



HAL
open science

Dissecting the software-based measurement of CPU energy consumption: a comparative analysis

Guillaume Raffin, Denis Trystram

► To cite this version:

Guillaume Raffin, Denis Trystram. Dissecting the software-based measurement of CPU energy consumption: a comparative analysis. *IEEE Transactions on Parallel and Distributed Systems*, 2024, 36 (1), pp.96. 10.1109/TPDS.2024.3492336 . hal-04420527v3

HAL Id: hal-04420527

<https://hal.science/hal-04420527v3>

Submitted on 23 Dec 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Dissecting the software-based measurement of CPU energy consumption: a comparative analysis

Guillaume Raffin^{*†}, Denis Trystram^{*}

^{*}Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France

[†] Bull SAS (Eviden, Atos group), France

{*guillaume.raffin, denis.trystram*}@univ-grenoble-alpes.fr

Abstract—Every day, we experience the effects of the global warming: extreme weather events, major forest fires, storms, global warming, etc. The scientific community acknowledges that this crisis is a consequence of human activities where Information and Communications Technologies (ICT) are an increasingly important contributor.

Computer scientists need tools for measuring the footprint of the code they produce and for optimizing it. Running Average Power Limit (RAPL) is a low-level interface designed by Intel that provides a measure of the energy consumption of a CPU (and more) without the need for additional hardware. Since 2017, it is available on most computing devices, including non-Intel devices such as AMD processors. More and more people are using RAPL for energy measurement, mostly like a black box without deep knowledge of its behavior. Unfortunately, this causes mistakes when implementing measurement tools.

In this paper, we propose to come back to the basic mechanisms that allow to use RAPL measurements and present a critical analysis of their operations. In addition to long-established mechanisms, we explore the suitability of the recent eBPF technology (formerly and abbreviation for extended Berkeley Packet Filter) for working with RAPL. For each mechanism, we release an implementation in Rust that avoids the pitfalls we detected in existing tools, improving correctness, timing accuracy and performance. These new implementations have desirable properties for monitoring and profiling parallel applications.

We also provide an experimental study with multiple benchmarks and processor models (Intel and AMD) in order to evaluate the efficiency of the various mechanisms and their impact on parallel software. These experiments show that no mechanism provides a significant performance advantage over the others. However, they differ significantly in terms of ease-of-use and resiliency.

We believe that this work will help the community to develop correct, resilient and lightweight measurement tools.

Index Terms—energy consumption, energy efficiency, performance analysis, software measurement, RAPL library (Running Average Power Limit)

I. INTRODUCTION

A. Context and motivation

The impact of ICT in the environmental crisis has unfortunately increased, and it is likely going to continue increasing [1, 2, 3, 4, 5]. The carbon footprint of the field was evaluated to about 1.2 to 2.2 Gigatons equivalent CO₂ (denoted in short by GtCO₂e) in 2020, which corresponds to about 2.1 to 3.9% of worldwide greenhouse gases (GHG) emission [6]. They are expected to reach 5.1 to 5.3 GtCO₂e by 2040 [3]. Yet, ICT are also often presented as an effective solution for decreasing the environmental footprint of other sectors, mostly

thanks to optimization and substitution effects [7, 8]. Some reports hence claim that ICT's environmental benefits can be several times greater than their own environmental burden [9]. Nevertheless, the assessment of these benefits generates a lot of controversy [6, 10, 11], as the global pressure of humanity on the environment keeps on increasing [12, 13]. In this context, measuring the energy consumption of ICT is required to become aware of its impact (such as its carbon footprint) and establish environmentally friendly practices such as optimization efforts, power capping, restriction use, etc.

Section 4 of the paper of Khan et al. [14] shows that energy efficiency is increasing in the Top500, but at a lower rate as computing performance. *Aurora* became the second machine to reach exascale in June 2024 [15], with a record-breaking consumption of 38.7 MW with only about 40% of its nodes. This generates about 270 tons of CO₂e emissions a day (according to Illinois 2022 electricity profile). With such an impact, HPC systems are more and more studied from the perspective of energy efficiency. Assessing the energy efficiency of a system calls for measuring the energy consumption.

There are multiple families of tools that can be used to collect the energy consumption ranging from physical wattmeters (e.g. connected PDUs – Power Distribution Units – or BMCs – Baseboard Management Controllers) to web browser extensions that run estimation models. They have different accuracy, acquisition frequency and ease-of-use.

In this paper, we focus on the Running Average Power Limit (RAPL) technology [16], which is the basic building block for many software measurement tools. RAPL measures the electricity consumption of the CPU and more components thanks to sensors integrated into the system-on-chip, and exposes it to the operating system through model-specific registers (MSR in short) designed by the CPU manufacturer. The advantage of RAPL is that no external powermeter is required, nor a privileged access to the BMC (which could be used to power off the server). It also provides more details than a PDU or a BMC, since it monitors internal components of the node. Moreover, RAPL is more accurate than any statistical estimation model, even though they can be tuned to reduce their error [17, 18]. The underlying philosophy of software measurement tools based on RAPL is that each device owner monitors the software running on it.

Extracting the measurements from the CPU requires an interface. One can either use the low-level RAPL MSR directly, or choose a higher-level interface provided by the

operating system. Linux provides two of them, namely, the Power Capping framework (powercap) and the Performance Counters subsystem (perf-events). These software interfaces query the processor registers provided by RAPL.

Sometimes, the same interface can be used in a number of very different ways. This is the case with perf-events, which can be read from user space or from kernel space using eBPF. We propose to study these two ways separately. This makes a total of four mechanisms that can be chosen to extract the consumption measurements from the CPU: MSR, powercap, perf-events in user space, and perf-events with eBPF. They are described in section V.

Higher level software measurement tools, such as CodeCarbon [19], PowerAPI [20] and Scaphandre [21], are based on these raw mechanisms. They use RAPL measurements to estimate the electricity consumption of each active process.

B. Objectives

The purpose of this work is to provide engineers and researchers that use or want to use RAPL with a deep understanding of its access mechanisms and of the associated best practices. From their point of view, the goal is to develop measurement tools that are correct, efficient, and maintainable over time.

In order to achieve this goal, we need to evaluate the qualitative differences between the possible measurement mechanisms, based on criteria such as required expertise level and resiliency. This highlights the features and trade-offs of the mechanisms. Furthermore, we assess the overhead of the various mechanisms. In particular, we address the following questions:

What is the overhead on the other programs of measuring the energy consumption of the CPU? What is the impact of the measurement on an idle machine? Are some mechanisms more efficient than others? Is there a difference between Intel and AMD processors? To the best of our knowledge, these issues have not been fully addressed yet.

C. Contributions

To achieve the previous objectives, we propose a minimal measurement tool that includes an implementation of each mechanism. We release it with an open-source license in order to provide a reference implementation to the community. We test different strategies to avoid the common mistakes that we detected in other tools. Furthermore, we analyze and compare the four measurement mechanisms.

Using this minimal tool, we experimentally study the impact of the measurement on parallel software and on an idle server. Each mechanism has been benchmarked with several HPC testbeds, CPU vendors and RAPL domains. The benchmark results have been accumulated during approximately one month, with the aim to reduce the uncertainty of the statistical analysis.

In light of the comparative analysis and of the benchmark results, we are able to give a recommendation on the choice of the right mechanism.

II. RELATED WORKS

Hackenberg et al. evaluated the accuracy of RAPL measurements, in 2013 [22] and 2015 [23], using the MSR mechanism. Measurement overhead was not considered.

Huang et al. [24] showed that RAPL measurements were quite accurate for Haswell-EP processors. They assessed the performance overhead of the PAPI library. The different low-level mechanisms were not investigated.

Descrochers et al. [25] compared the values returned by RAPL for the RAM with power measurements obtained by instrumenting the hardware. Like the previous papers, this one concentrated on RAPL accuracy, not on its overhead.

In 2018, Khan et al. [26] evaluated the accuracy, update frequency and performance overhead of RAPL measurements on several parallel benchmarks. Their work focused on the MSR mechanism and did not compare it with others.

Several works were dedicated to the construction of statistical models that estimate the energy consumption of software applications [18, 17, 27]. While they compared their results with RAPL measurements, investing the various mechanisms was out of their scope.

In 2021, Schole et al. analyzed the measurements of AMD's implementation of RAPL on the Zen 2 architecture, and revealed that it provided less information than Intel's [28]. One mechanism was operated, and the study concentrated on assessing the reliability of AMD's implementation.

More recently, Jay et al. [29] compared some high-level tools that internally use RAPL. They evaluated their accuracy and overhead on NAS parallel benchmarks. We also use the NAS benchmarks, and conduct a more robust statistical analysis of their results. A similar study was proposed by Huguerte et al. for AI models [30]. We go deeper than both papers and analyze the underlying mechanisms instead of the higher level tools.

Due to the amount of work on RAPL measurements validation, we consider that this technology is reliable enough. Therefore, we choose not to assess again the correlation between RAPL values and external measurements. Instead, we focus on a comparative analysis of the measurement mechanisms, based on qualitative criteria as well as an experimental evaluation of their performance and energy overhead.

III. LIMITATIONS OF THE EXISTING APPROACHES

To understand why it is relevant to open the black box of RAPL, let us begin by an overview of the pitfalls that we have detected in many existing RAPL-based tools. We believe that these limitations result from the lack of a precise understanding of RAPL and the interfaces provided by the Linux kernel on top of it.

A. Unclear measurement perimeter

RAPL, that stands for Running Average Power Limit, is a power management technology first implemented by Intel in the Sandy Bridge architecture, in 2011. It allows to measure the energy consumption and to limit the power consumption of various parts of the computing device, called domains. In this paper, we only look at the measurement interface, not the

power limit. AMD followed suit by implementing a similar measurement interface in the first Zen architecture, in 2017. AMD's interface is so similar that it is used in the same way as Intel's. The difference lies in the addresses of the registers and in the number of available domains. In both cases, understanding what is measured by each domain is crucial.

Unfortunately, the scopes of the domains are not clearly explained in the documentation of the Linux kernel interfaces that are used by the measurement tools. The Intel Software Developer Manual [16] provides more information, but it remains vague about the most recent domains such as "*psys*". Consequently, several contradictory descriptions of the domains can be found online, and it is difficult to understand what is really measured.

By completing the documentation with experiments, we offer a new representation of all the RAPL domains known to date, which can be seen in figure 1. Khan et al. [26] published a similar figure, but we provide here some clarifications and corrections, especially with regard to the role of "*psys*".

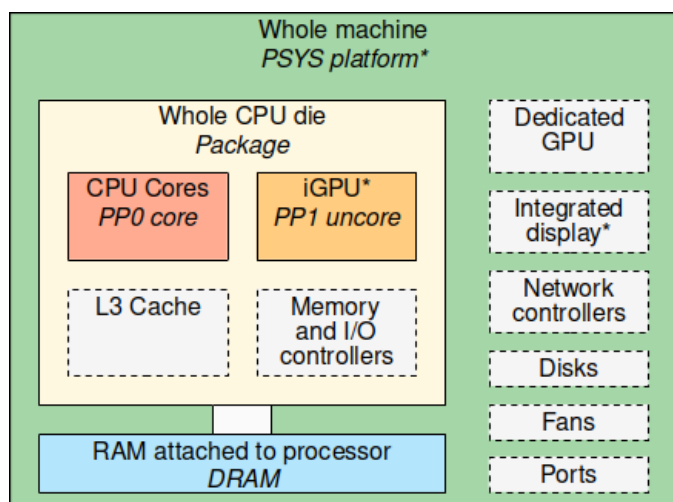


Figure 1. Hierarchy of the possible RAPL domains and their corresponding hardware components. Domain names are in italic, and grayed items do not form a domain on their own. Items marked with an asterisk are not present on servers.

First, figure 1 shows that the *core* and *uncore* domains are subsets of the *package* domain. This means that their reported consumption is included into the consumption of the *package* domain. Second, let us take a look at the *platform* domain, also known as *psys*. According to Intel [31], its content depends on the manufacturer of the machine. To understand its actual scope, we used recent laptops (Lenovo Thinkpad L15 Gen1 and Gen2, and Alienware m17 R3) and plugged each of them into an external wattmeter. By comparing the energy consumption reported by the wattmeter with the consumption reported by the RAPL domains, we discovered that the *psys* domain reported the same consumption as an external wattmeter, that is, the total consumption of the laptop. This domain can therefore include the display, dedicated GPU and all the other domains. Third, we want to make clear that, while the *psys* domain is unique (i.e. there is only one *psys* for the entire system-on-chip), the *dram* domain is linked to a specific CPU socket (i.e. there is one *dram* per socket). This clarifies

what the Intel manual means by "directly-attached RAM". We have tested this using multi-CPU servers Gigabyte H261-Z60. Finally, AMD's microarchitectures Zen 1 to Zen 4 only support the *package* and *core* domains.

B. Overflows that distort measurements

Measurements provided by RAPL have been evaluated by prior work to be accurate. However, bugs in measurement software can distort this accuracy. In particular, a potential difficulty of RAPL is that the energy counters overflow. These overflows occur after approximately "60 seconds under heavy load" according to Intel's manual [16], and the only way to detect them is to poll the value frequently enough (the CPU does not signal them). It may appear as a trivial bug, however in this case it can actually be hard to detect, because the overflow frequency depends on the consumption of the machine. Testing the program on a mainstream laptop or on a server that is mostly idle can therefore make it invisible to the tester's eyes. Furthermore, if the user look at aggregated indicators, detecting the error becomes even harder.

We have found that many software measurement tools have been affected by bugs related to this overflow. Not all tools have resolved the issue as of this writing [32, 33, 34], which could cast doubt on the measurements they provide. A robust solution is presented in section IV-B.

C. Lack of accurate control of the acquisition frequency

As stressed previously, the acquisition frequency is a key point of a RAPL-based measurement tool. Besides the overflow correction, one could wish to increase the frequency in order to capture a more precise profile of the running applications. For example, estimating the consumption of a single function has been proven to be possible in 2012 [35]. To do so, we have to know the beginning and end of the function, and to maintain an acquisition frequency that is high enough to distinguish the different function calls, but low enough to minimize the disruption of the system. Parallel applications, which use multiple cores and call complex libraries, require even more caution. For these reasons, it is necessary to give control of the frequency to the end user, and to make the tool precisely follow the supplied frequency.

Unfortunately, this is not the case with the measurement tools that we tested. Usually, the acquisition frequency is capped to 1 Hz, and the actual frequency differs from the target. We found that the implementation of the polling loop had a great influence on the reliability of its timing. In particular, using a traditional `sleep` function (like Scaphandre [21], PyJoules [36] and others) proved to be unreliable: as assessed by our experiments, the time between each measurement varies and the high frequencies cannot be achieved. Thus, we believe that this approach cannot be used for energy profiling. We were able to overcome this issue by implementing a minimal tool using a different technique, which is described in section IV-C.

D. Performance and quality concerns

Every tool that we tested only supports one RAPL-based measurement mechanism, but does not justify its choice.

Some tools like Likwid [37] use an MSR-based mechanism, while others like CodeCarbon [19] use a powercap-based mechanism. Most of the works presented in section II are based on the MSR. Using the low-level MSR seems harder a priori, but could have been chosen to obtain a performance advantage. We show that there is actually no such advantage in section VI-B.

The choice of the mechanism also raises a quality concern. First, it impacts the amount of work required to create the tool and maintain it over time. We analyze the qualitative aspects of RAPL-based mechanisms in section V. Second, while implementing our minimal measurement tool, we found that bugs in the Linux kernel could impede the operation of the interfaces. On the AMD server used to conduct our benchmarks, powercap and perf-events did not list the same available RAPL domains. Powercap listed two domains: *package* and *core*, whereas perf-events listed all the possible domains. It turned out that only the *package* domain made sense for this CPU model. The bug has been fixed in Linux kernel version 6.6 [38], but datacenters take time to update their systems. Supporting multiple mechanisms would allow such bugs to be detected automatically, making the tools more robust.

IV. IMPLEMENTATION OF A MINIMAL TOOL

A. Architecture

To evaluate some solutions to the aforementioned issues and compare the four RAPL-based mechanisms, we implemented a minimal measurement tool in Rust. Its architecture is depicted on figure 2.

It features a command-line interface that allows the user to choose the domains to measure, the mechanism to use, the acquisition frequency of the polling loop, and the output file. We chose to always write the measurements during our benchmarks, in the CSV format, in order to be more representative of a real situation. Indeed, discarding the values as soon as we read them would be useless. Thus, the impact that we evaluate in section VI includes the cost of saving the measurements. To limit this cost when the acquisition frequency is high, while regularly making the data available to the user, we chose to flush the measurements to the output file once per second.

The minimal tool allows to access the “raw” energy consumption measurements, not a derived metric like Code Carbon [19]. We believe that this is more rigorous and that it gives more freedom to use the measurements in a way that is suited to the context. Furthermore, it allows to ensure that the overflow correction presented in section IV-B is correct.

More information is available on a public Git repository ¹.

B. Correcting the overflow of the counters

As mentioned in section III-B, RAPL energy measurements are prone to overflows, whose frequency depends not only on the energy consumption of the domains but also on the measurement mechanism, because they store the values in variables of different sizes and sometimes apply transformations

¹<https://github.com/TheElectronWill/cpu-energy-consumption-comparative-analysis>: minimal tool with an implementation of each studied mechanism

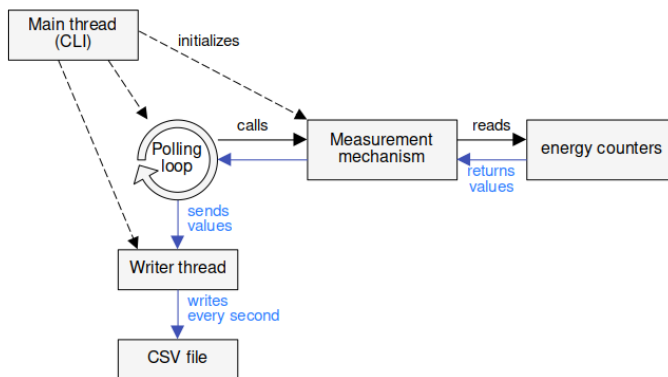


Figure 2. Architecture of our measurement tool

on them. To ensure the correctness of the measurements, two conditions must be met. First, the time between two readings of the energy consumption must be smaller than the time it takes for an overflow to occur. This allows to detect an overflow between two successive measurements m_{prev} and $m_{current}$, as highlighted by previous work on RAPL. Second, the following correction must be applied:

$$\Delta m = \begin{cases} m_{current} - m_{prev} + C & \text{if } m_{current} < m_{prev} \\ m_{current} - m_{prev} & \text{otherwise} \end{cases}$$

where C is a correction constant that depends on the chosen mechanism. Table I gives the right value of C for each mechanism. Only then can Δm be used as a measure of the energy consumption of the RAPL domain during the small time period between the two measurements.

mechanism	constant C
MSR	u32::MAX i.e. $2^{32} - 1$
perf-events	u64::MAX i.e. $2^{64} - 1$
perf-events with eBPF	u64::MAX i.e. $2^{64} - 1$
powercap	value given by the file <code>max_energy_uj</code> in the <code>sysfs</code> folder of the RAPL domain

Table I
TABLE OF THE OVERFLOW CORRECTION CONSTANT FOR EACH MEASUREMENT MECHANISM

C. Ensuring the accuracy of the acquisition frequency

When we implemented the tool, we found that using the `sleep` function in the polling loop was not reliable enough. The variations between each sleep were big enough to be detectable, and it was impossible to reliably reach a frequency of 1000 Hz. To solve this issue, we replaced the standard sleep by `timerfd`. While it cannot guarantee to perfectly respect the frequency in all conditions, especially on a non-realtime kernel, we assessed its superiority over a standard sleep.

In addition to using a more precise sleep function, we designed the tool so that the polling loop only does the minimum amount of work required to gather the measurements. Writing the data to a file is delegated to another thread. This way, the polling loop is not impacted by I/O latency.

To evaluate the effect of these two optimizations, we built two versions of the minimal tool: one that included the optimizations, and one that did not. We executed them on an idle

laptop, gradually increased the target frequency, and computed the actual output rate, i.e. the number of measurements per second in the result file. The first and last second were removed from the data, in order not to skew the statistics with incomplete measurement periods.

We did the same experiment with two other tools: Scaphandre [21] and CodeCarbon [19]. We chose them because they are among the most popular open-source RAPL-based tools, and because they differ in some key implementation choices. In particular, Scaphandre [21] uses a `sleep`-based timing loop and CodeCarbon [19] uses a timer library for Python. We modified them to allow measurement frequencies above 1 Hz. To keep the comparison fair, we also disabled a part of their code in order to only collect the raw RAPL measurements. Their measurement perimeter was thus identical to our minimal tool. Figure 3 shows the results obtained by each measurement tool on the laptop.

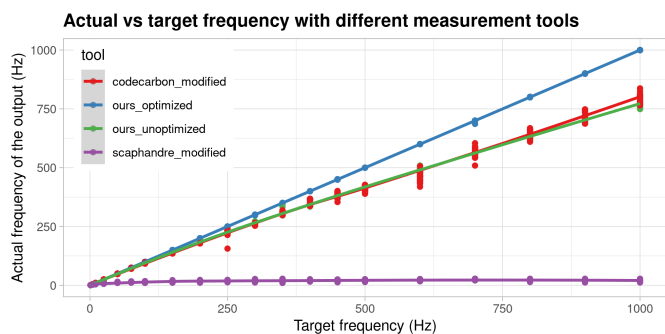


Figure 3. Plot of the actual measurement frequency achieved by the tools for each target frequency supplied as a command-line argument (on a Lenovo Thinkpad L15 Gen1 laptop)

As shown on the plot above, only the optimized version of the minimal tool (in blue) is able to consistently reach the target frequency of 1000 Hz with a very low standard deviation of 0.7. With the unoptimized version, the actual frequency decreases to about 847 Hz (15% lower than the target) and the standard deviation increases to about 20. We also tested that enabling only one of the two optimizations is not enough to obtain the best results. This demonstrates the benefit of combining them in an efficient measurement loop.

A surprising result is that, while CodeCarbon is written in Python and Scaphandre is written in Rust, the former performs significantly better than the latter. The modified version of CodeCarbon exhibits a frequency shift that is similar to our unoptimized tool, whereas the modified version of Scaphandre is unable to go beyond 28 Hz. This probably indicates that the implementation of Scaphandre is suboptimal.

CodeCarbon, despite using a timer-based loop, does not perform better than our unoptimized tool, which is sleep-based. Furthermore, it consumes significantly more CPU time by constantly using about 37% of one core at 1000 Hz, whereas our unoptimized tool uses only 8% at most.

We replicated the experiments on a server with two Intel Xeon Gold 6130 CPUs and obtained similar results. A notable difference was the improvement caused by the separate I/O thread, which we found to be smaller on the server than on

the laptop. This can be explained by the size of the OS disk cache, which is larger on the server.

Overall, the experiments show that the two simple optimizations of the minimal measurement tool have made it accurate, and allowed it to achieve a high measurement frequency. This is a clear improvement over existing RAPL-based tools, even when the latter are modified to allow high frequencies.

D. Brief discussion about limitations

Although it allowed us to perform extensive experiments, the minimal tool that we built is limited in some ways.

First, it does not support the hot-plugging of CPUs, because this feature was useless for our experiments. It could be added by using the Linux hotplug API to be notified when a CPU goes offline or online [39].

In addition, our work is strictly limited to Linux. Although the MSR should be accessible from other operating systems, we have not tried to use them. The low-level access of the registers would be similar but the higher-level interfaces, if any, would be different. To the best of our knowledge, Windows does not provide a sysfs-like interface to access RAPL measurements.

Finally, we have focused here on x86 Intel and AMD processors. ARM processors have not been studied because, as far as we know, most of them do not provide a RAPL-like interface. It seems to be possible on some platforms, though, since the *powermetrics* tool [40] is able to report the instantaneous power of Apple’s ARM chips. Incorporating another OS and another low-level interface into the minimal tool would require more exploratory and technical work.

V. COMPARATIVE ANALYSIS OF THE RAPL-BASED MECHANISMS

A. Comparison criteria

Section III-A introduced the fundamental operation of the RAPL technology, which provides energy counters associated to domains. To read these counters on Linux, several mechanisms can be implemented. By leveraging the experience that we obtained by building the minimal measurement tool, we now analyze and compare the possible choices. Three of them are relatively well-known, yet rarely understood in depth. We also offer a new way of reading RAPL counters and compare it to the existing ones.

All the mechanisms enable the same end result: obtaining, at a given acquisition frequency and for a given set of domains, the values of the energy consumption counters provided by the CPU’s RAPL interface. A measurement tool can be based on any of them, but that choice implies some trade-offs. In order to compare the mechanisms, we have retained the following criteria:

- the **technical difficulty** required for setting up the mechanism, i.e. the amount of time it takes to implement it, and the experience required for someone to do it.
- the amount of **knowledge** required to implement the mechanism properly, for instance whether we need to know some details of the CPU’s microarchitecture or

not. It depends on the abstraction offered by the system interface.

- the **safeguards** it offers to the developer, i.e. whether it is easy to make mistakes when using the mechanism
- the **privileges** required to execute the mechanism, e.g. whether the end user of the software tool that uses the mechanism has to be `root`
- the **resiliency** of the mechanism, i.e. its ability to adapt to changes, such as the release of a new generation of processors, a new RAPL domain, etc.

B. Model Specific Registers

At the lowest level, the CPU provides model specific registers (MSR) exposing the RAPL data [16]. A user program can read them by opening `/dev/cpu/N/msr`, where N is the number of the CPU core, and reading it at a specific offset. Intel and AMD use different offsets for the energy-related registers and, as far as we know, the only way to determine them is to check the processor model and to read the corresponding documentation. Furthermore, the values obtained from the registers are expressed in a machine-dependent unit. Is it therefore necessary to detect the unit at runtime by decoding some other registers. Every mechanism requires a conversion step, but it is harder to implement for the MSR.

For these reasons, measuring with the MSR requires expert knowledge about the processor. We would like to insist upon the fact that implementing a distinction between AMD and Intel processors is not sufficient: while vendors often use the same offsets and units across several generations of processors, there are exceptions. For instance, Intel's manual [16] gives a default value for the energy unit, but it does not apply to the *dram* domain of servers equipped with Xeon E5-2600 processors.

The MSR mechanism is low-level and provides no safeguard to the developer. Reading the registers at the wrong address, or converting the bits read in the wrong way leads to a wrong measurement that can be hard to detect. Thankfully, there should be no risk of breaking the operating system nor the hardware by misusing the MSR, because we only perform reading operations, never writings.

The kernel restricts access to the MSR because it can contain sensitive information. It has been shown that the RAPL counters could be used to perform side-channel attacks, turning into a security risk [41]. For this reason, even reading the registers requires high privileges: the binary of the measurement tool must be given the Linux capability `CAP_SYS_RAWIO` (or `CAP_SYS_ADMIN` on old kernels) or must be run as `root`. Furthermore, access control to the MSR is all-or-nothing: the `msr` module offers no way to limit a program to a restricted set of registers. This problem can be mitigated with the `msr-safe` module [42], but the other difficulties remain.

C. Power Capping Framework

The Power Capping framework (`powercap`) [43] is a software interface provided by the Linux kernel on top of the low-level RAPL interface. It allows

to control RAPL from userspace through the `sysfs` virtual file system. The hierarchy of the `sysfs` under `/sys/devices/virtual/powercap/intel-rapl` resembles the hierarchy of the RAPL domains, allowing tools to discover which domains are available on the computer.

There is one confusing difference, though: in contrast to figure 1, `powercap` puts the *dram* domain inside of the *package* domain. We have found no justification of this difference in the other works related to RAPL. We think that it can be explained by the fact that, in a multi-socket system, each CPU typically has its own directly-accessible memory. Since the registers are provided by the CPU, the *dram* energy counter of each CPU is different and only reports the consumption of the memory that is attached to it. `Powercap`'s authors would have chosen to reflect this by putting the *dram* domain next to the *core* domain, for each socket. Even so, our tests demonstrate that the energy consumption of the memory is not included in the energy consumption of the *package*.

Being a high-level interface, `powercap` is easier to use than the MSR. On one hand, the measurements are provided in text files, and the MSR unit conversion is automatically applied. This is handy for testing or developing simple scripts. On the other hand, overflows frequently occur, like with the MSR mechanism. Almost no knowledge about the processor is required, even if one must be careful with the meaning of the domains' hierarchy, as previously explained. No special case is required, because the Linux kernel takes care of them.

`Powercap` just needs read access on the `intel-rapl` directory. The file permissions or access control lists can be adjusted accordingly, which allows to limit the privileges of the measurement tools.

Finally, the resiliency of the mechanism is good, thanks to an automatic adaptation of the hierarchy of the `sysfs` to the available domains. Of course, to benefit from this advantage, the hierarchy must not be hard-coded in the measurement tool. Our reference implementation is able to adapt to the availability of the domains.

D. perf-events

The "perf events" subsystem (hereafter "perf-events") is another Linux kernel interface. It provides event-oriented performance monitoring capabilities [44]. There are two types of events: counting events, whose latest values must be polled, and sampling events, which are periodically added to a buffer. The RAPL energy counters have been integrated as counting events. Therefore, periodically polling the counters to detect the overflows is necessary, as with the other mechanisms. An advantage of perf-events is that overflows are already corrected by the subsystem. Of course, the value returned by perf-events has a limited size (64 bits integer), so an overflow is still possible in theory, but very unlikely in practice. Thus, the probability of reporting erroneous measurements is much lower than with MSR and `powercap`.

Usually, polling the events is done from userspace. First, the list of available events can be read from the `sysfs`, by inspecting the `/sys/devices/power/events/` directory. Each event corresponds to a RAPL domain and needs to be

opened by calling `perf_event_open` [45]. The obtained file descriptor can then be read periodically to access the measurements. Unlike `powercap`, `perf-events` does not organize the events in a hierarchy matching the different CPU sockets. The only difficulty that we have found is that each event must be opened exactly once for each socket. Using `perf-events` requires no expert knowledge and allows operating at a higher level of abstraction than the MSR mechanism.

In terms of resiliency, `perf-events` is less versatile than `powercap`, because one needs to call `perf_event_open`. It is therefore not available in every language, and it is much harder – though not impossible – to use it in Bash scripts. Yet, when used in a compatible environment, `perf-events` is as easy to maintain over time as `powercap`, and can easily be adapted to support new domains and microarchitectures.

Fine-grained access control is possible, since it requires either the `CAP_PERFMON` capability (`CAP_SYS_ADMIN` before Linux 5.8) or the kernel setting `perf_event_paranoid` set to 0 or -1.

We have analyzed here the usual mechanism implemented on top of `perf-events`. Another way of handling the events is to read the file descriptors from kernel space using the recent eBPF technology, as described in the following section. As far as we know, this technique has not been investigated yet.

E. perf-event via eBPF

Originally, eBPF was an abbreviation for “extended Berkeley Packet Filter”, but it is now a standalone name [46]. It refers to a technology that allows to inject code into a kernel, in particular the Linux kernel since version 3.15 [47]. Thanks to eBPF, a user program can be attached to an existing function of the kernel, which has proven to be useful for implementing fast packet filtering [48] and profiling.

In this context, we want to examine whether eBPF is a good fit for measuring the energy consumption with RAPL. The idea is that, under the hood, the aforementioned mechanisms will lead to a call to the `rdmsr x86` instruction, which needs to be done from the kernel. By reading the values with eBPF, all our measurement code would be executed by the kernel, and we would not need to switch from user mode to kernel mode. This could lower the overhead of the measurement. We test this hypothesis by designing a new measurement mechanism based on eBPF and by benchmarking its implementation (see section VI-A).

Figure 4 illustrates the inner workings of this new mechanism. As a preliminary step, `perf_event_open` is called and the returned file descriptors are passed to the eBPF program. This program is then injected into the Linux kernel and attached to a `SF_CPU_CLOCK` event. At a given frequency (1000 Hz in our implementation), the clock triggers the program, which reads the RAPL energy counters via `perf-events` and pushes the measurements into a buffer. At a regular interval managed by another timer, the userspace program polls the content of the buffer and obtains the energy consumption measurements.

Using eBPF is significantly more complicated than using the “regular” userspace variant of `perf-events`, that is why we evaluate its technical difficulty to “high” (see section V-F).

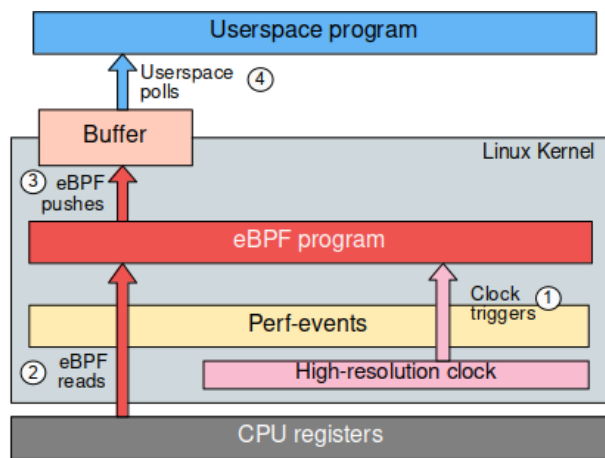


Figure 4. Measurement mechanism based on `perf-events` and eBPF

In terms of safeguards, it is just as robust to overflows as the regular `perf-events`, but other types of errors can occur. For example, we hit multiple difficulties related to the transfer of values between user and kernel space. Our implementation is written in Rust, which prevents certain types of error. We believe that it would have been even more challenging to write it in C.

Using eBPF requires to have the capability `CAP_BPF` (`CAP_SYS_ADMIN` before Linux 5.8) or to be run as root.

This mechanism demands a bit more work to adapt to changes, because the sizes of the buffers must be adjusted, and the injected code must be updated in consequence. In fact, old versions of the Linux kernel are still used in production, and they do not support loops in eBPF kernel code. That is why the implementation we propose uses a match-case on the number of RAPL domains. Supporting a new domain hence requires to update the code.

F. Synthesis

Table II below summarizes the results of the previous comparative analysis.

mechanism	technical difficulty	required knowledge	safeguards	privileges	resiliency
MSR	medium	CPU knowledge	none	<code>SYS_RAWIO</code> cap. + msr module	poor
perf-events + eBPF	high (long, complicated code)	limited	overflows unlikely, many other possible mistakes	<code>PERFMON</code> and BPF capabilities	manual tweaks necessary for adaptation
perf-events	low	limited	good, overflows unlikely	<code>PERFMON</code> capability	good
powercap	low	limited	beware of overflows	read access to one dir	good, very flexible

Table II
QUALITATIVE COMPARISON OF THE MEASUREMENT MECHANISMS

VI. EXPERIMENTAL STUDY OF THE MEASUREMENT OVERHEAD

A. Benchmarking Protocol

We describe here the protocol of the experimental study we conducted in order to assess the impact of the measurement mechanisms. This quantitative analysis completes the previous qualitative comparison. We are interested in the performance overhead on the other running applications, but also on the energy consumption of the machine when it is idle.

1) *Test Environment*: To run the benchmarks, we used two dedicated server-class machines in Atos' datacenters. The first one was equipped with two AMD EPYC 7702 64-cores processors and RHEL (Red-Hat Enterprise Linux) 8.7. The second was composed of two Intel Xeon E5-2680 v4 14-cores processors and ran RHEL 8.5. Both servers were running the Linux kernel version 4.18.0. On the AMD machine, the only domain available in every mechanism was the *package* domain. On the Intel machine, we were able to measure both *package* and *dram* domains.

We chose to run the well-known NAS benchmarks [49], mainly to allow our results to be compared with other papers such as the experiments of Jay et al. [29]. In order to get various types of parallel applications, we used three NAS benchmarks: BT (block tri-diagonal solver), CG (conjugate gradient) and EP (embarrassingly parallel). The benchmarks were compiled on each server and configured to use all the cores.

2) *Inputs and Outputs*: We recorded the running time of the NAS program, as reported by itself. Thus, this metric only includes the time it takes to solve the benchmark's problem, not the call to `posix_spawn`, nor the final clean-up of the program's memory, etc. We also performed some runs with a sleep of 20 minutes instead of a NAS program in order to evaluate the impact of the measurement on an idle CPU. During such a sleep, we gathered statistics about the state of the CPU with the `turbostat` command. During the experiments, we used a script to change the following variables:

- the NAS benchmark to execute (BT, CG, EP, or a 20-minutes sleep)
- the measurement mechanism to use (MSR, powercap, perf-events userspace, perf-events eBPF)
- the frequency of the measurement (0 i.e. no measurement, or 0.1Hz, 1Hz, 10Hz, 100Hz or 1000Hz)
- the RAPL domains to measure (*package* only, or *package* and *dram*)

That makes 96 combinations to test on the AMD server, and 192 on the Intel server. For every run of a combination, we recorded the aforementioned metrics, the variables, and the current time. At the same time, we recorded the evolution of the power consumption of each server by querying their BMC. This was done from another server, in order not to interfere with the benchmarks.

3) *Benchmark repetitions*: We ran the benchmarks for approximately one month, for as long as we could reserve the servers for our experiments. Instead of running many repetitions of one combination, then many repetitions of another combination, and so on for every combination until the end,

we spread the repetitions over time. As illustrated by figure 5, we ran 3 repetitions of the first combination, then 3 repetitions of the next one, etc. Once all the combinations were done, we started all over again: 3 repetitions of the first combination (the rightmost green square on figure 5), and so on.

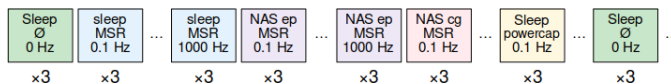


Figure 5. Benchmark repetitions spread over time

The goal of this spreading is to avoid bias we cannot control, such as another user executing something, a technical problem in the datacenter, a change in weather that affects the temperature of the room, etc. These external factors are likely to happen at a specific point in time, and thanks to this way of benchmarking they have a lower chance to spoil the results of a specific combination.

4) *Outliers*: After the execution of the benchmarks, we analyzed them with statistical tools. The first step was to eliminate the outliers in each combination. To identify them, we used the usual interquartile range. A value was considered to be an outlier if it was outside of the range $[q_1 - 3(q_3 - q_1), q_3 + 3(q_3 - q_1)]$, where q_1 is the first quartile and q_3 is the third quartile. After this step, we were left with 2186 observations for the AMD server and 2564 for the Intel one.

B. Results

1) *Impact on parallel software*: Figures 6 and 7 show the running time of the three NAS benchmarks on the AMD and Intel server respectively. Our goal here is to determine whether some mechanisms are more efficient than others, and whether increasing the frequency makes the potential performance overhead more visible.

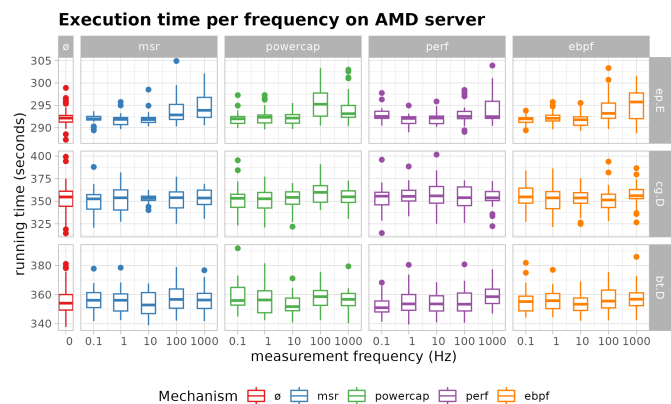


Figure 6. Performance impact of the measurement on NAS benchmarks (AMD server). The leftmost column contains the baseline.

Visually, it is difficult to see an overhead effect, even with a high measurement frequency. To test the existence of an overhead in a more rigorous way, we conducted a statistical test on each group. A group here is a subset of the data referring to the same server, NAS benchmark and RAPL mechanism. Our data did not meet the criteria for an ANOVA

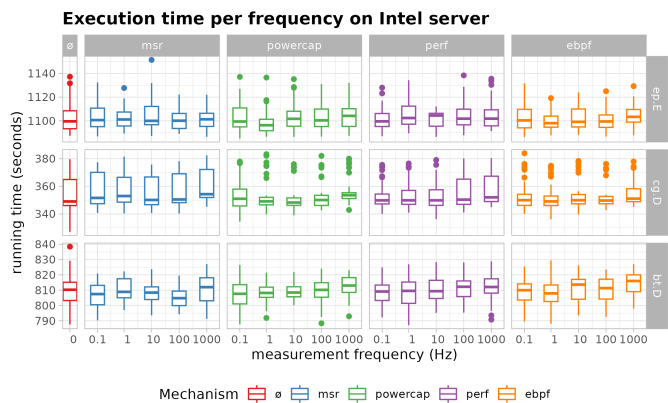


Figure 7. Performance impact of the measurement on NAS benchmarks (Intel server). The leftmost column contains the baseline.

test (analysis of variance), therefore we applied a one-sided Wilcoxon rank sum test. We used the standard value of 5% for the significance level α and applied the Holm-Bonferroni correction on the p-values. For each group we compared the running time of each frequency to the running time obtained without any measurement. The latter is represented by red boxplots on the left of figures 6, 7 and 8. Our alternative hypothesis for the statistical test is then “the running time with measurement is higher than without”.

The test shows that the overhead is statistically significant on the “ep.E” benchmark on the AMD server, at a frequency of 1000 Hz for msr, powercap and eBPF, and at a frequency of 100 Hz for powercap. While significant, the overhead remains small, with a location shift estimated at 1.5 to 3.4 seconds, that is, 0.5% to 1.2% (see appendix A). The location shift is an estimator of the median of the difference between a sample of the first group (with some measurement frequency) and a sample of the second group (without any measurement). Surprisingly, perf-events seems to have a smaller overhead than the other mechanisms on this benchmark. In particular, the low-level MSR interface seems slower than perf-events, which is counter-intuitive. We discuss this point in section VI-D. Regarding the other benchmarks, there is not enough evidence to say that measuring increases the running time.

The AMD server is not impacted by the measurement on all benchmarks. Only the “ep.E” benchmark, which consists of heavy floating-point operations is slowed down. The other benchmarks, which use more memory operations, are not significantly affected. Interestingly, the Intel server does not exhibit this behavior, since the only significant differences are on “cg.D” and msr, and “bt.D” and eBPF, with a location shift of respectively 5.6 seconds (1.6%) and 5.7 seconds (0.7%). An acquisition frequency of 10 Hz or less causes a negligible impact on the running time in every tested configuration.

In comparison, we measured that the modified versions of Scaphandre and CodeCarbon, which have a restricted feature set as described in section IV-C, have a non-negligible overhead of respectively 3-4% (depending on the benchmarks) and 0.5-0.8% at 10 Hz on the Intel server.

Note that figure 7 does not make a difference between the

benchmarks ran while measuring one RAPL domain and the benchmarks ran while measuring two RAPL domains. This is because, according to our analysis, that there is insufficient evidence to conclude in favor of a difference between these two cases. In other words, measuring one more domain does not change the overhead of the measurement on the benchmarks.

In addition to the running time, we recorded the power consumption of the entire servers, using their BMC. As can be seen on figure 8, we found that the measurement operation does not impact the average power when the machine is under heavy load. The two servers showed similar results.

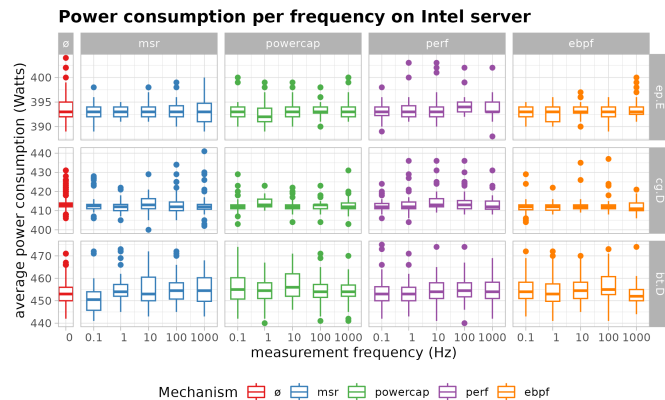


Figure 8. Impact of the measurement on power consumption during NAS benchmarks (Intel server). The leftmost column contains the baseline.

This lack of overconsumption under heavy load can be explained by the fact that, when running the benchmarks, the OS puts the CPUs at their maximum power (via DVFS: Dynamic Voltage and Frequency Scaling). Hence, it cannot increase further.

2) *Impact on idle CPU:* We have just showed that the impact of RAPL energy measurements on a loaded CPU was low. In this section, we show that there can be a significant impact on an idle CPU, depending on the measurement frequency. To evaluate this impact, we used the data collected during the “sleep” runs, which consist of a simple sleep of 20 minutes. In addition to the total power read with the BMC, we collected the C-States of the processor. In “C0” state, the processor is active and executing instructions. Other states are idle states, during which a subset of the CPU is disabled in order to save power. Looking at the C-States provides further insight about the interaction between DVFS and low-level measurement.

First, let us look at the average power consumption of the server during each run of the benchmarks. As we can see in figure 9, the two servers do not react in the same way.

On the AMD server, the power increases starting at a measurement frequency of 1000 Hz, for which the machine draws around 2.7 to 3.3 Watts more than without measurement. On the Intel server, a Wilcoxon rank sum test indicates that the effect is significant starting at 10 Hz for powercap and msr, and 100 Hz for perf-events and eBPF. The details are available in appendix B. At the highest frequency, the average power increases by 48.6 Watts for powercap, 40.0 for msr, 27.2 for perf-events and 25.4 for eBPF. These last two mechanisms,

Average power consumption vs. frequency

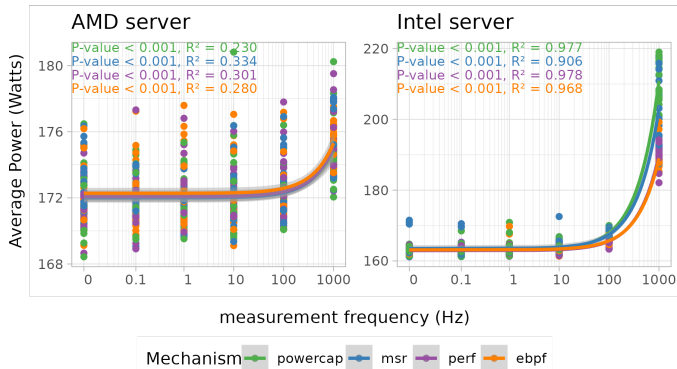


Figure 9. Relationship between acquisition frequency and idle power consumption (AMD and Intel server)

both based on the perf-events interface, have a clear advantage when the Intel processor is idle.

This difference between the two processors could be explained by the fact that the Intel one has more CPU states than the AMD one (note in general, Intel provides more C-States than AMD as of 2023). As confirmed by figure 10, querying the RAPL counters too frequently forces the Intel CPU to spend less time in C6, which consumes the least power, and more time in the higher levels, in particular C0, which consumes the most. Increasing the measurement frequency naturally causes more instructions to be executed, hence the increase of the time spent in C0. Interestingly, a high frequency of 1000 Hz makes the CPU use the intermediary states more often (C1, C1E and C3 on the Intel CPU), to the detriment of the deepest state (C6 on the Intel CPU). This means that the system determined that it was less appropriate to use the deepest state, perhaps because of its higher exit latency (it takes more time to exit C6 than to exit C3, C1E or C1). Whether Intel's or AMD's state management is optimal is not in the scope of this work, but it is interesting to know that using high measurement frequencies while the CPU is idle forces it to reduce its use of the deepest C-States. A good strategy would therefore be to dynamically adapt the measurement frequency to the CPU load.

The AMD CPU also spends less time in C2 and more time in C1 and C0, as can be seen on figure 11. But there is obviously less differences between AMD C2 and C0 than Intel C6 and C0. The different mechanisms behave similarly with regards to the C-States.

C. Measurement reading latency

The last experiment is a microbenchmark, that aims to assess the time it takes to read one value of a RAPL counter, using the various mechanisms. To build and run the microbenchmark, we used the Rust benchmarking tool *Criterion* [50] on three machines: the two previously mentioned servers, and a Lenovo Thinkpad L15 Gen1 laptop running Ubuntu 22.04 LTS with Linux 6.1.

Table III reveals that perf-events is the fastest mechanism, followed by MSR. Not only does perf-events have the lowest

CPU State versus frequency (Intel server)

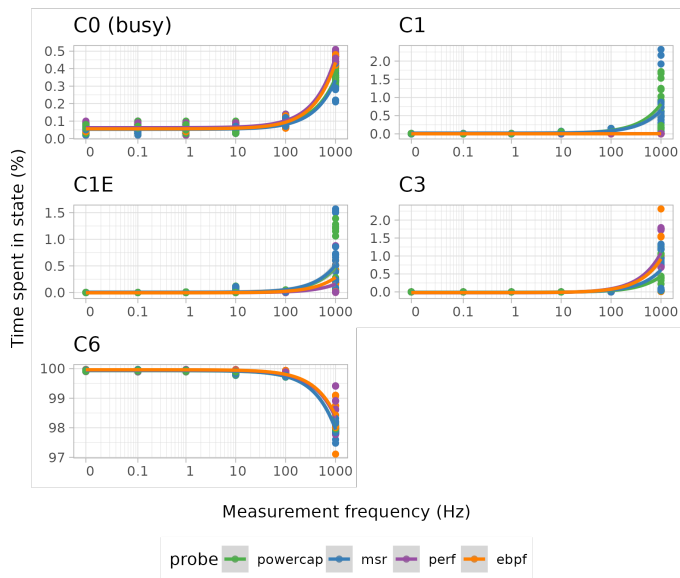


Figure 10. Relationship between acquisition frequency and idle CPU states (Intel server)

CPU State versus frequency (AMD server)

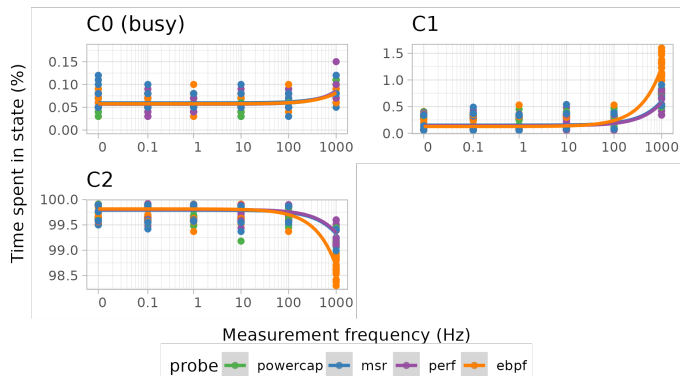


Figure 11. Relationship between acquisition frequency and idle CPU states (AMD server)

latency, but it also has the lowest variance, which means that its performance is more stable than the other mechanisms. This better performance may explain the surprising result of section VI-B1 on the AMD server. We see that powercap and MSR are faster on the laptop than on the servers. This could be due to the more recent kernel version of the laptop, i.e. these interfaces could have been optimized after Linux 4.18.

As indicated by "N/A"s in the last row, we have chosen to exclude eBPF from this microbenchmark because of its non-representative results. In the novel eBPF-based mechanism presented in section V-E, the reading of the counter, as most of the work, happens in the kernel, and thus cannot be measured by the Criterion benchmark tool.

D. Synthesis and recommendations

Our experiments reveal that reading the RAPL energy consumption counters does not decrease the throughput of parallel

mechanism	time to read one RAPL counter (95% confidence interval, microseconds)		
	recent laptop	AMD server	Intel server
powercap	[1.966, 2.106]	[6.083, 6.117]	[4.230, 4.463]
perf-events	[0.5371, 0.5385]	[0.4990, 0.4993]	[0.7376, 0.7377]
msr	[0.5391, 0.5688]	[4.437, 4.448]	[2.149, 2.348]
ebpf	N/A	N/A	N/A

Table III
TABLE OF THE COST OF POLLING ONE MEASUREMENT FROM RAPL

software in almost all tested configurations. All four tested mechanisms have a small or negligible impact on the running time of the benchmarks, even when the acquisition frequency is high. The differences reported by the microbenchmarks are less visible in the long benchmark. Since the benchmarks used all the cores, we conclude that the cost of the context switches caused by the RAPL measurements is very small.

Compared to existing software tools, our work is significantly more lightweight. We have measured that the simplified versions of Scaphandre and CodeCarbon, described in section IV-C, have a non-negligible overhead of at least 3% and 0.5% respectively, at only 10 Hz. According to prior work [29], the full versions exhibit an even higher overhead (between 2 to 7% at 1 Hz). Minimizing the overhead is good because the latter affects the results of the measurement.

We found that measuring at high frequencies had a significant impact on the idle consumption. In particular, it reduces the time spent by the processor in low-power states, which increases the energy consumption, especially on the tested Intel processor. We hence recommend adapting the acquisition frequency to the state of the node. Under heavy load, a high frequency can be used in order to capture more information about the running processes. On the opposite, when it is lightly loaded a lower frequency should be used. In addition, the perf-events and eBPF mechanisms seem to be the most energy-efficient when the processor is idle.

While eBPF can be used to obtain the energy consumption reported by RAPL, it has no advantage over the other mechanisms. In light of its complexity, analyzed in section V-E, we recommend not to use eBPF for this task. We make the same recommendation about MSR: unless there is no choice, that is if the OS does not provide any interface on top of the registers (like Windows), the user will prefer a higher-level mechanism.

A counter-intuitive result of our experiments is that MSR is actually slower than perf-events for energy measurement. Analyzing the implementation of the MSR module may provide an explanation. As detailed in section V-B, reading the counters is achieved by reading the file `/dev/cpu/N/msr`, where N is the id of the CPU core. The module relies on the file's metadata to determine on which CPU core the `rdmsr` x86 instruction must be executed, and fetches some information related to the inode on every read. On the contrary, perf-events keeps track of the relevant information in a simple structure initialized on `perf_event_open`, not on every read.

Perf-events and powercap have similar qualities, yet impor-

tant differences. On one hand, perf-events is almost immune to overflows, consumes less on idle and has a lower latency. On the other hand, powercap uses the more friendly sysfs API, which allows to get the measurements by simply reading a text file. If possible, perf-events should be preferred for its efficiency, but can be harder to use in some contexts like Bash scripts. The choice is therefore in the hands of the developer.

VII. CONCLUSION

In this work, we considered the different software mechanisms that allow to access RAPL energy counters, including a new eBPF-based mechanism that we designed. We provided a precise understanding of their operations and compared them on the basis of qualitative criteria, namely, technical difficulty, required knowledge, provided safeguards, necessary privileges and resiliency. To highlight their differences and help the developers of software measurements tools, we released a new minimal tool with a reference implementation of each studied mechanism. We explained how difficulties such as overflows, inaccurate timing and I/O jitter could be overcome. We assessed that our implementation was able to sustain high measurement frequencies with a low CPU overhead and very low standard deviation, hence improving over the two existing tools that we tested.

Using this minimal tool, we conducted an experimental study on two processor models. Our results showed that, despite their differences, the mechanisms had a similar performance and energy overhead when the machine was loaded. Under nearly all benchmark configurations, running the tool had a negligible impact on the running time and power consumption, even with a high acquisition frequency of 1000 Hz. This indicates that our implementation is more lightweight and more efficient than existing software tools. It would therefore be interesting to add more features to the minimal tool, such as the estimation of the consumption of each individual process, while keeping its overhead small. The aim would then be to extend the minimal tool in order to make it better suited for end users.

We found notable differences of overhead between the mechanisms on an idle server and in microbenchmarks. In light of the experimental results and qualitative comparison, we were able to provide recommendations on the choice of the most adequate mechanism. Our main recommendation is that, quite unexpectedly, one does not need to use the most complex mechanisms (MSR, eBPF) in order to be efficient. Prefer to use the perf-events interface whenever it is possible.

Dynamically adapting the acquisition frequency to the load is also recommended. It could be particularly useful to limit the energy consumption of edge devices while collecting precise measurements. This idea could be explored in a future work, with more diverse benchmarks.

This study did not consider GPUs nor TPUs. It looks like there are fewer choices to make since GPU vendors provide a relatively high-level software library (NVML for NVidia, ROCm for AMD) and impose an underlying mechanism. Nevertheless, a natural research extension is to conduct a similar comparative and experimental study on these specialized hardware.

REFERENCES

- [1] Jens Malmmodin and Dag Lundén. “The Energy and Carbon Footprint of the Global ICT and E&M Sectors 2010–2015”. In: *Sustainability* 10.9 (2018). ISSN: 2071-1050. DOI: 10.3390/su10093027. URL.
- [2] Anders Andrae. “Prediction Studies of Electricity Use of Global Computing in 2030”. In: 8 (Mar. 2019), pp. 27–33.
- [3] Lotfi Belkhir and Ahmed Elmeligi. “Assessing ICT global emissions footprint: Trends to 2040 & recommendations”. In: *Journal of Cleaner Production* 177 (2018), pp. 448–463. ISSN: 0959-6526. DOI: <https://doi.org/10.1016/j.jclepro.2017.12.239>. URL.
- [4] David Bol, Thibault Pirson, and Rémi Dekimpe. “Moore’s Law and ICT Innovation in the Anthropocene”. In: *2021 Design, Automation & Test in Europe Conference & Exhibition*. 2021, pp. 19–24. DOI: 10.23919/DAT51398.2021.9474110.
- [5] The Shift Project. *Environmental impacts of digital technology : 5-year trends and 5G governance*. Apr. 2023. URL.
- [6] Charlotte Freitag et al. “The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations”. In: *Patterns* 2.9 (2021), p. 100340. ISSN: 2666-3899. DOI: <https://doi.org/10.1016/j.patter.2021.100340>. URL.
- [7] Lorenz M. Hilty et al. “The relevance of information and communication technologies for environmental sustainability – A prospective simulation study”. In: *Environmental Modelling & Software* 21.11 (2006). Environmental Informatics, pp. 1618–1629. ISSN: 1364-8152. DOI: <https://doi.org/10.1016/j.envsoft.2006.05.007>. URL.
- [8] Lorenz M. Hilty and Bernard Aebischer. “ICT for Sustainability: An Emerging Research Field”. In: *ICT Innovations for Sustainability*. Ed. by Lorenz M. Hilty and Bernard Aebischer. Cham: Springer International Publishing, 2015, pp. 3–36. ISBN: 978-3-319-09228-7.
- [9] James DeLoss. “AI helping to unravel complexity of climate, weather and land use, find solutions to climate change”. In: (2023). URL.
- [10] Aina Rasoldier et al. “How realistic are claims about the benefits of using digital technologies for GHG emissions mitigation?” In: *Eighth Workshop on Computing within Limits 2022*. LIMITS. June 2022. URL.
- [11] Vlad C. Coroamă et al. “A Methodology for Assessing the Environmental Effects Induced by ICT Services: Part I: Single Services”. In: *Proceedings of the 7th International Conference on ICT for Sustainability*. ICT4S2020. Bristol, United Kingdom: Association for Computing Machinery, 2020, pp. 36–45. ISBN: 9781450375955. DOI: 10.1145/3401335.3401716. URL.
- [12] IPCC. *Climate Change 2022: Mitigation of Climate Change. Contribution of Working Group III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. Ed. by P.R. Shukla et al. Cambridge, UK and New York, NY, USA: Cambridge University Press, 2022. DOI: 10.1017/9781009157926. URL.
- [13] Will Steffen et al. “Planetary boundaries: Guiding human development on a changing planet”. In: *Science* 347.6223 (2015), p. 1259855. DOI: 10.1126/science.1259855. eprint: <https://www.science.org/doi/pdf/10.1126/science.1259855>. URL.
- [14] Awais Khan et al. “An Analysis of System Balance and Architectural Trends Based on Top500 Supercomputers”. In: *The International Conference on High Performance Computing in Asia-Pacific Region*. HPC Asia 2021. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 11–22. ISBN: 9781450388429. DOI: 10.1145/3432261.3432263. URL.
- [15] *Top500 list*. URL.
- [16] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer’s Manuals, Volume 3B*. 2023.
- [17] Franz Christian Heinrich et al. “Predicting the energy-consumption of MPI applications at scale using only a single node”. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. Sept. 2017, pp. 92–102. DOI: 10.1109/CLUSTER.2017.66.
- [18] Guillaume Fieni, Romain Rouvoy, and Lionel Seinturier. “SmartWatts: self-calibrating software-defined power meter for containers”. en. In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. Melbourne, Australia: IEEE, May 2020, pp. 479–488. ISBN: 978-1-72816-095-5. DOI: 10.1109/CCGrid49817.2020.00-45.
- [19] Victor Schmidt et al. “CodeCarbon: estimate and track carbon emissions from machine learning computing”. In: *Zenodo* (2021).
- [20] Aurélien Bourdon et al. “Powerapi: A software library to monitor the energy consumed at the process-level”. In: *ERCIM News* 2013.92 (2013).
- [21] Benoit Petit. *Scaphandre*. 2020. URL.
- [22] Daniel Hackenberg et al. “Power measurement techniques on standard compute nodes: a quantitative comparison”. en. In: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Austin, TX, USA: IEEE, Apr. 2013, pp. 194–204. ISBN: 978-1-4673-5779-1 978-1-4673-5776-0 978-1-4673-5778-4. DOI: 10.1109/ISPASS.2013.6557170.
- [23] Daniel Hackenberg et al. “An energy efficiency feature survey of the Intel Haswell processor”. en. In: *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. Hyderabad, India: IEEE, May 2015, pp. 896–904. ISBN: 978-1-4673-7684-6. DOI: 10.1109/IPDPSW.2015.70.
- [24] Song Huang et al. “Measurement and Characterization of Haswell Power and Energy Consumption”. In: *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*. E2SC ’15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450339940. DOI: 10.1145/2834800.2834807. URL.
- [25] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. “A validation of DRAM RAPL power measure-

- ments”. en. In: *Proceedings of the Second International Symposium on Memory Systems*. Alexandria VA USA: ACM, Oct. 2016, pp. 455–470. ISBN: 978-1-4503-4305-3. DOI: 10.1145/2989081.2989088.
- [26] Kashif Nizam Khan et al. “RAPL in Action: Experiences in Using RAPL for Power Measurements”. In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3.2 (Mar. 2018). ISSN: 2376-3639. DOI: 10.1145/3177754. URL.
- [27] Eva García-Martín et al. “Estimation of energy consumption in machine learning”. en. In: *Journal of Parallel and Distributed Computing* 134 (Dec. 2019), pp. 75–88. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2019.07.007.
- [28] Robert Schöne et al. “Energy efficiency aspects of the AMD Zen 2 architecture”. en. In: *2021 IEEE International Conference on Cluster Computing (CLUSTER)* (Sept. 2021), pp. 562–571. DOI: 10.1109/Cluster48925.2021.00087.
- [29] Mathilde Jay et al. “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. URL.
- [30] Lucia Bouza Heguerte, Aurélie Bugeau, and Loïc Lanelongue. “How to estimate carbon footprint when training deep learning models? A guide and review”. In: *Environmental Research Communications* (2023). DOI: 10.1088/2515-7620/acf81b. URL.
- [31] Srinivas Pandruvada. 2016. URL (visited on 09/01/2023).
- [32] Guillaume Raffin. *Detect and correct overflows of the RAPL microjoule counter*. 2023. URL (visited on 09/01/2023).
- [33] Martin Pollard. 2022. URL (visited on 09/01/2023).
- [34] Thomas Gruber. *Review overflow handling for RAPL counters*. 2015. URL (visited on 09/01/2023).
- [35] Marcus Hähnel et al. “Measuring energy consumption for short code paths using RAPL”. In: *ACM SIGMETRICS Performance Evaluation Review* 40.3 (Jan. 2012), pp. 13–17. ISSN: 0163-5999. DOI: 10.1145/2425248.2425252.
- [36] Inria SPIRALS. *PyJoules*. 2019. URL.
- [37] J. Treibig, G. Hager, and G. Wellein. “LIKWID: a lightweight performance-oriented tool suite for x86 multicore environments”. In: *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*. San Diego CA, 2010.
- [38] Stephane Eranian. *perf/x86/rapl: fix AMD event handling (commit 0036fb0)*. 2022. URL.
- [39] *The Linux Kernel - CPU hotplug in the Kernel*. URL.
- [40] *MacOS General Commands Manual - powermetrics*. URL.
- [41] Zhenkai Zhang et al. “Red Alert for Power Leakage: Exploiting Intel RAPL-Induced Side Channels”. In: May 2021, pp. 162–175. DOI: 10.1145/3433210.3437517.
- [42] *msr_safe: kernel module implementing access-control lists for MSR*. URL (visited on 05/16/2024).
- [43] *The Linux Kernel - Powercap Documentation*. URL.
- [44] *The Linux Kernel - perf wiki*. URL.
- [45] *Linux manual - perf_event_open*. URL.
- [46] *What is eBPF? An introduction and deep dive into the eBPF technology*. URL (visited on 09/21/2023).
- [47] *BPF Features by Linux Kernel Version*. URL.
- [48] Dominik Scholz et al. “Performance Implications of Packet Filtering with Linux eBPF”. In: *2018 30th International Teletraffic Congress (ITC 30)*. Vol. 01. 2018, pp. 209–217. DOI: 10.1109/ITC30.2018.00039.
- [49] David Bailey et al. *The NAS parallel benchmarks 2.0*. Tech. rep. Technical Report NAS-95-020, NASA Ames Research Center, 1995.
- [50] *A statistics-driven micro-benchmarking library written in Rust*. URL (visited on 09/01/2023).



Guillaume Raffin is a PhD student at the institute of technology of Univ. Grenoble-Alpes and at the R&D department of Bull SAS, which is part of Eviden (Atos group). He is working on distributed multi-site scheduling and measurement techniques, with a focus on environmental impact and industrial applications in both traditional HPC and cloud computing.



Denis Trystram is a distinguished professor at the institute of technology of Univ. Grenoble-Alpes. He is an honorary member of the Institut Universitaire de France. He was working for a long time on resource management in parallel and distributed platforms (including HPC clusters, clouds, Internet of Things) and multi-objective optimization with a special focus on minimizing the energy consumption. Since 2019, he is leading a research program at the multidisciplinary AI institute in Grenoble on edge intelligence and frugal AI. His academic record

is composed of more than 100 publications in international peer reviewed journals and more than 150 conferences. More details can be found at: <http://datamove.imag.fr/denis.trystram/index.php>