# A Flexible Operational Framework for Energy Profiling of Programs

Roblex Nana Tchakoute, Claude Tadonki, Petr Dokladal, Petr Dokladal, Youssef Mesri

# A Flexible Operational Framework for Energy Profiling of Programs

Roblex Nana Tchakoute*, Claude Tadonki*, Petr Dokladal† and Youssef Mesri‡
*Centre de Recherche en Informatique (CRI), Mines Paris - PSL, Fontainebleau, France**
*Centre de Morphologie Mathématique (CMM), Mines Paris - PSL, Fontainebleau, France†*
*Centre de Mise en Forme de Matériaux (CEMEF), Mines Paris - PSL, Sophia Antipolis, France‡*
Email : {roblex.nana_tchakoute, claude.tadonki, petr.dokladal, youssef.mesri}@minesparis.psl.eu

***Abstract*—Energy has become a serious concern in the HPC ecosystem for various reasons. The level of the issue is from crucial to critical depending on the context. Considering the variety of hardware devices and the granularity of the desired measurements, addressing the problem of energy profiling clearly requires adequate support tools for *measurements* and (energy aware) *monitoring*. There are several frameworks that aim at estimating the energy consumed by a given program on a given computing system, however they mainly suffer from some shortcomings related to *flexibility*, *portability*, *accuracy* and *programmability*. Thus, we propose a tool, so-named EA2P, that is reasonably flexible from both hardware and software standpoints, supporting the major parallel paradigms including GPU acceleration. The tool is publicly available at https://github.com/HPC-CRI/EA2P.**

***Index Terms*—Energy, power, profiling, code instrumentation**

## I. Introduction

Beside traditional complexity metrics like *time* and *space*, an important focus is nowadays on *energy*. This is due to the increasing concern about energy cost and related consequences, together with an intensive usage of computing systems for scientific research and real-life applications [1]. Vendors in the HPC industry are struggling to provide hardware and software solutions to meet this demand. Traditional Central Processing Units (CPUs) are no longer sufficient to efficiently handle cutting-edge AI applications and the vast data streams from diverse sensors. Leading technology companies like NVIDIA, AMD, Intel and ARM have been actively addressing these evolving requirements by pushing the boundaries of technology to provide efficient and adaptable processing solutions. However, the energy consumption of these HPC devices is becoming so important that it has to be seriously taken into account. This concern is also motivated by the correlation with the associated *carbon footprint* [2]. To illustrate the level of energy consumption of computing activities, we cite two examples: (1) large-scale data centers consume more than 100 megawatts ($\approx$ power consumption of 80,000 households) [3]; (2) the energy consumption of training the current world's largest open multilingual language Model (BLOOM) is 433,196 kWh $\approx$ 24.69 tons of CO2 emissions [4]. According to "Bird & Bird" [5], the high-tech sector is responsible for around 7% of global electricity consumption with a prediction of 13% by 2030. It is thus clear that optimizing the energy of computing activities is of a genuine importance, and the *need for efficient/accurate energy measurement tools* for this purpose is highly relevant.

Energy consumption can be measured with available tools like Perf [6], PAPI [7], Likwid [8], Intel PowerGadget [9], Intel PowerTop [10], Linaro Forge DDT & MAP [11], CrayPat [12], Score-P [13], to name few of them, or newer generation tools like perun [14], PyJoules [15], CodeCarbon [16], Eco2AI [17], Variorum [18], Carbon Tracker [19], Tracarbon [20], Experiment-impact-tracker(EIT) [21], Green Algorithms [22]. An experimental comparison of software-based power meters is provided by Jay et al. [23]. Most of existing tools have a weakness related to important characteristics like *flexibility, portability, accuracy, programmability and maintainability* that are necessary for energy profiling. Flexibility is crucial in a context of heterogeneous computing as the measurements have to be done for various kinds of device using the same tool. Indeed, a typical HPC system nowadays mixes multi-core CPUs, Graphics Processing Units (GPUs), and Field Programmable Gate Arrays (FPGAs). Programmability is important for the design of energy-aware programs directly or through code generation [24].

In this paper, we present "EA2P (Energy-Aware Application Profiler)", a flexible, accurate and multi-platform tool for energy profiling of Python Programs. EA2P is a Python package to measure the energy consumption of the major devices (RAM, CPU, GPU) when running Python applications (like most AI frameworks). Our tool is built on top of a set of APIs that includes RAPL interfaces, Linux Perf tools, Nvidia-SMI and ROCm-SMI. Our methodology allows to accurately estimate the energy consumption of the main memory (RAM) on platforms that do not have specific hardware counters for example. Our experimental evaluation illustrates the *flexibility* and *accuracy* of our tool for energy measurements.

The keys contribution of this paper are :

- Design and implementation of multi-devices energy profiler that works for the major cases.
- An analytical model for RAM energy measurement on platforms that do not integrate RAPL DRAM.

We provide an experimental validation of our tool and its model through a comparison with the results obtained with other existing tools, considering different use cases like ML training (Python) and benchmarking C/C++ (for non-Python cases through the binaries).

The remainder of this paper is organised as follows. Section II provides a detailed explanation about the motivation behind the design of our proposed tool, followed by the description of the basic technical background about energy/power measurement in Section III. The design and implementation of our tool is presented in Section IV, followed by the results of our experimental evaluation in Section V. Section VI concludes the paper and indicated some perspectives.

## II. Preliminaries and background

The major contributors to the energy consumption in a common computing system include the *GPU* (if any), *CPU*, and *memory*. Nevertheless, there are other components that also impact on energy like *fans*, *optical drives*, *motherboards*, and *hard drives*. Moreover, energy consumption can fluctuate due to external factors like the *ambient temperature*, the *overhead of the OS* and the *dynamic behaviour* of user-applications. Figure 1 displays an overview of the power ranges of the main components of a standard computer. A study by Appuswamy et al. [25] indicates that the power consumption by a DRAM becomes increasingly significant in overloaded systems, getting close to the CPU power consumption at 1.5 TB of memory for 2 processor chips, and 3 TB for 4 chips. At 6 TB, the DRAM significantly consumes more power than the idle CPUs (2.4× to 4.7×). Note that the values provided by Fig. 1 can be merged since several components might operate concomitantly.

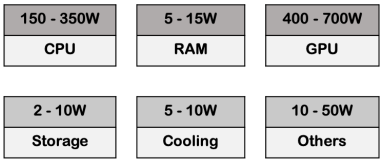


Fig. 1. Typical power ranges in a computing system.

Standard solutions for fine-grain energy profiling are based on built-in sensors and native support tools/APIs, despite being thereby limited to what provided by the manufacturers. Table I provides a compatibility overview of the aforementioned tools including ours (EA2P), *where CodeC = CodeCarbon, EIT = Experimental Impact Tracker, CTrac = Carbon tracker, TraC=Tracarbon, IntelPG = Intel PowerGadget, DDT-MAP = Linaro Forge DDT & MAP and our* ***"EA2P"*****.**

Considering the major tools listed in Table I (Perun, CodeCarbon, EIT, Eco2AI, TraCarbon, Likwid, and Variorum) and their respective limitations, we have been motivated to design a new framework that offers more flexibility.

**Perun** is a Python package designed to estimate the energy consumption of Python scripts on standard processors and clusters (it can handle MPI implementation). Perun can be used as a command-line tool or through directives provided within the code to be profiled. The user has to provide information likes *carbon intensity* and *electricity prices*. Additionally, Perun does not provide detailed measurements for systems with multiple CPUs or GPUs. Its RAM energy measurement is limited to Intel servers that expose RAPL energy domains.

**Eco2AI** is a Python library for CO2 emission tracking. It monitors energy consumption of CPU and GPU devices, estimating the equivalent carbon emissions based on regional emission coefficients. Eco2AI can handle any Python script. An important drawback of this tool is that its provides a rough estimation of the energy based on device utilization rates and nominal TDP values (it does not access hardware sensors).

**CodeCarbon, EIT**, and **TraCarbon** are also Python libraries designed for tracking energy consumption and estimating carbon emissions. However, these tools do not support all devices and lack detailed measurements related to RAPL power domains *(e.g., uncore, core, pkg, psys, RAM)* and are hard to adapt to new devices. A weak accuracy is likely to be observed on platforms without energy-related sensors.

**Likwid** was originally designed as a performance monitoring tool. The addition of the `Likwid-Powermeter` module allows to get energy values from hardware sensors. The tool offers a C/C++ API for instrumentation of specific sections of a given program. However, Likwid is not designed for Python code instrumentation and lacks comprehensive support for energy monitoring across multiple devices, particularly GPUs.

**Variorum** represents a significant evolution of the older `libmsr` library developed at LLNL. It has a flexible design and can be easily ported to various devices. However, it is not intended for fine-grain energy profiling of programs such as measuring the energy consumption of specific modules or code sections at runtime. Its API focuses on providing wrappers for vendor-specific power subroutines, making it more suitable for integration into higher-level energy management tools like GEOPM [26] and PowerAPI [27].

In addition, none of these tools can provide accurate energy measurements for systems that lack hardware sensors for main memory (RAM).

The framework we propose, named **EA2P**, addresses the aforementioned limitations by offering a flexible support for the major devices, featuring a simple and generic design that facilitates its adoption for other device types such as *ARM, POWER CPUs, FPGAs,* and *ASICs*. EA2P's modular Python structure allows for an easy extension by adding a few lines of code to read the values of the hardware sensors. In addition, the EA2P has a RAM energy model for an accurate estimation of the power consumption of the RAM.

## III. Technical basis of energy measurement

The primary challenge in utilizing hardware energy meters lies in the difficulty of obtaining critical technical information from both native documentation and scientific literature. Frequently, programmers must resort to reverse engineering in order to figure out the specific behavior of energy-related components. Below, we enumerate the main interfaces for accessing energy-related counters on the major computing devices.

- **Intel RAPL interface**
  The Running Average Power Limit (RAPL) is a monitoring feature that calculates and reports the combined energy consumption of the CPU, RAM, and some other

TABLE I
COMPATIBILITY OVERVIEW OF ENERGY MEASUREMENT FRAMEWORKS.

| Support | Perun | CodeC | EIT | CTrac | Eco2AI | TraC | PyJoules | Perf | LIKWID | PAPI | IntelPG | Powertop | Score-P | Variorum | CrayPat | DDT-MAP | EA2P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **GPU support** | | | | | | | | | | | | | | | | | |
| Nvidia GPU | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | ✓ | ✓ | ✓ | ✓ |
| AMD GPU | ✓ | | | | | | | | | | | | | ✓ | | ✓ | ✓ |
| Intel GPU | | | | | | | | | | | | | | ✓ | | | |
| **CPU and RAM supports** | | | | | | | | | | | | | | | | | |
| Intel CPU | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| AMD CPU | | | | | ✓ | | | ✓ | ✓ | | | ✓ | ✓ | ✓ | | | ✓ |
| RAM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | | | ✓ | | | ✓ |
| **OS support** | | | | | | | | | | | | | | | | | |
| Linux | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Windows | | ✓ | | | | | | | | | ✓ | | | | | | ✓ |
| Mac OS | | ✓ | ✓ | | | ✓ | | | | | ✓ | | | | | | |
| **Other important characteristics** | | | | | | | | | | | | | | | | | |
| Documentation | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ |
| Configurable | ✓ | ✓ | ✓ | | | | | | | | | ✓ | | | | | ✓ |
| Code API | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | | | | ✓ |
| Open Source | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ |
| Perf oriented | | | | | | | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | ✓ | |
| Energy oriented | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | ✓ | ✓ | | ✓ | | | ✓ |
| Multi-Nodes | ✓ | | | | | | | | ✓ | | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Device details | | | | | | | | | | | | | | | | | ✓ |

components in a computer system. RAPL gained support from integrated voltage regulators, which enhanced its power modeling capabilities [28]. Extensive research has been conducted to validate the effectiveness of RAPL for software-based monitoring of computer systems [29].

- **AMD RAPL interface**
  The 5.8 version of the Linux kernel brought significant enhancements for AMD processors, like the AMD Energy Driver for detailed energy reporting and the integration of RAPL support for AMD Zen/Zen2 CPUs [30]. Recall that the RAPL framework was originally developed by Intel to enable fine-grained control and monitoring of power with their processors. AMD's integration of RAPL support into Linux kernel was made to provide similar capabilities for AMD CPUs.
- **Nvidia-SMI**
  Nvidia-SMI (Nvidia System Management Interface) [31] offers a simpler and more accessible solution for power measurements, but with limited information. It reports the current active power of the entire GPU board.
- **ROCm-SMI**
  ROCm-SMI (Radeon Open Compute System Management Interface) [32] is a command-line interface developed by AMD as part of the ROCm software stack. It provides a set of tools for managing and monitoring AMD GPUs kernels that are compatible with the ROCm platform. ROCm-SMI has been designed to help for the optimization of performance and power consumption of AMD GPUs.

In most cases, RAPL, Nvidia-SMI and ROCm-SMI provide energy/power measurements for the entire board or socket. To estimate the energy consumed by a specific program, an approach commonly used is to multiply the RAPL value by the percentage of CPU and memory it has used. However, it is important to note that the energy consumed by a set of programs is not strictly additive. Several factors contribute to this complexity:

1) **Fixed Energy Waste:** There is a baseline energy consumption associated with idle states and other background processes. This fixed energy waste can vary and is not directly proportional to the load of the running programs.
2) **Non-linearity:** The relationship between computer resource usage (e.g., CPU, memory) and energy consumption is not linear. Energy consumption may exhibit time delays or non-proportional increases as resource utilization varies.

Considering the aforementioned factors, it is recommended to benchmark a program when it is running alone on the machine in order to get a more accurate pattern of its energy consumption. This will provide a baseline for energy consumption that can be used as a reference when comparing it to multi-program or multi-task scenarios. Understanding these nuances is crucial for accurate energy profiling and optimization of software and hardware systems. We now present and describe our energy profiling tool so-called EA2P (Energy Aware Application Profiler), which we designed considering the major weakness points we have previously mentioned w.r.t. existing measurement tools.

## IV. PRESENTATION AND DESCRIPTION OF OUR TOOL

We now describe the motivation, the design, the implementation, and the global workflow of our tool.

### A. Key characteristics of a profiling tool

**- Programmability**: This refers to the possibility to call specific features of the tool within a high-level program through an API. This possibility allows to annotate the user code to be profiled to as to be able to measure the energy consumed by specific sections or modules of the application. Such feature also allows for a fine-grained profiling.

**- Flexibility**: Flexibility of use means being able to adapt the tool to particular needs like measuring the energy of specific hardware units, handling special configurations such as sampling frequency, target hardware selection, and porting to other architectures. Flexibility is a suitable characteristic for heterogeneous systems since they combine devices from different natures.

**- Standalone/Freestanding**: Energy profiling is a complex task that requires efficient and user-friendly tools that are easy to install and handle. If a given framework requires third-party tools that are complex to handle, this could be the reason for giving up with that choice. Hardware counters typically require appropriate OS user rights for example. For instance, only root user has the ability to enable*Perf Events*, *Intel PowerCap* and *DMI decode* back-ends.

**- Portability across devices**: Our primary technical motivation was to maximize compatibility across various architectures or devices. Hardware counters compatibility between different architectures is not always granted. To address the issue as well as the complexity of analyzing different systems, we chose to create and connect with tools that rely on standard hardware counters, which are more likely to be available on newer devices. As result, we have prioritized the consideration of popular tools with user-friendly interfaces.

**- Accuracy**: To be able to draw relevant conclusions from the measurements provided by a tool, it is necessary to perform two levels of validation. The first one is to ensure that the returned values correspond to what is being measured, and the second one is to validate the correctness the values provided by the hardware counters. Our approach is based on the aggregation of the measurements obtained by querying the hardware interface of all devices, assuming that they operate properly.

### *B. Technical overview our tool*

Our tool is built on top of some energy measurement APIs for various devices, and its global underlying diagram is displayed in Figure 2.

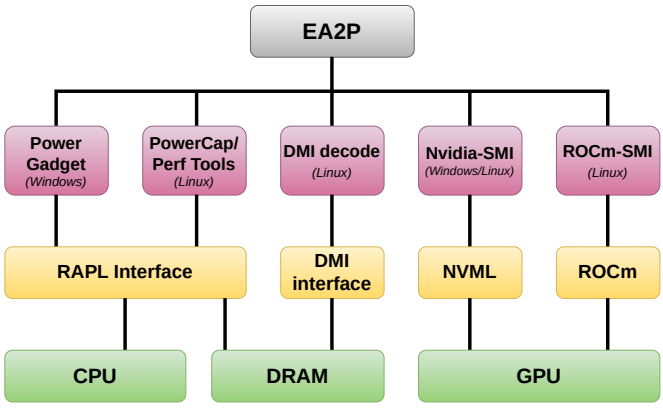


Fig. 2. Main diagram of our framework

We have considered the following aspects to enhance the utility and usability of our tool.

- *Modularity and Flexibility*: The modular nature of Python enables an easy integration of different (third-party) tools and APIs, thereby our tool should adapt to various environments and requirements seamlessly.
- *Ease of Use*: The readability of Python together with its extensive set of libraries make our tool more accessible to ordinary users by easing its understanding, modification, and extension.
- *Registers Access*: Using intermediate-level tools to retrieve the values of power-related registers allows for direct access to hardware-level information, which thereby improves the accuracy of energy measurements.
- *Dynamic Tools Selection*: Automatic identification of required third-party tools based on the system configuration makes the installation process more simpler.
- *Versatility*: Offering both standalone and API-based functionalities makes our tool easily adaptable to various use cases. Users can choose the mode that best suits their needs (either for a direct use or for an integration into an existing application).
- *Programmability*: Our tool is provided with an API for direct calls of specific features, thus allowing developers to seamlessly integrate energy measurement functionalities into their applications or workflows.
- *Compatibility and Portability*: The compatibility across different systems and platforms for our tool comes from the interpreted nature and the popularity of Python.
- *Documentation*: A detailed documentation explaining the functionalities of the tool and how to integrate them into a workflow is important for the developers from the hand-on standpoint.

**About Windows OS:**
For Windows support, we have two scenarios: *direct support* and *WSL2 support*.

For direct support, our tool currently relies on Intel Power Gadget for CPU energy measurement, while GPU measurement remains dependent on Nvidia-SMI. We have not yet tested AMD devices, and direct RAM support has not been implemented yet. Windows support is quite limited due to the lack of a command-line interface similar to Linux systems, thus Intel Power Gadget needs to be pre-installed on Windows.

For WSL2 support, EA2P runs like on a real Linux system. However, the results are not guaranteed since WSL2 might experience some limitations (functionality and permissions) to access hardware sensors on Windows. We have not yet tested this portability, but it is a target for future versions of EA2P for Windows users.

### *C. Implementation of our tool*

We describe each of the main feature of our tool w.r.t. energy measurement depending on the device or the hardware unit considered.

*1) RAM energy measurement:* On Intel server architectures, energy profiling of the main memory (RAM) system aligns with the PowerCap interface, similar to that of the CPU, as the RAM is covered by RAPL. However, this approach is not applicable to *Intel client* and *AMD CPU* systems for which an alternative method is thus needed to estimate the energy consumed by the RAM.

A typical solution uses analytical models considering the workload together with TDP or nominal power values. However, there is a genuine difficulty that comes from the variations among different generations of DDR RAM, particularly in terms of the Power Management Circuit (PMC). For instance, DDR5 technology stands out as a significant advance over DDR4, where a Power Management Integrated Circuit (PMIC) is directly integrated into the dual in-line memory module (DIMM), thereby resulting in a noticeable 30% increase in power efficiency [33]. Consequently, any analytical model must consider this evolution with DDR technology.

Another crucial aspect that influences the power consumption of the RAM is the amount of memory space and the total number of memory sticks on the DIMM slots. According to Micron Technology [34], a general guideline suggests allocating approximately 3 watts of power for every 8GB of DDR3 or DDR4 memory. Interestingly, the amount of RAM has a marginal impact on the overall power consumption. For instance, a 4GB DDR3 RAM stick consumes a comparable amount of power than a 8GB DDR3 RAM stick, assuming they have the same clock speed [35].

We have considered a discretized approach together with the classical correlation between *power* and *time* to get an estimation of the global *energy* as expressed by formula (2).

For Intel Client architectures and AMD CPU systems, we use *"dmidecode"* to retrieve necessary information about the RAM (type, slots, size) that are used to get the *Power_base* value considered in formula (1) for the average power of the whole RAM.

$$Power\_RAM = Power\_base \times nb\_slots \times mem\_use \quad (1)$$

$$Energy\_RAM = (\sum_{i=1}^{N} Power\_RAM) \times interval, \quad (2)$$

where:

- **Power_base** is the TDP like value (i.e. the maximum possible power in this case) associated to each memory type and capacity. A list of common memory types and memory capacity together with their corresponding TDP is provided.
- **nb_slots** is the total number of memory slots within the entire motherboard. We consider the fact that the power consumption of the RAM depends on its number of slots. Thus a system with 4 slots in use is expected to consume 4 times more power than the same system with only 1 slot in use [36].
- **mem_use** is the percentage (i.e. footprint) of the total system memory that is used by the profiled code. The value is recorded at a the granularity of a sampling interval, so we end up with as many values as the number of intervals (i.e. $N$). We process this measurement in parallel in order to reduce the time overhead.
- **interval** is the sampling frequency (in seconds) of the experiment.

**Limitations:** The power consumption of the memory seems to grow non-linearly w.r.t. to its size, transitioning from sublinear at low capacity to superlinear at high capacity. For instance, while a 16× increase in capacity from 8 GB to 128 GB results in a modest 3× rise in power consumption, the subsequent 4× increase to 1 TB leads to a significant 10× increase [25]. This is due to the necessity of using high-density DRAM DIMMs to achieve high capacities, coupled with the considerably higher power consumption of these high-density DIMMs compared to their low-density counterparts. These factors are not currently considered in our work at this time.

According to Crucial [37], with an empirical rule of 3W for every 8GB of DDR3/DDR4 RAM, and Skhynix [38] that states "Our 256GB DDR4 RDIMM consumes around 8.5W of power and LRDIMM around 10.5W, both the most competitive levels in the High Density Module market". We fixed Power_base at 10W for 256GB stick, 8W for 128GB, 6W for 64GB, 5W for 32GB and 4W for 16GB. Additionally, for less than 5% of real usage, we consider 30% of the theoretical value as we have the baseline power consumption for Idle state which is not-null. from 5% to 15% we consider 60% of the theoretical value as we enter in active state and so memory allocation on multiple DIMM memory. From 15% to 25%, we define 70%. For 25 to 50% we defined 75% and for more than 50% we set 80%. Our experimental validation (see fig 6) show that it work well on a server with 512 GB as it gave values close to those of Intel RAPL DRAM domains.

*2) CPU energy measurement:* For Intel systems, we use PowerCap interface. We periodically (at the *sampling frequency*) examine all energy log files, principally the file so-called *energy_uj*, within the *"/sys/class/powercap/intel-rapl"* directory and sum up all so collected results at the end of profiling. We take into account the impact of *max-energy-value* of the PowerCap interface, which is an upper bound of the power value that can be recorded.

For other CPUs, PowerCap interface does not apply. AMD RAPL does not implement it, so it's more difficult to get accurate measurements. The Linux *"perf tool"* provides a measurement for the "CPU package", which is equivalent to the *package* domain of Intel RAPL. The problem is that this tool is more suitable for profiling an entire application, not specific sections. Thus we execute (in parallel) the command *"perf stat -e power/energy-pkg"* and kill the corresponding processes after getting out of the instrumented code section. We store the values thus collected during the profiling in a log file and ultimately process its content to get the global output profile report. As mentioned for the RAM energy measurement in the previous section, all the snooping processes are performed in parallel in order to reduce the time overheads. The total energy of CPU domain using Intel RAPL power domain

is calculated using the update described by the algorithm displayed in Figure (3), where $E_i$ is the value provided by the RAPL from request $i^{th}$. In fact, each time we read the current value of the energy consumed so far, we update the global energy only if that value it is not lower than the previous one, which occurs when there has been a reset of the counter meanwhile.

$$
\begin{aligned}
&E_{CPU_{dom}} = 0 \\
&e_{current} = 0 \\
&\textbf{if } (E_i > e_{current}) \\
&\quad E_{CPU_{dom}} = E_{CPU_{dom}} + (E_i - e_{current}) \\
&e_{current} = E_i
\end{aligned}
$$

Fig. 3. Update of the overall CPU energy

For AMD processors, using *"perf stat"* results in only two measures per run (one at the beginning and one at the end). The procedure described in Figure (3) is for Intel CPUs as we can query multiple values from the powerCap RAPL interfaces at the same sampling frequency. It is important to highlight the fact that multiple queries are essential because there is an automatic reset of powerCap registers when the *energy_uj* files reach a maximum value predefined by the system. This behavior must has been considered in the implementation of our tool as previously explained about the measurement procedure.

*3) GPU energy measurement:* Regarding energy measurement of a GPU, our tool use the native vendor interfaces as indicated in section III (i.e., Nvidia-SMI and ROCm-SMI) through command line calls to get the power at a given frequency. Similar to the case of standard CPUs as previously explained, we collect the measurements (at the *sampling frequency*) in a log file that is processed at the end to get the expected profile report. The overall energy is calculated using formula (3).

$$Energy_{GPU} = \sum_{i=1}^{N} P_i \times \Delta T_i, \tag{3}$$

where $P_i$ (resp. $\Delta T_i$) is the power value of the GPU (resp. the $i^{th}$ time interval during which we assume a constant power $P_i$) at measurement step $i^{th}$.

*4) MPI extension for multi-node energy measurement:* Our multi-node energy measurement feature is based on the *MPI (Message Passing Interface)* implementation, specifically through the *"mpi4py"* library [39]. By leveraging *mpi4py* features, our tool can measure and report the overall energy consumption of a distributed memory parallel machine.

Each node runs its own instance of the energy measurement tool, capturing detailed energy metrics (mpi4py implementation ensures that these measurements are synchronized across nodes). Ultimately, energy data are collected from all nodes and aggregated to get the overall energy consumed by the cluster.

*5) Docker containerization for portability and ease of use:* To further enhance the usability and portability of our tool, we consider containerization using Docker. Docker is a popular platform for developing, shipping, and running applications in a consistent environment across different systems. By containerizing the tool, we ensure that it can be easily deployed on various systems without the need for manual installation of dependencies or system-specific configurations. This is particularly advantageous for users who wish to quickly set up and run the tool on heterogeneous environments.

Note that the Docker container does not includes all necessary dependencies for system tools such as *powerCap, dmidecode, nvidia-smi, and rocm-smi*. Therefore, the container to be in a privileged mode in order to be able to access hardware counters related to energy. On an Intel system for example, the container should be launch as described below:

```
docker run --privileged -it --rm \
  --device /dev/cpu/0/msr:/dev/cpu/0/msr \
  -v /sys/class/powercap:/sys/class/powercap\
  -v /proc:/proc \
  -v /dev/mem:/dev/mem \
  --gpus all \
  ea2p-tool
```

By providing a Dockerized version of our tool, we make things easier for users who are not familiar with OS-level considerations or who not want to configure their environment manually. The containerization approach also supports reproducibility, as the same container can be used across different systems. This makes the tool not only more accessible but also more reliable for conducting energy efficiency studies in diverse computing environments.

### D. Functional workflow of EA2P

Figure 4 provides an overview of the step-by-step functional workflow followed by our framework. It is a structured approach for setting up, running, and concluding an energy measurement experiment using EA2P. The use of threads is for a parallel processing for data collection, thus ensuring minimal time overhead for the measurement itself.

The formatted output can be easily analyzed and considered as a valuable record of the experiment. We now describe each of the main hand-on steps.

- **Step 1: Environment Setup**
  *1.1. Installation Procedure:* Our energy measurement tool and the required third-party libraries can be installed using a package manager or a script. The EA2P package is available as open source on PyPi[1] website. It is required to have installed all necessary drivers, as well as to have root privileges to be able to read sensors outputs.

[1]https://pypi.org/project/EA2P/

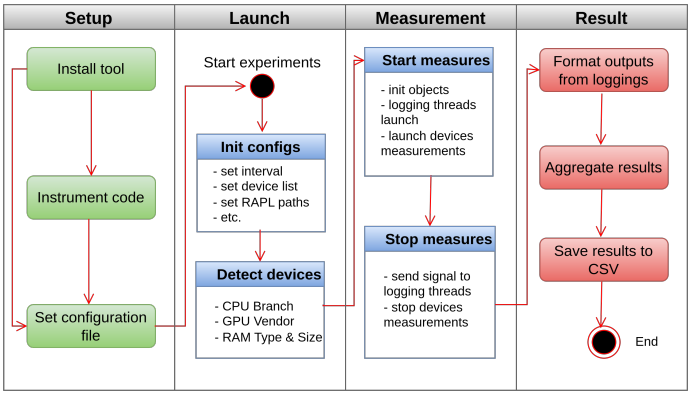


Fig. 4. EA2P general workflow.

*1.2. Code Instrumentation:* This is done by either *instrumenting the code* with the corresponding annotations wherever needed or *integrating energy measurement routines* into the code in order to capture energy related data as indicated in the documentation[2].

- **Step 2: Settings and Initialization**
  *2.1. Basic Parameterization:* The tool reads the configuration files for the main parameters like *sampling frequency, device details*, and *location of the output file*.
  *2.2. Automatic Detection of Devices:* We have implemented a mechanism that automatically detects and lists the main active devices for the profiling. In fact, we use appropriate APIs or libraries to identify and connect to the devices specified in the configuration files.
  *2.3. Experiment Initialization:* The tool sets up the main variables related to the experiment and then connects to the devices that have been detected in the previous step.
- **Step 3: Launching and Running the Profiling**
  *3.1. Parallel Energy Measurement:* We consider a multi-thread implementation (using Python threading library) to get and process energy values of the devices, so the measurements are done in parallel and independently. The measurement threads run in the background and collect energy data at a constant sampling frequency.
  *3.2. Instrumented Code Execution:* The main instrumented code region to be profiled is executed, with the energy measurement threads running concomitantly on the background as previously explained.
- **Step 4: Epilogue of the Experiment**
  *4.1. Termination:* We terminate the threads responsible for recording the energy measures once the main instrumented code region has reached its end. For Python MPI programs, data from each node (rank) are sent to master node which make final aggregation.
  *4.2. Output:* We process and format the collected energy data so a to make easier any further analysis and interpretation. Data are aggregated and organized based on the device or time intervals and the formatted output is written into to specified output files (CSV or text files). Metadata that provide experiment details are added. For distributed systems, the output contains per node data and a last column with total energies and times values.

[2]https://hpc-cri.github.io/EA2P/

## V. Experimental Evaluations

### A. Our test-bed

Our test-bed is based on the devices listed in table II, with CPUs and GPUs from Intel and AMD.

TABLE II
Platform Characteristics

| name | Intel-1 | AMD-1 | AMD-2 | Intel-2 |
|---|---|---|---|---|
| CPU name | i9 12950HX | EPYC 7452 | EPYC 7513 | E5-2698v4 |
| GPU name | RTX 3080Ti | A100 SXM4 | A100 SXM4 | Tesla V100 |
| CPU TDP | 55W | 155W (x2) | 200W(x1) | 135W (x2) |
| GPU TDP | 150W | 400W (x2) | 400W (x4) | 300W (x8) |
| CPU threads | 24 | 64 (x2) | 64 (x1) | 40 (x2) |
| GPU VRAM | 16GB | 40GB (x2) | 40GB (x4) | 32GB (x8) |
| CPU RAM | 32 GB | 128 GB | 512 GB | 512 GB |
| Multi-SoC | No | Yes | No | Yes |

Our validation process focuses on the following aspects:

- **Tool Accuracy Assessment:** Validate the accuracy of the measurements across different devices and hardware components (CPU, RAM and GPU).
- **Energy Profiling Consistency:** Ensure the consistency of profiling results across multiple different platforms (AMD, Intel, and Nvidia).
- **Workload Characterization:** Profile various kinds of workload ({CPU, GPU, CPU/GPU}-intensive) to evaluate the robustness of the tool.
- **Cross-platform Compatibility:** Check the portability of the tool on different platforms (AMD/Intel CPUs and AMD/Nvidia GPUs) for versatility.
- **Profiling Overhead Analysis:** Measure the time cost of the profiling.

The primary objective of our experiment is to measure and compare the energy consumption during the fine-tuning of the VGG16 [40] deep learning model on two different hardware configurations: CPU-only and GPU-accelerated. We use two datasets for this purpose: CIFAR-10 [41] and Stanford Dogs [42], [43]. Additionally, we use the “sleep“ function from Python’s “time“ package to validate idle energy consumption. To analyze NUMA and thread scalability for energy consumption (this aspect is not really in the scope of this paper), we perform parallel matrix multiplication using OpenMP and measure the whole code. The matrix size is set to 8000x8000 in single precision. This use case demonstrates the use of the EA2P command-line API to measure the energy consumption of various executable files. Consequently, C/C++, Fortran, and other codebases can be evaluated using this approach, even our API does not handle explicit source-code instrumentation for high-level languages other than Python. This functionality is particularly useful for energy optimization.

### B. Experimental results

We validate our tool beside established tools like Linux perf tools for CPU and RAM energy and CodeCarbon for GPU energy (see tables IV, V, VI, and VII). Understanding and mitigating the overhead, especially when it is considerably high, is key to obtaining accurate and consistent measurements.

We globally see that our tool returns nearly similar measurements than the baseline with nearly equal overheads. Each of the experiments was repeated 5 times to assess the consistency through the standard deviation which turned to be tiny. For simplicity, we only report the average values of the 5 runs and especially for table III, we report standard deviation to illustrate the variability which is due to the operating system and the machine state.

We describe the items of Table V :

- **cores:** is Power Plane 0 (PP0), associated with the CPU cores. It tracks the energy consumption of all cores.
- **uncore**: is Power Plane 1 (PP1), it measures the energy consumption of the integrated GPU (not to be confused with an external Nvidia or AMD GPU) on Intel client architectures only. Remember that powerCap Linux interface name this package as *"uncore"* while in Linux perf tools it correspond to *"energy-gpu"* event.
- **DRAM (RAM):** This domain specifically measures the energy consumption of the RAM.
- **Package (Pkg):** This domain tracks the overall energy consumption of the System on Chip (SoC). It includes the consumption of all cores, integrated graphics and also the uncore components (QuickPath Interconnect (QPI) controllers, L3 cache, snoop agent pipeline, on-die memory controller, on-die PCI Express Root Complex, Thunderbolt controller).
- **Psys:** It provides an aggregate view of power consumption across various components: package domain, System Agent, Platform Controller Hub(PCH), eDRAM and a few more domains on a single socket SoC. It is useful especially when the main source of the power consumption is neither the CPU nor the GPU, but the whole system instead.

*1) EA2P overhead analysis:* Table III provides an overview of the overhead associated with EA2P and perf tools when profiling the same workload (executed five times). This experiment aims to validate the stability of our tool considering the OS activity and its impact on the accuracy of the measurements. We reported the mean values for energy and runtime to test the sleep state or idle power on the CPU-intensive VGG16 CIFAR10 fine-tuning over 10 epochs. The deviation from these five runs was less than 3% for RAM and CPU during the CPU-intensive task. For idle system energy tests, this overhead is even more negligible, less than 1% for both CPU and RAM. These behaviors are highly correlated with runtime deviations and can be attributed to OS task management and thermal throttling, as current voltage and frequency may vary over time. The difference between EA2P and perf results is associated to the fact that EA2P focuses on instrumented code regions, while perf measures the energy of the entire application. Consequently, the results of perf are typically higher than those of EA2P.

TABLE III
CONSISTENCY OVER SYSTEM OVERHEAD ON INTEL-2

| Application | tool | Pkg (std) | RAM (std) | time(s) (std) |
|---|---|---|---|---|
| sleep | perf | 2.254(0.004) | 1.290(0.002) | 183.508(0.02) |
| | EA2P | 2.181(0.02) | 1.270(0.001) | 180.272(0.002) |
| | **gap (perf-EA2P)** | **0.073** | **0.02** | **3.236** |
| CIFAR-CPU | perf | 28.592(0.32) | 4.899(0.10) | 445.236(5.36) |
| | EA2P | 28.252(0.40) | 4.825(0.12) | 438.33(6.81) |
| | **gap (perf-EA2P)** | **0.34** | **0.074** | **6.906** |

TABLE IV
CPU AND DRAM VALIDATION

| | | Intel-2 | | | AMD-1 | | |
|---|---|---|---|---|---|---|---|
| | | Energy(Wh) | | time(s) | Energy(Wh) | | time(s) |
| Application | Tool | Pkg | RAM | | Pkg | RAM | |
| sleep | perf | 2.254 | 1.290 | 183.508 | 4.781 | / | 185.105 |
| | EA2P | 2.181 | 1.270 | 180.272 | 4.650 | 4.853 | 180.503 |
| CIFAR-CPU | perf | 28.592 | 4.899 | 445.236 | 45.580 | / | 574.215 |
| | EA2P | 28.252 | 4.825 | 438.333 | 45.293 | 14.241 | 557.021 |
| CIFAR-GPU | perf | 1.627 | 0.514 | 68.40 | 1.610 | / | 45.105 |
| | EA2P | 1.229 | 0.371 | 52.50 | 1.225 | 0.968 | 33.921 |

*2) Validation of EA2P through comparison to SOTA tools:* EA2P measurements seems to be lower than those of perf. But this behavior is associated to the fact that EA2P profiles only a specific section of the code while perf profile the entire application. This can be viewed as a clue regarding consistency.

TABLE V
CPU AND GPU VALIDATION ON INTEL-1 (NVIDIA RTX 3080TI)

| Application | tool | CPU (Wh) | GPU (Wh) | time(s) |
|---|---|---|---|---|
| sleep | CodeCarbon | 0.305 | 0.987 | 181.931 |
| | EA2P | 0.204 | 0.824 | 180.706 |
| CIFAR-GPU | CodeCarbon | 0.229 | 2.077 | 67.993 |
| | EA2P | 0.230 | 2.047 | 67.757 |

EA2P values seem to be lower than those from CodeCarbon, especially for CPU energy, with a non negligible gap. This behavior can be associated to the fact that CodeCarbon chooses 80% TDP of the CPU when it cannot detect the system path of RAPL, which thus leads to less accurate result. We checked this behavior on an Intel core i9 12950 HX (see Table V).

TABLE VI
CPU AND DRAM VALIDATION ON INTEL-1 (CORE I9 12950HX)

| | | Energy(Wh) | | | | | time(s) |
|---|---|---|---|---|---|---|---|
| Application | Tool | cores | uncore | pkg | psys | RAM | |
| Sleep | perf | 0.008 | 0.000 | 0.149 | 0.520 | / | 180.029 |
| | EA2P | 0.008 | 0.000 | 0.149 | 0.520 | 0.031 | 180.192 |
| CIFAR-GPU | perf | 0.089 | 0.001 | 0.274 | 2.78 | / | 72.626 |
| | EA2P | 0.056 | 0.001 | 0.229 | 2.672 | 0.014 | 66.903 |
| CIFAR-CPU | perf | 3.715 | 0.007 | 5.949 | 12.001 | / | 1476.905 |
| | EA2P | 3.696 | 0.007 | 5.952 | 13.488 | 0.295 | 1478.121 |

*3) Multi-GPU system analysis:* The discrepancy w.r.t. energy consumption and execution time between CPU-based and GPU-based computations highlights the relative time-energy positioning between the two processing units. Table VII shows our case with fine-tuning VGG16 on the Stanford dog dataset. We got nearly 77 Wh for more than 9 minutes as execution time on Intel Xeon. The GPU was disabled with CUDA_VISIBLE_DEVICE flag for the TensorFlow case so as to operate only with the CPU. The same code took around 10 Wh of energy for less than a minute on the same machine with 8 Nvidia V100 GPUs. We can also notice that only one GPU was effectively used for the computation since by default the TensorFlow engine did not distribute across the GPUs. We can also see it with the highest energy value on GPU 0 when each of the other GPUs consumes similar low values.

The highly parallel execution mode of the GPUs makes them significantly faster compared to CPUs, resulting in significantly lower energy consumption for the same task. This is more noticeable when running complex algorithms like deep learning models.

*4) Self-validation of EA2P on shared-memory parallelism:* Analyzing the energy consumption beside time performance considering a multithread execution provides valuable optimization insights. We did the measurement with our tool also for consistency purpose and the results are reported in figure 5 and 6, where we can see the correlation between the two metrics. This kind of experiment can help to seek an optimal balance between energy efficiency and time performance. In our example, we can observe that the best time-energy trade-off is obtained with 64 or 128 threads.

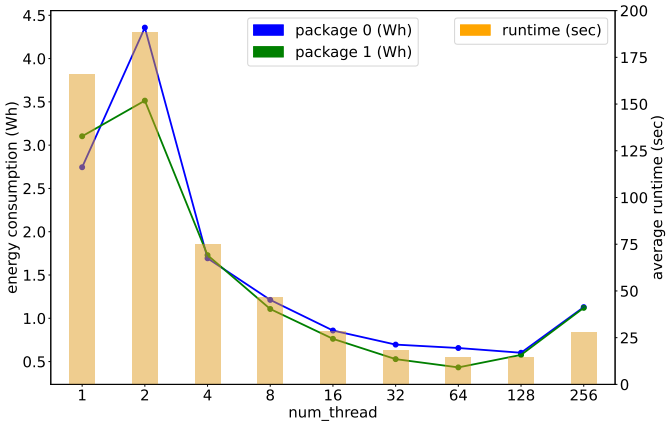


Fig. 5. Time vs energy reports with a multi-threaded case on **AMD-1**.

*5) Impact of the sampling frequency:* The *sampling frequency* is important in capturing accurate energy values and their correlation with running time as shown in Figure 7. Here are some considerations related to our experiment:

- **Correlation between Energy and Execution Time:** Smaller sampling intervals tend to capture finer-grained changes in energy consumption and correspond more closely with time intervals. Larger sampling intervals might miss transient spikes or nuanced changes in energy usage on shorter duration. For example, the psys (system power) typically encompasses the entire board's power consumption and is normally higher than the sum of the values related to the CPU package and the GPU (see Figure 7). This consolidated measurement might show a higher correlation with execution time due to its broader scope.
- **Challenges with GPU Energy Reporting:** Nvidia-smi reports power rather than energy, causing inconsistencies when attempting to correlate it directly with energy values from other components. This can be a limitation on the way to consistent and accurate measurements, especially with low sampling intervals. In addition, the Nvidia driver significantly impacts the querying of power values, as at low frequencies (milliseconds and less), it returns *"Unknown"* values.
- **Time Overhead and Thread Joining:** When threads join at a report process, especially in scenarios with large intervals, that might cause long idle states of the threads. The time taken for these threads to join might become a significant overhead, which can impact the accuracy of execution time measurements.

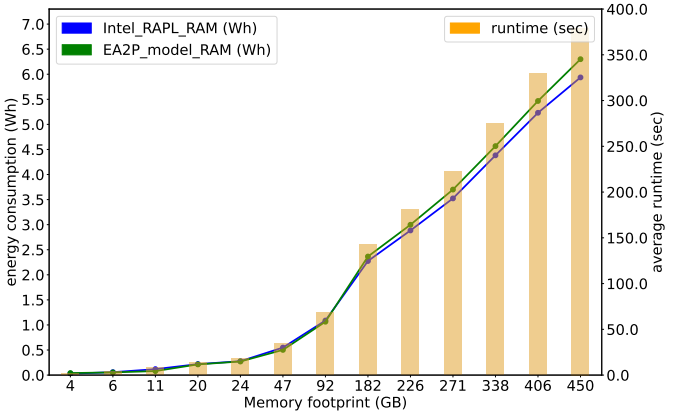


Fig. 6. EA2P RAM energy model vs Intel RAPL RAM domain on **Intel-2**.

*6) Illustrative MPI case:* In this section, we illustrate the ability of our tool to measure the energy of an MPI application through a matrix multiplication in MPI for Python, focusing on global and point-to-point communication modes.

We use a distributed computing system with four compute nodes (AMD-2 described in table II), each equipped with AMD Zen3 CPUs and Nvidia A100 GPUs. We leverage MPI with Python's `mpi4py` to the workloads of the matrix multiplication across the nodes, each `numpy.matmul`. Note that the energy reported for the GPUs is just indicative since they are idle in this case because `numpy` runs only on the CPU (denoted by Pkg in all tables).

- **Scenario 1: Collective communications (broadcast and gathering)**. In this scenario, each of the four nodes

TABLE VII
MULTI GPU ENERGY REPORT ON INTEL-2 WITH NVIDIA GPUS

| Apps | pkgs(Wh) | RAM(Wh) | GPU 0(Wh) | GPU 1(Wh) | GPU 2(Wh) | GPU 3(Wh) | GPU 4(Wh) | GPU 5(Wh) | GPU 6(Wh) | GPU 7(Wh) | time(sec) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sleep | 2.194 | 1.333 | 2.179 | 2.103 | 2.127 | 2.104 | 2.103 | 2.129 | 2.105 | 2.143 | 181.038 |
| DOG-CPU | **28.528** | 5.407 | 5.633 | 5.419 | 5.505 | 5.413 | 5.399 | 5.490 | 5.418 | 5.524 | 495.096 |
| DOG-GPU | 1.219 | 0.388 | **2.519** | 0.811 | 0.816 | 0.804 | 0.810 | 0.816 | 0.802 | 0.813 | 52.459 |

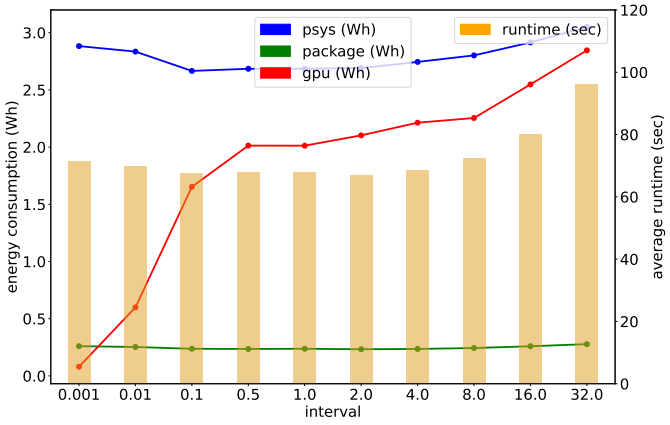


Fig. 7. Sampling interval behavior on **Intel-1**

computes a portion of the matrix and sends the result to the master node (rank 0), which aggregates the final result. The energy measurements for this scenario are reported in tables VIII and IX.

- **Scenario 2: Point-to-Point Communication (with 2 of 4 Nodes)**. Here, only two of the four nodes participate in the global matrix multiplication and cooperate though with point-to-point communication. Energy measurement is still conducted across all four nodes, which helps identify energy consumption patterns in the idle mode (for passive nodes). The energy measurements for this scenario are reported in tables X and XI

TABLE VIII
MEASUREMENTS ON **AMD-2** WITH $2^{14} \times 2^{14}$ MATRICES (1)

| Node | Pkg(J) | GPU 0 (J) | GPU 1 (J) | GPU 2 (J) | GPU 3 (J) | RAM (J) | Time (s) |
|---|---|---|---|---|---|---|---|
| 0 | 1493.59 | 648.52 | 680.59 | 649.10 | 666.36 | 201.6 | 13.25 |
| 1 | 1419.99 | 714.36 | 633.82 | 625.94 | 670.95 | 193.2 | 12.54 |
| 2 | 1457.43 | 664.03 | 666.67 | 652.40 | 719.44 | 201.6 | 13.03 |
| 3 | 1472.65 | 667.21 | 653.88 | 659.93 | 689.50 | 201.0 | 13.11 |
| Total | 5843.66 | 2694.11 | 2634.95 | 2587.35 | 2746.25 | 596.4 | 51.93 |

TABLE IX
MEASUREMENTS ON **AMD-2** WITH $2^{15} \times 2^{15}$ MATRICES (2)

| Node | Pkg(J) | GPU 0 (J) | GPU 1 (J) | GPU 2 (J) | GPU 3 (J) | RAM (J) | Time (s) |
|---|---|---|---|---|---|---|---|
| 0 | 9109.88 | 3374.76 | 3530.46 | 3319.83 | 3425.45 | 1058.4 | 74.66 |
| 1 | 8777.08 | 3674.04 | 3255.40 | 3208.25 | 3430.79 | 999.6 | 70.65 |
| 2 | 9012.43 | 3353.60 | 3363.65 | 3291.97 | 3638.53 | 1041.6 | 73.26 |
| 3 | 9090.88 | 3408.30 | 3347.95 | 3435.98 | 3442.57 | 1058.4 | 74.43 |
| Total | 35990.27 | 13810.70 | 13497.44 | 13256.03 | 13937.32 | 4158.0 | 293.00 |

TABLE X
MEASUREMENTS ON **AMD-2** WITH $10240 \times 10240$ MATRICES (1)

| Node | Pkg(J) | GPU 0 (J) | GPU 1 (J) | GPU 2 (J) | GPU 3 (J) | RAM (J) | Time (s) |
|---|---|---|---|---|---|---|---|
| 0 | 536.67 | 255.75 | 261.8 | 258.92 | 271.46 | 75.6 | 4.34 |
| 1 | 555.88 | 380.09 | 344.25 | 331.31 | 357.12 | 100.8 | 5.85 |
| 2 | 94.84 | 113.97 | 123.36 | 128.24 | 134.06 | 33.6 | 1.24 |
| 3 | 94.09 | 112.98 | 121.65 | 135.57 | 147.30 | 33.6 | 1.24 |
| Total | 1281.48 | 862.78 | 851.09 | 854.02 | 909.93 | 243.6 | 12.67 |

TABLE XI
MEASUREMENTS ON **AMD-2** WITH $20240 \times 20240$ MATRICES (2)

| Node | Pkg(J) | GPU 0 (J) | GPU 1 (J) | GPU 2 (J) | GPU 3 (J) | RAM (J) | Time (s) |
|---|---|---|---|---|---|---|---|
| 0 | 3604.08 | 1285.74 | 1374.71 | 1268.67 | 1314.77 | 403.2 | 28.09 |
| 1 | 3674.26 | 2200.34 | 1925.82 | 1892.41 | 2033.96 | 588.0 | 39.34 |
| 2 | 388.52 | 279.94 | 291.57 | 276.03 | 305.47 | 84.0 | 4.62 |
| 3 | 359.49 | 275.35 | 272.37 | 297.48 | 301.01 | 84.0 | 4.61 |
| Total | 8026.35 | 4041.37 | 3864.46 | 3734.58 | 3955.20 | 1159.2 | 76.66 |

## VI. CONCLUSION

We have designed and validated an operational framework for energy profiling. Our tool is hardware/system flexible and can be easily adapted to newer devices. Our experimental validation demonstrates the accuracy and consistency of its measurements, and also its portability across various devices. Our API allows the user to instrument python program for fine-grain profiling or to use command-line mode for binary codes (a way to handle compiled high-level languages). The tool can handle the major parallel models (shared memory, distributed memory and GPU-acceleration).

For future work, we plan to implement robust error handling mechanisms and logging functionalities to help handling troubleshooting.

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

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

[3] EnergyInovation, "How much energy do data centers really use?" 2020, accessed: 2024-08-29. [Online]. Available: https://energyinnovation.org/2020/03/17/how-much-energy-do-data-centers-really-use/

[4] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of bloom, a 176b parameter language model," *ArXiv*, vol. abs/2211.02001, 2022.

[5] M. Lang and R. Hendrikx, "Energy outlook 2023: Energy digitalisation," 2023, accessed: 2024-08-29. [Online]. Available: https://www.twobirds.com/en/insights/2023/global/energy-outlook-2023-energy-digitalisation

[6] Firefox, "Energy estimates," 2023, accessed: 2024-08-29. [Online]. Available: https://firefox-source-docs.mozilla.org/performance/perf.html

[7] H. McCraw, J. Ralph, A. Danalis, and J. Dongarra, "Power monitoring with papi for extreme scale architectures and dataflow-based programming models," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, 2014, pp. 385–391.

[8] 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.

[9] Intel, "Intel powergadget," 2019, accessed: 2024-08-29. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.htm

[10] ——, "Powertop," 2020, accessed: 2024-08-29. [Online]. Available: https://github.com/fenrus75/powertop

[11] Linaro, "Linaro forge release 23.0," 2023, accessed: 2024-08-29. [Online]. Available: https://docs.linaroforge.com/23.0/userguide-forge.pdf

[12] NERSC, "Craypat," 2017, accessed: 2024-08-29. [Online]. Available: https://docs.nersc.gov/tools/performance/craypat/

[13] A. Knüpfer, C. Rössel, D. a. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-p: A joint performance measurement run-time infrastructure for periscope,scalasca, tau, and vampir," in *Tools for High Performance Computing 2011*, H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 79–91.

[14] J. P. Gutiérrez Hermosillo Muriedas, K. Flügel, C. Debus, H. Obermaier, A. Streit, and M. Götz, "perun: Benchmarking energy consumption of high-performance computing applications," in *Euro-Par 2023: Parallel Processing: 29th International Conference on Parallel and Distributed Computing, Limassol, Cyprus, August 28 – September 1, 2023, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 17–31.

[15] Inria, "Welcome to pyjoules's documentation!" 2019, accessed: 2024-08-29. [Online]. Available: https://pyjoules.readthedocs.io/en/latest/

[16] B. GAMM, "Codecarbon," 2020, accessed: 2024-08-29. [Online]. Available: https://mlco2.github.io/codecarbon/index.html

[17] S. Budennyy, V. Lazarev, N. Zakharenko, A. Korovin, O. Plosskaya, D. Dimitrov, V. Akhripkin, I. Pavlov, I. Oseledets, I. Barsola *et al.*, "Eco2ai: carbon emissions tracking of machine learning models as the first step towards sustainable ai," in *Doklady Mathematics*. Springer, 2023, pp. 1–11.

[18] LLNS, "Variorum," 2023, accessed: 2024-08-29. [Online]. Available: https://variorum.readthedocs.io/en/latest/

[19] L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems, July 2020, arXiv:2007.03051.

[20] Tracarbon, "Tracarbon documentation," 2023, accessed: 2024-08-29. [Online]. Available: https://fvaleye.github.io/tracarbon/documentation/

[21] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *Journal of Machine Learning Research*, vol. 21, no. 1, jan 2020.

[22] L. Lannelongue, J. Grealey, and M. Inouye, "Green algorithms: Quantifying the carbon footprint of computation," *Advanced Science*, vol. 8, no. 12, p. 2100707, 2021.

[23] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A.-C. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: focus on CPU and GPU," in *CCGrid 2023 - 23rd IEEE/ACM international symposium on cluster, cloud and internet computing*. Bangalore, India: IEEE, May 2023, pp. 1–13.

[24] 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.

[25] R. Appuswamy, M. Olma, and A. Ailamaki, "Scaling the memory power wall with dram-aware data management," *Proceedings of the 11th International Workshop on Data Management on New Hardware*, 2015.

[26] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana, "Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions," in *High Performance Computing*, J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, Eds. Cham: Springer International Publishing, 2017, pp. 394–412.

[27] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier, "Powerapi: A software library to monitor the energy consumed at the process-level," *ERCIM News*, 2013.

[28] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, 2015.

[29] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 3, pp. 1 – 26, 2018.

[30] R. Schöne, T. Ilsche, M. Bielert, M. Velten, M. Schmidl, and D. Hackenberg, "Energy efficiency aspects of the amd zen 2 architecture," *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 562–571, 2021.

[31] NVIDIA, "nvidia-smi - nvidia system management interface program," 2016, accessed: 2024-08-29. [Online]. Available: https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf

[32] AMD, "Rocm - smi," 2014, accessed: 2024-08-29. [Online]. Available: https://sep5.readthedocs.io/en/latest/ROCm_System_Managment/ROCm-System-Managment.html

[33] Samsung, "Here to upgrade the world: Introducing samsung's game-changing ddr5 solution," 2022, accessed: 2024-08-29. [Online]. Available: https://semiconductor.samsung.com/news-events/tech-blog/here-to-upgrade-the-world-introducing-samsungs-game-changing-ddr5-solution/

[34] Crucial, "How much power does memory use?" 2019, accessed: 2024-08-29. [Online]. Available: https://www.crucial.com/support/articles-faq-memory/how-much-power-does-memory-use

[35] Buildcomputers.net, "Power consumption of pc components in watts," 2013, accessed: 2024-08-29. [Online]. Available: https://www.buildcomputers.net/power-consumption-of-pc-components.html

[36] I. Wallossek and C. Angelini, "Measuring ddr4 power consumption," 2014, accessed: 2024-08-29. [Online]. Available: https://www.tomshardware.com/reviews/intel-core-i7-5960x-haswell-e-cpu,3918-13.html

[37] M. Technology, "How much power does memory use?" 2024, accessed:2024-08-29. [Online]. Available: https://www.crucial.com/support/articles-faq-memory/how-much-power-does-memory-use

[38] Skhynix, "256gb ddr4 rdimm/lrdimm ultra-high capacities at industry's lowest power budget," 2021, accessed: 2024-08-29. [Online]. Available: https://product.skhynix.com/products/dram/module/ddr4dm.go

[39] L. Dalcin and Y.-L. L. Fang, "mpi4py: Status update after 12 years of development," *Computing in Science & Engineering*, vol. 23, no. 4, pp. 47–54, 2021.

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ArXiv*, vol. abs/1409.1556, 2015.

[41] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

[42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[43] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel dataset for fine-grained image categorization," in *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.