2P

For GPU benchmarking we used Python-based implementations of the same kernels plus two additional kernels (*3D stencil* and *Monte Carlo pi estimation*). For energy measurement, we used EA2P (*Energy-Aware Application Profiler*) [19], an energy profiler tool that can handle GPU-accelerated applications (unlike likwid). Considering Python codes allow to leverage popular GPU-accelerated frameworks such as `CuPy`, `JAX`, `TensorFlow`, and `PyTorch`.

EA2P is used to gather both energy and running time of GPU code blocks at desired granularity. We compute performance metrics like GFLOPS and GB/s by dividing the data size and the FLOPs by the measured running time (we roughly assume that each data is accessed once).

By leveraging the advanced capabilities of LIKWID for CPU instrumentation and EA2P for GPU profiling, we ensure that our methodology captures both performance and energy consumption with a good accuracy.

TABLE I
PLATFORM CHARACTERISTICS (TAKEN FROM [20] AND [21])

Name	Chirop	Chuc
CPU model	Intel Platinum 8358	AMD EPYC 7513
Clock speed	2.6 GHz	2.6 GHz
Turbo Speed	Up to 3.4 GHz	Up to 3.65 GHz
Physical Cores	32 (Threads: 64)	32 (Threads: 64)
L1 iCache	1,024KB 8-way set	1,024KB 8-way set
L1 dCache	1,536KB 12-way set	1,024KB 8-way set
L2 Cache	40MB 20-way set	16MB 8-way set
L3 Cache	48MB 12-way set	128MB 16-way set
DRAM Memory	512GB DDR4-3200	512GB DDR4-3200
CPU TDP	250W	200W
GPU Model	/	Nvidia A100 SXM4
GPU TDP	/	400W
GPU Memory	/	40GB HBM2
Data precision	Single	Single
SIMD extensions	SSE, AVX, AVX512	SSE, AVX
Operating System	Linux Debian 5.10.209	Linux Debian 5.10.209

IV. BASIC ANALYSIS FOR X86 CPU ARCHITECTURES

For all the findings in this section, **"acc."** means acceleration, **"tot."** is total (i.e. values gathered via hardware counters), **"app."** is the effective value of the metric (i.e. without the background overhead), and **"PKG"** is the CPU package domain.

A. Power consumption in idle state

The baseline energy consumption of the system (i.e., when no program is running except the OS) is expected to be proportional to time (i.e. constant power) because of several factors including thermal ones in addition to OS activity. We used a *sleep* program (written in C). The objective is to check the stability of the system in the idle mode so as to consider that energy overhead in our measurements. If the linear relation of the energy in idle mode holds, then we will consider the formulas in equation (1).

$$\begin{cases} E_{idle}(t) = \alpha + \beta \times t \\ E_{application} = E_{measured} - E_{idle}(t_{application}) \end{cases} \quad (1)$$

Where

- $E_{measured}$ is the measured energy,
- $E_{application}$ is the effective energy consumed by the application,
- α and β are the linear regression coefficients (they are platform-dependent),
- $t_{application}$ is the elapsed time of the application.

Our results as displayed in Table II shows that power is constant in the idle mode, with a little overhead at the beginning (first second) due to the loading of the sleep program itself. The Intel Ice Lake SP CPU consumes approximately 50W on average in idle state, which represents 20% of its theoretical TDP (250W). The AMD Zen 3 CPU consumes around 56W, which is about 28% of its TDP (200W). Additionally, the RAM package on the Intel system draws a significant amount of power in idle state, about 9-10W. Note that this RAM power consumption is associated with approximately 2.5GB/512GB

(1.59GB/504GB on AMD) of memory usage by the system in idle state.

Using a linear regression on our measurements in idle state (see table II), we get the relations shown in equation 2). As previously explained, we obtain the effective energy by subtracting this overhead to the measured values (See Tables IV, V, VI and VII).

$$\begin{cases} E_{idle-CPU-Intel}(t) = 4.44 + 47.95 \times t \\ E_{idle-RAM-Intel}(t) = 0.63 + 9.41 \times t \\ E_{idle-CPU-AMD}(t) = 1.88 + 56.14 \times t \end{cases} \quad (2)$$

With RMSE = 6.36 and R^2 = 1.00 for Intel CPU model; RMSE = 0.49 and R^2 = 1.00 for Intel RAM; and RMSE = 8.65 and R^2 = 0.9999 for AMD model

TABLE II
POWER-ENERGY OF SLEEP TEST FOR IDLE POWER CONSUMPTION FOR SINGLE CORE USAGE

runtime [s]	1	2	4	8	16	32	64
AMD							
power PKG [W]	57.73	55.60	55.47	55.62	56.52	56.78	56.02
energy PKG [J]	57.74	111.22	221.89	444.98	904.45	1817.08	3585.92
Intel							
power PKG [W]	52.22	49.11	49.68	49.07	48.48	47.64	48.10
energy PKG [J]	52.22	98.22	198.71	392.56	775.74	1524.49	3078.67
power RAM [W]	9.97	9.69	9.64	9.60	9.41	9.42	9.43
energy RAM [J]	9.97	19.38	38.55	76.78	150.53	301.36	603.49

We now consider the same sleep test on multiple cores (using OpenMP) in order to assess the effectiveness of power-saving features. We assume that CPU cores are disabled or in the power-saving mode when no workload is running. This test aims to validate the potential power variation caused by the creation of multiple processes pinned to CPU hardware cores, even if they are not handling any workload.

Table III shows the results of our measurements on a multicore context as previously explained. The increase in consumed energy is not strictly linear with respect to the number of active cores with shorter runtimes. However, the average power is more stable as the running time increases, and the overall multi-threaded power consumption becomes closer to that of single-thread power consumption on both Intel and AMD. Applying the linear regression on the data from Table III yields equation 3) and indicates that the linear model holds in this case as well.

$$\begin{cases} E_{idle-CPU-Intel}(t) = 3.87 + 48.36 \times t \\ E_{idle-RAM-Intel}(t) = -0.0003 + 9.67 \times t \\ E_{idle-CPU-AMD}(t) = 1.67 + 56.50 \times t \end{cases} \quad (3)$$

With RMSE = 6.23 and R^2 = 0.9977 for Intel CPU model; RMSE = 0.5599 and R^2 = 0.9995 for Intel RAM; and RMSE = 3.75 and R^2 = 0.9994 for AMD model

For our multicore measurements and analyses, we will proceed similarly with using the formulas of equation (2) to get the effective consumption of our parallel benchmarks.

TABLE III
IDLE POWER CONSUMPTION WITH MULTIPLE CORES

Sleep[s]	#threads	AMD		Intel			
		Energy PKG [J]	Power PKG [W]	Energy PKG [J]	Power PKG [W]	Energy DRAM [J]	Power DRAM [W]
1	seq	57.74	57.73	52.22	52.22	9.97	9.97
	2	56.21	56.20	53.27	53.26	9.71	9.71
	4	59.24	59.23	49.41	49.41	9.78	9.78
	8	60.94	60.92	49.71	49.71	9.38	9.38
	16	59.25	59.24	50.98	50.97	9.63	9.63
	32	62.55	62.54	57.17	57.16	9.72	9.72
2	seq	111.22	55.60	98.22	49.11	19.38	9.69
	2	113.62	56.80	100.36	50.18	19.24	9.62
	4	111.83	55.90	93.00	46.50	18.87	9.43
	8	112.10	56.05	99.75	49.87	19.11	9.55
	16	116.76	58.37	103.62	51.81	19.32	9.66
	32	118.36	59.17	101.09	50.54	19.51	9.75
4	seq	221.89	55.47	198.71	49.68	38.55	9.64
	2	228.46	57.11	190.24	47.56	38.13	9.53
	4	224.67	56.17	195.04	48.76	38.64	9.66
	8	227.36	56.84	198.30	49.57	37.73	9.43
	16	225.97	56.49	195.69	48.92	39.09	9.77
	32	231.96	57.98	218.63	54.66	40.70	10.18
8	seq	444.98	55.62	392.56	49.07	76.78	9.60
	2	452.97	56.62	376.29	47.03	76.84	9.60
	4	451.19	56.39	392.11	49.01	77.48	9.68
	8	461.09	57.63	387.63	48.45	77.23	9.65
	16	461.28	57.65	392.26	49.03	78.31	9.79
	32	453.65	56.70	399.56	49.94	77.45	9.68

B. Mono-core time-energy measurements and analysis

The results in Tables V and IV show that *time* and *energy* (including CPU and DRAM for Intel) are highly correlated on both platforms. Considering this correlation between time and energy, energy efficiency improves when the vectorization effectively reduces the execution time without a substantial increase in power consumption, which seems to be what we observed. However, applications that heavily use SIMD computing units are more sensitive to power consumption increases on AMD Zen 3 compared to Intel Ice Lake SP. This is more visible with DIST time-energy acceleration gap (2.03 time speedup vs 1.79 energy speedup). This behaviour might depend on the CPU architecture, as it was not observed on Intel, even with DIST.

TABLE IV
TIME AND ENERGY MEASUREMENTS OF SIMD ON AMD

Code version	Time [s]	acc. (time)	Energy PKG[J]	Energy E_{app}	acc. PKG	acc. E_{app}
SPMV_COO						
seq	14.26	1.00	1108.50	328.50	1.00	1.00
sse	15.22	0.94	1157.10	307.77	0.96	1.07
avx	11.32	1.26	897.72	259.14	1.23	1.27
TRIAD						
seq	7.17	1.00	547.18	144.32	1.00	1.00
sse	5.84	1.23	433.66	106.94	1.26	1.35
avx	3.19	2.25	251.31	71.77	2.18	2.01
GEMM						
seq	124.83	1.00	9160.88	2201.11	1.00	1.00
sse	46.95	2.66	3469.80	841.02	2.64	2.62
avx	29.02	4.30	2151.76	518.80	4.26	4.24
DIST						
seq	3.44	1.00	289.16	92.74	1.00	1.00
sse	2.96	1.16	268.70	100.46	1.08	0.92
avx	1.70	2.03	161.99	64.63	1.79	1.43

TABLE V
TIME AND ENERGY MEASUREMENTS OF SIMD ON INTEL

code	time		energy CPU [J]		CPU acc.		energy RAM [J]		RAM acc.	
ver.	[s]	acc.	PKG	app.	PKG	app.	tot.	app.	tot.	app.
SPMV_COO										
seq	9.91	1.00	991.37	476.01	1.00	1.00	164.17	70.02	1.00	1.00
sse	10.74	0.92	1070.30	511.83	0.93	0.93	175.74	73.72	0.93	0.95
avx	9.17	1.08	920.92	443.84	1.08	1.08	153.05	65.89	1.07	1.06
DIST										
seq	6.45	1.00	663.17	340.64	1.00	1.00	94.83	33.55	1.00	1.00
sse	4.52	1.43	453.75	227.94	1.46	1.49	66.98	24.08	1.42	1.39
avx	2.49	2.59	253.93	129.32	2.61	2.63	38.16	14.49	2.48	2.32
GEMM										
seq	116.56	1.00	11943.84	6116.03	1.00	1.00	1606.94	499.66	1.00	1.00
sse	56.08	2.08	5662.58	2858.58	2.11	2.14	774.71	241.95	2.07	2.07
avx	29.14	4.00	2920.22	1463.07	4.09	4.18	401.29	124.44	4.00	4.02
TRIAD										
seq	5.43	1.00	544.94	273.55	1.00	1.00	85.12	33.56	1.00	1.00
sse	3.38	1.61	342.35	173.20	1.59	1.58	55.58	23.44	1.53	1.43
avx	2.30	2.36	237.01	122.15	2.30	2.24	39.79	17.97	2.14	1.87

C. Multi-core time-energy measurements and analysis

The results in Tables VI and VII show that for both architectures, time and energy efficiencies follow similar tendencies with different slopes (unlike the single-core case). We can say that there is a significant power overhead when using more cores. Thus, the energy speedup is noticeably lower that time speedup (by factor 2 for some cases like SPMV_COO with 32 cores on both INTEL and AMD). In all cases, time and energy efficiencies match when using up to 8 cores, and a serious gap appears with 16 and 32 cores. We think that this phenomenon will generally apply with a large number of cores and get worse with NUMA configurations. Since we did not see this the idle state investigation, we can strongly presume that there is a significant energy cost associated to managing multithreading execution including OS orchestration and all associated hardware mechanism like cache coherency, bus contention, synchronization (if any).

D. Memory power consumption

Our results in Figures 1 and 2 confirm a clear correlation between memory bandwidth and power consumption. Higher bandwidth leads to higher power draw, but energy consumption can be reduced because of lower data transfer time. For memory-bound applications, the benefit is trivial, otherwise the energy efficiency will really depend on the balance between computation energy and memory accesses energy. As expected, for compute-bound applications like GEMM, the energy related to memory accesses is proportional to the overall energy whose also proportional to the execution time.

On INTEL, although the memory bandwidth increases for GEMM, the overall RAM power consumption remains relatively constant (see figure 1.d). In addition, data traffic decreases significantly (from 2.92GB with 2 threads to 0.98GB with 32 threads). Similarly, SPMV data traffic tends to slightly decrease with more threads (from 83GB to 74GB) and remains constant with DIST and TRIAD. There is a strong correlation between bandwidth and power even with large volumes of data.

TABLE VI
MULTITHREAD TIME AND ENERGY MEASUREMENTS ON AMD

#threads	Time	acc.	Energy	Energy	acc.	acc.
	[s]	(time)	PKG[J]	E_{app}	PKG	E_{app}
SPMV_COO						
seq	14.26	1.00	1108.50	328.50	1.00	1.00
2	5.78	2.47	544.11	214.29	2.04	1.53
4	3.22	4.43	338.55	157.49	3.27	2.09
8	3.07	4.64	327.58	156.87	3.38	2.09
16	1.87	7.65	227.27	122.29	4.88	2.69
32	1.63	8.72	229.85	122.81	4.82	2.67
TRIAD						
seq	7.17	1.00	547.18	144.32	1.00	1.00
2	3.87	1.85	334.74	114.63	1.63	1.26
4	2.13	3.36	215.68	97.38	2.54	1.48
8	1.82	3.93	200.93	97.04	2.72	1.49
16	0.99	7.26	135.64	74.24	4.03	1.94
32	0.68	10.60	110.45	71.09	4.95	2.03
GEMM						
seq	124.83	1.00	9160.88	2201.11	1.00	1.00
2	64.15	1.95	5248.30	1658.71	1.75	1.33
4	32.22	3.87	3039.53	1232.53	3.01	1.79
8	25.44	4.91	2468.35	1049.06	3.71	2.10
16	12.98	9.62	1562.17	849.76	5.86	2.59
32	7.14	17.49	1136.23	732.24	8.06	3.01
DIST						
seq	3.44	1.00	289.16	92.74	1.00	1.00
2	2.03	1.70	177.53	75.66	1.63	1.23
4	1.33	2.59	142.91	69.05	2.02	1.34
8	1.15	2.99	136.36	66.27	2.12	1.40
16	0.90	3.81	118.28	62.61	2.44	1.48
32	0.98	3.50	132.67	77.22	2.18	1.20

TABLE VII
MULTITHREAD TIME AND ENERGY MEASUREMENTS ON INTEL

#th	time		energy CPU [J]		CPU acc.		energy RAM [J]		RAM acc.	
	[s]	acc.	PKG	app.	PKG	app.	tot.	app.	tot	app.
SPMV_COO										
seq	9.91	1.00	991.37	476.01	1.00	1.00	164.17	70.02	1.00	1.00
2	5.63	1.76	597.85	305.27	1.66	1.57	105.59	52.14	1.55	1.34
4	2.96	3.35	349.97	196.12	2.83	2.44	66.35	38.24	2.47	1.83
8	1.66	5.97	230.20	143.83	4.31	3.32	46.37	30.59	3.54	2.29
16	0.99	9.98	175.81	124.16	5.64	3.85	34.74	24.81	4.73	2.82
32	0.71	14.03	159.36	115.57	6.22	4.14	29.40	22.33	5.58	3.14
DIST										
seq	6.45	1.00	663.17	340.64	1.00	1.00	94.83	33.55	1.00	1.00
2	3.34	1.93	370.88	203.91	1.79	1.67	51.59	19.86	1.84	1.69
4	1.69	3.82	219.11	134.72	3.03	2.53	28.37	12.33	3.34	2.72
8	0.88	7.34	144.69	100.76	4.58	3.38	16.51	7.72	5.75	4.35
16	0.50	12.86	111.27	85.20	5.96	4.00	10.99	5.98	8.63	5.61
32	0.41	15.72	95.19	69.80	6.97	4.88	10.14	6.03	9.36	5.56
GEMM										
seq	116.56	1.00	11943.84	6116.03	1.00	1.00	1606.94	499.66	1.00	1.00
2	56.84	2.05	6162.93	3320.87	1.94	1.84	786.53	246.54	2.04	2.03
4	34.26	3.40	4092.08	2378.91	2.92	2.57	473.73	148.22	3.39	3.37
8	17.14	6.80	2471.08	1614.09	4.83	3.79	236.93	74.10	6.78	6.74
16	8.61	13.54	1670.17	1239.89	7.15	4.93	119.88	38.12	13.41	13.11
32	5.56	20.97	1330.47	1052.57	8.98	5.81	77.29	24.49	20.79	20.40
TRIAD										
seq	5.43	1.00	544.94	273.55	1.00	1.00	85.12	33.56	1.00	1.00
2	2.94	1.85	313.14	166.28	1.74	1.65	49.97	22.06	1.70	1.52
4	1.50	3.63	185.50	110.67	2.94	2.47	29.80	15.58	2.86	2.15
8	0.80	6.80	120.49	80.54	4.52	3.40	19.48	11.49	4.37	2.92
16	0.49	11.03	95.21	69.64	5.72	3.93	15.05	10.13	5.65	3.31
32	0.48	11.23	96.20	66.22	5.66	4.13	14.63	9.79	5.82	3.43

On AMD (note that we could not get RAM energy as there no specific hardware counter for that), the volume of memory is stable but significantly fluctuates with GEMM although a constant bandwidth. For other applications, the bandwidth increases with the number of cores as the overall energy.

The main conclusion here is that optimizing for memory has a twofold benefit: reduction of both execution time and consumed energy.

V. GPU INVESTIGATION THROUGH PYTHON FRAMEWORKS

The goal of this section is twofold, we first show that EA2P is operational for measuring power consumption of specific applications on GPU and then we analyse the obtained measurements in order to provide some insights about the trade-off between speed and energy. Tables VIII and IX display our experimental results. In both cases, the whole program is launch on the CPU and only the main kernel is offloaded on the GPU. The timeframe of the measure (time and energy) is that of the execution on the GPU without data transfers. This means that the reported energy values for the GPU (resp. CPU) is related to its computation (resp. the moment the GPU operates). We generally see that, even if the CPU is not doing any computation during that time, it consumes a significant part of the overall energy, sometimes the major part like TRIAD/JAX, TRIAD/TORCH and STENCIL/JAX (see Table VIII, where the last two columns related to *power* are obtained by dividing the overall energy by the execution time). In addition, the considered framework has a significant impact on computing time on the GPU (SPMV is six times faster with JAX that with PyTorch for instance), and thus on energy consumption (ten times more with the same example). Power looks stable from the CPU side, which is normal as there no ongoing computation. However, from the GPU side, power fluctuates (55W with STENCIL to 392W with GEMM using JAX), a similar gap can be observed with TRIAD and GEMM using PyTorch. What we can say at this point is that, for a given application, the choice of the implementation is important depending on its structural nature.

A. Main experimental observations

GEMM *(Compute-Bound Kernel):* GEMM is the most power-hungry kernel across all frameworks (see the penultimate column of Table VIII), which aligns with the fact that it is heavily compute-bound. TensorFlow and JAX stand out, with TensorFlow leading in GFLOPS/W while JAX has good performance but with the highest power consumption.

TRIAD *(Memory-Bound Kernel):* CuPy and TensorFlow are the most power-hungry for this kernel, probably the price from their aggressive approach to maximize performance.

SPMV *(Balanced Kernel):* TensorFlow demonstrates the best performance (which indicates that it is well-optimized for sparse computation on GPU) while also JAX is the most power-efficient. PyTorch does not perform well neither on speed nor on power consumption.

Monte Carlo Pi Estimation *(Compute-Bound Kernel):* JAX (resp. CuPy) is the most (resp. less) efficient both in time and power. JAX seems particularly well-suited for compute-bound applications, where it efficiently converts power into computational output (best Gflops/W).

7-Point 3D Stencil *(Memory-Bound Kernel):* In the 3D stencil benchmark, JAX provides the best energy-efficiency (W), and it also achieves the highest performance per watt (Gflops/W). This indicates that JAX is well-optimized for handling large-scale memory-intensive tasks on GPU.

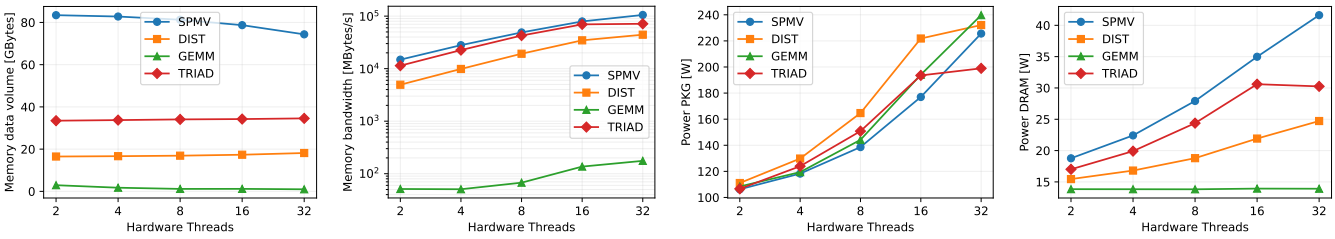
DIST *(Compute-Bound Kernel):* JAX achieves the best performance with similar power that the others (thus the best Gflops/W). JAX's strength in compute-bound scenarios looks consistent, making it a good choice for intensive calculation applications.

TABLE VIII
BENCHMARK RESULTS FROM THE FRAMEWORKS STANDPOINT

			Energy(J)		Power(W)	
Ben	Bench	Time (s)	GPU	CPU	GPU	CPU
JAX	triad	22.707	1019.163	1333.950	44.883	58.746
	gemm	2.868	1124.698	200.235	392.185	69.823
	dist	16.405	4199.758	1189.615	256.004	72.515
	stencil	20.468	1139.352	1238.476	55.665	60.508
	spmv	4.507	496.708	328.250	110.197	72.824
	monte_carlo	5.761	1409.745	428.575	244.721	74.397
TORCH	triad	23.729	913.214	1354.176	38.485	57.068
	gemm	22.346	7832.080	1621.745	350.498	72.576
	dist	6.460	1477.742	458.006	228.736	70.894
	stencil	9.561	2304.111	685.952	240.980	71.742
	spmv	26.672	5853.350	1928.214	219.460	72.295
	monte_carlo	6.983	1194.045	486.140	171.004	69.622
TFLOW	triad	3.496	828.763	245.760	237.071	70.300
	gemm	3.964	1426.650	298.625	359.942	75.343
	dist	6.292	1511.220	450.808	240.175	71.646
	stencil	12.418	2942.685	904.744	236.968	72.857
	spmv	3.416	632.714	246.714	185.196	72.213
	monte_carlo	7.890	2084.313	572.970	264.187	72.624
CUPY	triad	3.477	983.167	246.026	282.737	70.752
	gemm	23.337	8309.033	1685.315	356.049	72.217
	dist	6.191	1909.106	445.360	308.350	71.933
	stencil	8.585	2331.348	621.098	271.563	72.348
	spmv	23.529	3263.848	1712.680	138.718	72.791
	monte_carlo	5.725	1777.493	414.570	310.486	72.416

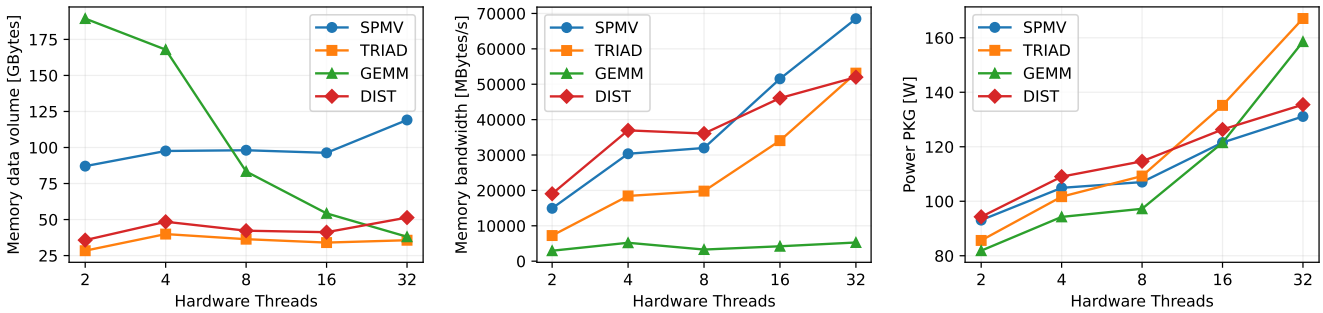
TABLE IX
BENCHMARK RESULTS FROM THE LIBRARIES STANDPOINT

Bench	lib	CPU (J)	GPU (J)	GPU Power (W)	Time (s)	Gflops/s	Gflops/W
Stencil	torch	685.95	2304.11	241.01	9.56	63.18	0.262
	tflow	904.74	2942.69	236.93	12.42	28.17	0.119
	jax	1238.48	1139.35	55.66	20.47	57.68	1.036
	cupy	621.10	2331.35	271.08	8.59	70.34	0.259
DIST	torch	458.01	1477.74	228.75	6.46	123.62	0.540
	tflow	450.81	1511.22	240.25	6.29	126.91	0.528
	jax	1189.62	4199.76	255.93	16.41	730.87	2.856
	cupy	445.36	1909.11	308.42	6.19	96.84	0.314
TRIAD	torch	1354.18	913.21	38.48	23.73	16.85	0.295
	tflow	245.76	828.76	236.79	3.50	114.12	0.482
	jax	1333.95	1019.16	44.87	22.71	17.61	0.392
	cupy	246.03	983.17	282.52	3.48	114.71	0.406
SPMV	torch	1928.21	5853.35	219.47	26.67	1.88	0.008
	tflow	246.71	632.71	185.00	3.42	35.32	0.190
	jax	328.25	496.71	110.13	4.51	3.20	0.029
	cupy	1712.68	3263.85	138.71	23.53	0.23	0.002
Monte carlo	torch	486.14	1194.05	171.06	6.98	214.44	1.253
	tflow	572.97	2084.31	264.17	7.89	253.18	0.958
	jax	428.58	1409.75	244.75	5.76	1732.50	7.079
	cupy	414.57	1777.49	310.21	5.73	261.56	0.843
GEMM	torch	1621.75	7832.08	350.43	22.35	17454.74	49.810
	tflow	298.63	1426.65	360.26	3.96	103368.22	286.923
	jax	200.24	1124.70	391.88	2.87	95814.29	244.498
	cupy	1685.32	8309.03	355.00	23.34	17595.68	49.426



(a) (b) (c) (d)

Fig. 1. Memory performance and power consumption with an INTEL multicore



(a) (b) (c)

Fig. 2. Memory performance and power consumption with an AMD multicore

B. *Global analysis of the considered frameworks*

- **JAX**: Demonstrates its strength not only on compute-bound tasks but also on complex memory-bound operations like 3D-stencil. Its ability to handle complex workloads with high efficiency makes it a strong client for tasks that require high memory bandwidth and good computing efficiency.
- **TensorFlow**: While TensorFlow show good performances in a broad range of tasks, it appears less efficient in a memory-bound scenario like 3D-stencil.
- **PyTorch**: Memory-bound tasks are still a little challenging. However, it remains a versatile framework, particularly for tasks with large data size support.
- **CuPy**: CuPy is globally powerful and can handle highly optimized workloads at the price of higher power consumption especially on memory-bound cases.

These insights are valuable for selecting the right framework based on the metric of interest (*time* or *energy*) and on the specific nature of the application (*compute-bound*, *memory-bound*, or a *mixed*).

VI. Summary, Conclusion and Future Work

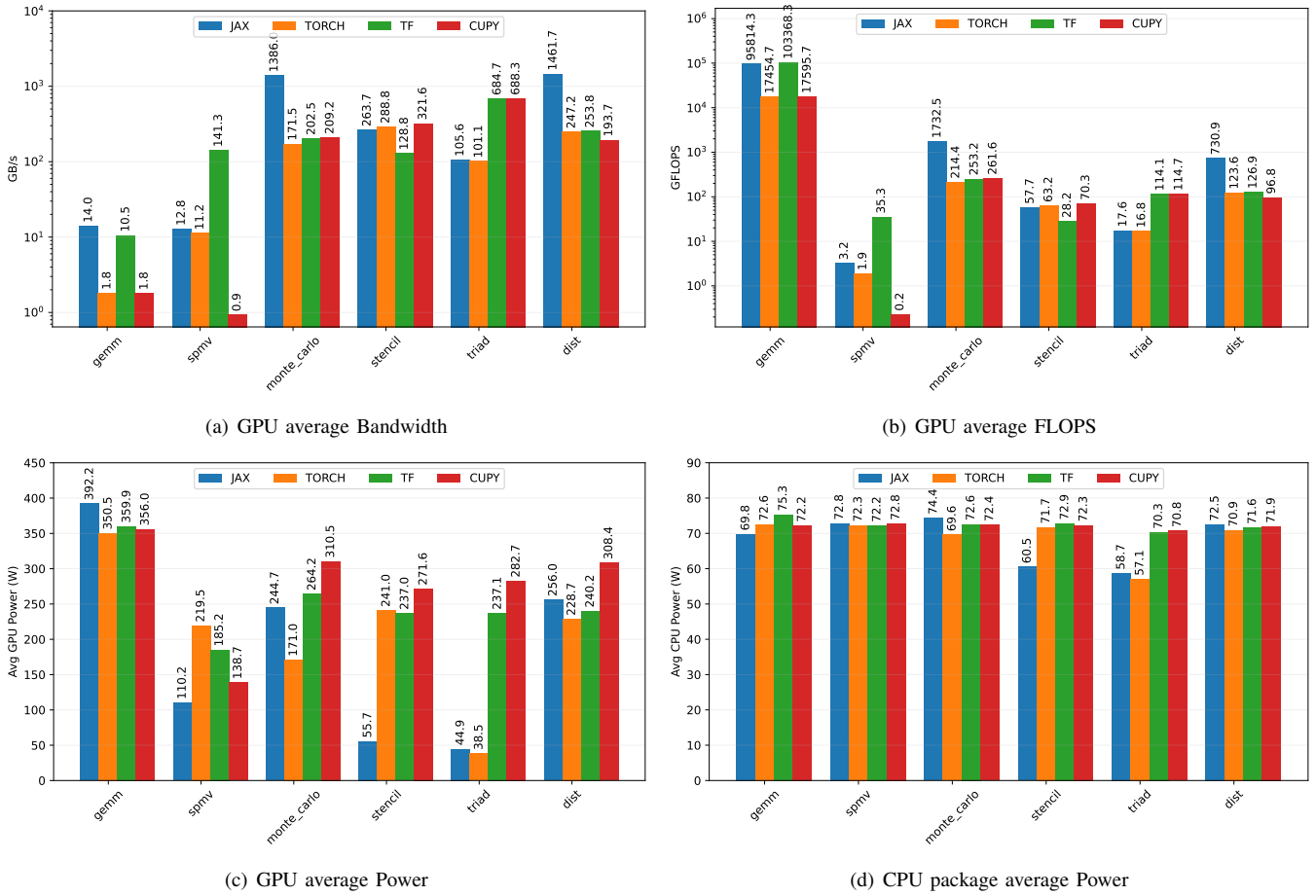
Our goal was to conduct an experimental investigation of the power consumption pattern considering standard both CPU (single-core and multi-core) and GPU so as to provide some insights about the correlation between speedup and energy reduction.

The correlation between time and energy seems to remain valid even in the multi-core context to some extents. The regression models shows that while multi-threading (potentially) improves the computing speed, it does not (always) scale linearly with power consumption, thus highlighting the needs for specific tuning to achieve an optimal balance between performance and energy efficiency.

Regarding power consumption related to memory accesses, higher bandwidth is likely to yield an increase of the power draw but can improve the overall energy efficiency, particularly with large data volume. Things are less predictable with memory-bound tasks, thus the need for careful memory management so as to avoid unnecessary power consumption overhead.

In summary, this research underscores the complex relationship between performance, power consumption, and energy efficiency across different architectures and workloads. Optimizing for both energy efficiency and performance requires a comprehension of workload characteristics, hardware-specific features, and the impact of parallelization and vectorization. These aspects are the basis for investigating energy-efficient computing with CPUs and GPUs.

For future work, we plan to design power management techniques by deeply investigating all hardware power-saving features for energy-aware programming and scheduling. From the programming side, we wish to consider our observations together with existing fine-grain profiling tools/API to design a methodology to seek energy-efficient implementation in the major computing platforms. A DSL approach [22] could be considered too, so as to seek an energy-aware code generation.



(a) GPU average Bandwidth

(b) GPU average FLOPS

(c) GPU average Power

(d) CPU package average Power

Fig. 3. GFLOPS and GB/s vs Power all benchmaks and frameworks

VII. Acknowledgements

This research was supported by The Transition Institute 1.5 driven by École des Mines de Paris - PSL.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.

References

[1] S. R. Sukumar, J. A. Balma, C. Xu, and S. Serebryakov, "Survival of the fittest amidst the cambrian explosion of processor architectures for artificial intelligence : Invited paper," in *2021 IEEE/ACM Programming Environments for Heterogeneous Computing (PEHC)*, 2021, pp. 34–43.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[3] R. Nana, C. Tadonki, P. Dokladal, and Y. Mesri, "Energy concerns with hpc systems and applications," *ArXiv*, vol. abs/2309.08615, 2023.

[4] International Energy Agency (IEA), "Electricity 2024," IEA, Paris, 2024, https://www.iea.org/reports/electricity-2024.

[5] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday, "The real climate and transformative impact of ict: A critique of estimates, trends, and regulations," *Patterns*, vol. 2, no. 9, 2021.

[6] N. J. Bates and M. K. Patterson, "Achieving the 20MW target: mobilizing the HPC community to accelerate energy efficient computing." *IEEE International Conference on High Performance Computing, Data, and Analytics*, pp. 37–45, 1 2012. [Online]. Available: https://doi.org/10.3233/978-1-61499-324-7-37

[7] S. Bhalachandra, B. Austin, and N. J. Wright, "Understanding power variation and its implications on performance optimization on the cori supercomputer," in *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2021, pp. 51–62.

[8] M. A. H. Monil, S. Lee, J. S. Vetter, and A. D. Malony, "Understanding the impact of memory access patterns in intel processors," in *2020 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, 2020, pp. 52–61.

[9] C. Tadonki, "High performance computing as a combination of machines and methods and programming," Ph.D. dissertation, Université Paris Sud-Paris XI, 2013.

[10] H. Inoue, "How simd width affects energy efficiency: A case study on sorting," in *2016 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX)*, 2016, pp. 1–3.

[11] A. Guermouche and A. Orgerie, "Thermal design power and vectorized instructions behavior," *Concurrency and Computation: Practice and Experience*, vol. 34, 03 2021.

[12] K. J. Brown, A. K. Sujeeth, H. J. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun, "A heterogeneous parallel framework for domain-specific languages," in *2011 International Conference on Parallel Architectures and Compilation Techniques*, 2011, pp. 89–100.

[13] T. Jakobs and G. Runger, "Examining energy efficiency of vectorization techniques using a gaussian elimination," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*, 2018, pp. 268–275.

[14] T. Aamodt, W. Fung, and T. Rogers, *General-Purpose Graphics Processor Architectures*, ser. Synthesis Lectures on Computer Architecture. Springer International Publishing, 2022.

[15] Nvidia, https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html.

[16] S. Georgiou, M. Kechagia, T. Sharma, F. Sarro, and Y. Zou, "Green ai: Do deep learning frameworks have different costs?" in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1082–1094.
[17] H. H. Holm, A. R. Brodtkorb, and M. L. Sætra, "Gpu computing with python: Performance, energy efficiency and usability," *Computation*, vol. 8, no. 1, 2020.
[18] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *2010 39th International Conference on Parallel Processing Workshops*, 2010, pp. 207–216.
[19] CRI ENSMP, https://github.com/HPC-CRI/EA2P.
[20] Wikichip, "Epyc 7513 - amd," 2021, accessed: 2024-05-07. [Online]. Available: https://en.wikichip.org/wiki/amd/epyc/7513
[21] CPU-world, "Intel xeon 8358 specifications," 2023, accessed: 2024-05-07. [Online]. Available: https://www.cpu-world.com/CPUs/Xeon/Intel-Xeon\%208358.html
[22] D. Barthou, G. Grosdidier, M. Kruse, O. Pene, and C. Tadonki, "Qiral: A high level language for lattice qcd code generation," *arXiv preprint arXiv:1208.4035*, 2012.