A Protocol to Assess the Accuracy of Process-Level Power Models

Emile Cadorel, Dimitri Saingre

A Protocol to Assess the Accuracy of Process-Level Power Models

1st Emile Cadorel
Davison consulting
Rennes, France
emile.cadorel@davidson.fr

2nd Dimitri Saingre
Davison consulting
Rennes, France
dimitri.saingre@davidson.fr

***Abstract*—The energy consumption of servers has been a critical research area, drawing significant interest from academic and industrial sectors. Various models have been developed to estimate power consumption of computing devices, ranging from simple linear models to more complex ones. In recent years, as the cloud paradigm has expanded and allowed applications to be hosted alongside others in virtual machines, power models have evolved to distribute energy usage among applications on a server. These models aim to achieve two primary objectives: monitoring the energy footprint and optimizing energy consumption. Nonetheless, little attention has been given in academic literature to evaluate the efficiency and accuracy of the allocation phase of this models. This paper presents a definition of power division and a protocol to evaluate models using such division. The proposed protocol is used to evaluate models on physical machines with different performance settings, toggling hyperthreading and turboboost. Before discussing the conceptual distinctions between power and energy allocation models, our results show the existence of some limitations in the existing models. These results provide valuable insight into the missing information needed to improve the accuracy of power models.**

***Index Terms*—energy consumption, power usage, modeling, allocation, virtual machines, performances**

I. Introduction

The energy consumption of servers is a research topic on its own that has attracted increasing interest in both the academic and industrial worlds in recent years. Many models have been developed to estimate the energy consumption of computing devices, ranging from simple linear models to more complex models. Historically such models were designed to represent the consumption of the entire computing device, (e.g. a server in an HPC cluster [1], [2], or a battery-powered mobile device [3]). In more recent years, due to the increasing use of the Cloud paradigm, applications are no longer hosted alone on a server, but are collocated with other applications, usually running inside virtual machines. To account for this new reality, power models have been adapted to divide the estimated consumption among the different applications that are running on the server [4]–[6]. In this paper, we will refer to this type of model as a power division model. Such models use sensors to acquire the energy consumption of the targeted server (e.g. RAPL), and use system metrics (e.g. performance counters) to divide this consumption among applications.

The objective generally put forward by the authors of power division model is to assign an energy consumption to the various software programs running on a given server, and to use this information to determine which software program is the most energy-consuming among the others. This objective can be analogized to the creation of *Life Cycle Assessment* (LCA) [7], [8] for the different piece of software running on a given infrastructure. Some power division models extend the system of division of energy among software applications, to division among virtual machines (VMs). In that context, VMs may belong to different users, and context of deployment (host machine, CPU architecture and size, neighbour VMs…) is invisible within the virtual machines. This energy consumption may be divided a second time among the applications running inside the VM. Two different actors that might be interested by power division models can be identified. a) the Cloud provider, to divide the consumption among the different VMs running on their infrastructure, and to allocate these consumptions to their clients, b) the end users, to assign the energy consumptions of their VMs to the different applications that are being executed inside their VMs.

On the other hand, it is claimed [4] that power division models can be used to optimize the energy consumption of running applications, and can help developers to understand their behaviors, in a context where multiple applications are running concurrently. From this point of view, a leverage is generally brought forward [7], [8]; modifying a preexisting source code to optimize its performance in terms of its energy consumption. It is worth noting that power division models are intended for use in a shared context where multiple applications run concurrently (this is the main reason for division) as well as in a production context. Unlike in HPC environments, where hosts are configured and optimized to accommodate specific applications, production platforms are set up to accommodate a variety of workloads, and end-users have no control over these settings. The emergence of models like Kepler [9], which target deployments in Kubernetes, demonstrate a willingness to utilize these models within a Cloud context. From our perspective, Cloud represents the primary use case for these models, and this paper will explore this context.

However, the literature has given little attention to validating the allocation algorithm utilized by power division models, claiming that there is no truth value to compare to, and have limited their investigation to the ability of the

models to recover the global consumption of the machine. This paper provides a clear definition of power division and a standardized protocol for evaluating models utilizing such division. The proposed protocol is used to evaluate models on physical machines with different performance settings, toggling hyperthreading and turboboost. Our results reveal certain limitations in the existing power division models, which should be addressed to enhance their accuracy.

The paper is structured as follows. Firstly, Section II introduces the topic of power models and highlights the limitations of the validation protocols from the literature. Section III proposes a formal definition of the problem of power division, and use it to create a protocol to evaluate power division models. Sections IV presents the experimental results obtained using the proposed protocol, and highlights limitations of the state of the art models, and identifies areas requiring improvement to enhance their efficiency. Section V outlines the conceptual differences between power and energy division models. Finally, Section VI concludes the paper with suggestions for future research work.

II. Related work

This section summarizes existing methods and interesting related work on models for measuring and estimating energy and power consumption.

There are various methods available for measuring the power consumption of a computing node (or server). One option is to use an external device, known as a power meter, which can be connected to the server's power supply via a wall socket, as demonstrated in previous studies [10]–[12]. Although this approach has the benefit of providing highly accurate measurements, it may prove relatively costly as a power meter would need to be acquired for each individual computing server. Furthermore, it is important to note that such a device is unable to isolate consumption of particular components (e.g. CPU, RAM, disk). An alternative method frequently utilized involves the implementation of digital sensors integrated directly within the components by the manufacturer. The RAPL (Running Average Power Limit) sensor, initially introduced by Intel in [13], has been widely adopted since [14]–[16]. This instrument measures power consumption at different component levels, encompassing the entire CPU socket, all CPU cores, integrated GPU, and RAM. According to the authors of [14], RAPL gives really accurate results with very little overhead, as the sensor is based on integrated voltage regulators since Intel Haswell architecture. AMD processors are equipped with RAPL sensors, which operate on a software model, unlike Intel processors. This can lead to inconsistencies, as demonstrated by the authors of [17]. Other components, such as Nvidia graphics cards, can be measured with good accuracy using the NVML (Nvidia Management Library) API [18]. In this paper, we will concentrate solely on the power usage of the CPU, as this is the component that is generally prioritized by power division models.

Prior to the introduction of RAPL, software models were used to model the power consumption of physical machines. These models still have a purpose in estimating the power consumption of components that are not covered by RAPL, or other embedded sensors (e.g. hard disks, network cards, etc.). In their work [19], the authors present a taxonomy of power models based on software monitoring. Software monitoring is typically performed by gathering system metrics, such as performance counters, that offer insights into the activities and utilization of the physical machine, such as CPU usage, instruction count, cycle count, I/O accesses, etc. In their study [20]–[22], the authors propose linear power models based on CPU and memory usage to estimate VM consumption. As outlined in [19], creating a universal power model is a challenging task because CPU usage is not fully correlated with power consumption, and it would need to be adapted for each type of application, let alone applications with changing behaviour. The authors of [2] use an application classification to determine a subset of performance counters that can be used to represent the power consumption of a given application. Although this classification may be appropriate in an HPC context where only one application is used at a time on a given machine, it does not take into account the variability of the application (change in behaviour during execution) and does not take into account a more cloud-oriented context where multiple applications with different behaviours may be running on the same hardware at the same time. Similar research was carried out by the authors of [5], who used performance counters to create a power model with adjustable weights to estimate the energy consumption of applications.

Based on prior research in [2], [5], a tool called *PowerAPI* [4] has been introduced. This tool uses the performance counters of the applications running on a given machine to estimate the energy consumption of the whole machine, and also divides this estimated consumption among the applications running on it. A linear regression using RAPL as the objective value is carried out to perform this estimation and division. *Scaphandre* [23] and Kepler [9] are both tools that aim to distribute physical machine power consumption among running applications. *Scaphandre* does this by using values obtained from RAPL and CPU usage data, while Kepler focuses on Kubernetes containers and uses performance counters collected with eBPF and energy measurements collected with RAPL. Close approaches are used to estimate the energy consumption of machine learning training phases by [24]–[26], generally using NVML to capture GPU power consumption. While Kepler [9] is intended to be used in a cloud context with containers (Kubernetes), the authors of *PowerAPI* [4] and *Scaphandre* [23] claim that their models can be used in a cloud context to divide the consumption among the VMs hosted on a given physical machine.

We consider that there is a lack of research evaluating the accuracy and effectiveness of the proposed methods for dividing or assigning physical machine power consumption to running applications. The studies conducted by [4] only evaluate the ability of the proposed model to recover the global power consumption of RAPL. A separate evaluation of state-of-the-art models is presented in [6], focusing on *Power-*

API and *Scaphandre*. The authors report that various models from the literature [4], [23] produce different allocations, but they point out the lack of research conducted to establish the most accurate model. To the best of our knowledge, these are the only two evaluations of power division models in the literature, and neither assesses the accuracy of the division itself. The lack of evaluation in this area is often attributed to the difficulty in providing reference values with which to compare the results of the model, which in our view is due to the lack of a proper definition of what is intended when power division models are used.

To address this unresolved issue, this paper provides a formal definition of the power division, which is then used to construct a truth value. Following that, this definition is applied to build an experimental protocol that evaluates the state of the art models, highlighting their limitations. It will be demonstrated in the remainder of this paper that this protocol provides useful information about the conceptual limitations of power division models and what is missing to improve them.

III. Definition and validation protocol for power division models

As outlined in the related work section, several papers introducing a new power model assess their findings at the server level, either against a software solution, such as RAPL, or a physical wattmeter. As a result, it is often assumed that the division of energy consumption at the software level is valid, despite a lack of validation protocols. As previously discussed, this lack of evaluation is frequently justified by the absence of a baseline against which to compare. From our perspective, the lack of objective value in this area stems from the absence of a formal definition for what is intended and expected from power division models. In this section, we aim to outline a clear definition of the division phase of such models. This section will solely focus on power division; however, Section V will cover the conceptual distinctions between power and energy modelization, and present open research perspectives on energy division.

A) Notations - To fully comprehend our proposed definition of power division models, it is necessary to introduce some notations and some context on power consumption of physical machines. Let $\mathbb{P}$ represent a set of applications that can be executed on a physical machine M, with each P_i denoting the i-th application. A scenario $S \subset \mathbb{P}$ comprises of n applications from $\mathbb{P}$. There are two types of scenarios. Firstly, a parallel scenario, denoted as $P_0 \parallel P_1$, where two applications P_0 and P_1 run concurrently on the machine. On the contrary, a sequential scenario is denoted as just P_i, where only one applications namely P_i is executed. The scenario S/P_i denotes the scenario S without the application P_i, with $P_i \in S$. Let $T_S^{P_i}$ represent the time taken by application P_i in scenario S, measured in seconds, and T_S the time taken by the execution of the whole scenario S. $C_{S,t}$ denote the power consumption of machine M executing scenario S at instant $t \leq T_s$. Let $Ce_{S,t}^{P_i}$ represent the estimated power consumption of the application P_i at a given moment t when executed in scenario S. For instance, consider two applications P_0 and P_1 executed in the scenario $P_0 \parallel P_1$, $Ce_{P_0 \parallel P_1,t}^{P_0}$ is the estimated power consumption of the application P_0 at instant t. Please refer to Table I for a summary of these notations.

Without any knowledge of the specifics of a physical machine's power consumption, one can consider the following definition for the allocation of consumption among processes. *The estimated consumption of a process is the extra consumption that would not be observed if the process was not running on the machine.* For a specific scenario S, this definition can be condensed into the Equation 1, assuming that T_S and T_{S/P_i} are equal (see Section V). This definition is incomplete and incompatible with reality, as will be demonstrated in the upcoming section.

$$\begin{aligned} &\forall P_i \in S, \forall t \in T_S, \\ &Ce_{S,t}^{P_i} = C_{S,t} - C_{S/P_i,t} \end{aligned} \tag{1}$$

Symbol	Explanation
$P_i \in \mathbb{P}$	Application i
$P_i \parallel P_j$	a scenario where P_i and P_j are executed in parallel
S/P_i	a scenario of multiple applications, where P_i is removed
$C_{S,t}$	The real power consumption of the scenario S at instant t (e.g. RAPL)
R	The residual consumption of the machine
$A_{S,t}$	The active consumption of the scenario S at instant t
$Ce_{S,t}^{P_i}$	The estimated consumption of P_i in the scenario S at instant t
$Ae_{S,t}^{P_i}$	The estimated active consumption of P_i in the scenario S at instant t
T_S	The time taken to execute the scenario S
$T_S^{P_i}$	The time taken by the application P_i in the scenario S

TABLE I: Notations

Power division models are applicable on machines with different settings, including toggling hyperthreading and turboboost. The power consumption of a physical machine is significantly affected by the presence or absence of Hyperthreading and turboboost. In this paper, we use their presence and absence as two contexts for both laboratory simplified context and production context. The context we are calling laboratory is the context we used to stabilize the power consumption of the machines and to have a simplified environment where comparisons of models are easier.

B) CPU power consumption without hyperthreading and turboboost - Power division models commonly divide CPU power consumption. This paper analyses this use case. The power consumption of a CPU depends on the workload being executed on it, which mainly depends on the number of cores currently in use. To obtain the curve of the CPU consumption, various functions from the stress-ng suite [1] were executed with the number of processes varying between 0 and the number of physical cores present on the machine. Functions from the stress-ng suite were selected due to their highly stable resource and power usage (generally less than 0.5 watt of variance in their load). Two different machines were employed during the experiments, and their specifications are outlined in Table II. DAHUis a machine which is accessible in the Grid'5000 testbed[2] from the Cluster in Grenoble.

[1]https://github.com/ColinIanKing/stress-ng

[2]https://www.grid5000.fr

Name	Description
SMALL INTEL	6 cores (12 logicals) Intel(R) Xeon(R) W-2133 CPU, 32GB RAM, 256GB SSD
DAHU	2 × 16 cores (64 logicals) Intel Xeon Gold 6130 CPUs, 192GB RAM, 240GB SSD

TABLE II: List of machines used in the experiments

In this initial power measurement, both hyperthreading and turboboost were disabled. Figure 1 plots the maximum and minimum power consumption recorded on the machines during various stress tests as a function of CPU load (i.e. the number of cores used compared to the total number of physical cores available), where 0% indicates that the machine is idle.

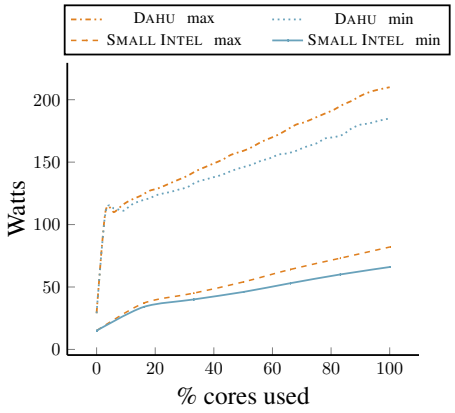


Fig. 1: Minimal and maximal power usage of the machines operating without Hyperthreading and turboboost capabilities

Upon analysis of the results, it becomes apparent that there is a certain degree of variation in power consumption when exposed to the same CPU load. On DAHU, we recorded a variation of 25 watts, or more than 10% of its maximum power consumption. This variance is due to the diverse range of stress applications that were executed on the nodes, producing different types of instructions with varying costs [17], [27]. The power consumption, ranging from 1 active core (3% usage on DAHU and 16% on SMALL INTEL) to 100% usage, appears almost linear to the CPU load. However, there is a large gap between the idle power consumption and the power consumption when 1 core is used. Indeed, on SMALL INTEL, the linear factor between 16% and 100% results in approximately 7 watts for the highest recorded consumption. However, there is a gap of about 22 watts between 0% and 16%. On DAHU, the gap is considerably larger at 81 watts, for a maximal linear factor of 2.8 watts. In the upcoming sections of this article, the term *residual consumption* will be utilized to refer to this consumption and will be denoted by R. It is important to note that residual consumption differs from idle power consumption as it only occurs when the CPU is under a load.

When trying to make an attribution, between two applications P_0 and P_1, using the definition presented in Equation 1, one can note a first problem. The sum of the estimations $Ce_{S,0}^{P_0} + Ce_{S,0}^{P_1}$ for $S = P_0 \parallel P_1$ is lower than the real power consumption $C_{S,0}$ as residual consumption is simply ignored (illustrated in Figure 2). Let's denote $A_{S,t}$ the active power consumption of a scenario at instant t, such that $A_{S,t} = C_{S,t} - R$, and $Ae_{S,t}^{P_i}$ the estimated active consumption of the application P_i in the scenario S. In our case the residual consumption includes the idle consumption. It can be argued that residual consumption is generated by both programs, as it appears only because the CPU is under load.

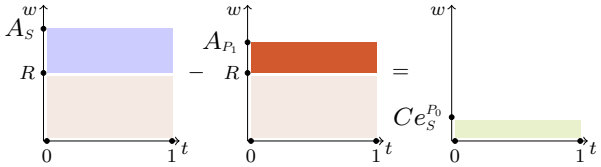


Fig. 2: Estimation of P_0 according to Equation 1

From a practical point of view, it makes sense to disregard this residual consumption R and say that, like idle consumption, it is due to all applications running on the machine and therefore cannot be allocated to any specific application. This residual consumption could be allocated according to many possible policies. For example, let's consider the three following families: **(F1)** divide the residual consumption according to the ratio of the respective active consumption of each running applications, **(F2)** divide the residual consumption so that the estimated consumption of two applications running in sequential and parallel contexts maintain the same ratio (i.e. $\frac{C_{P_0}}{C_{P_1}} = \frac{Ce_{P_0 \parallel P_1}^{P_0}}{Ce_{P_0 \parallel P_1}^{P_1}}$), **(F3)** disregard it, so that the estimated power consumption of an application does not depend on whether other applications are present. It can be reasonably argumented that all three are valid and provide different properties. This results in models that cannot be evaluated based on physical reality, but rather on allocation policies.

The state-of-the-art power division models, *Scaphandre*, *Kepler* and *PowerAPI*, apply the first policy. In fact they divide the total machine consumption (i.e. C_S) among applications without considering the presence of residual consumption, producing the same result as using the same ratio for active and residual consumption. Regardless of the decision made regarding residual consumption, the active consumption of an application can serve as a comparable value to verify the precision of power models, as specified in Equation 2, where x is to be defined by the family of power models in use. This equation will be extended in the next subsection for production context.

$$\forall P_i \in S, \forall t \in T_S, \quad Ce_{S,t}^{P_i} = A_{S,t} - A_{S/P_i,t} + xR \tag{2}$$

The frequency significantly affects residual consumption. For example, when limiting the maximum frequency of the CPU cores of SMALL INTEL to 2Ghz, the residual consumption drops from about 28 watts to 17 watts, and to 15 watts when setting the maximum frequency to the nominal

one (1.2Ghz). Residual consumption is not cumulative and is correlated to the frequency of the fastest-running core across all cores. Based on the given information, it can be argued that the increase in residual consumption should be attributed to the applications that caused one of the cores to increase CPU frequency. In our opinion, the second family, which maintains the same ratio in parallel execution as in isolated execution, has the advantage of taking this reality into account, since it considers the residual consumption as part of the application consumption. Thus, it will be presented as an alternative solution in the next section.

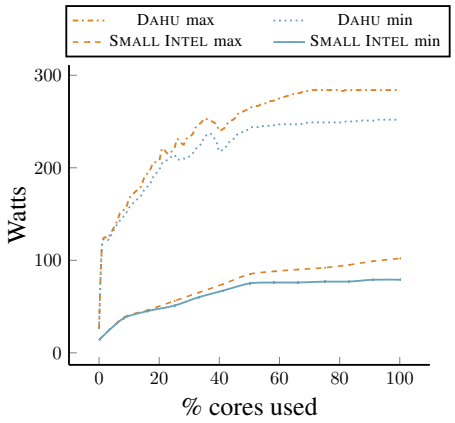


Fig. 3: Minimal and maximal power usage of the machines operating with Hyperthreading and turboboost capabilities

C) CPU power consumption with hyperthreading and turboboost - In the previous section, we observed a simplified setting where hyperthreading and turboboost were disabled. In a cloud computing context (we named production context), which is a typical setting for employing a power division model as presented in the introduction, this setting is unlikely to be used. Figure 3 plots the minimum and maximum power consumption recorded on the machine using the same protocol as in the last section, but with hyperthreading and turboboost enabled. The residual consumption is still observable, as expected, but the power consumption no longer follows a linear curve, but rather a logarithmic one. Due to this, the active power usage of a scenario is no longer equivalent to the sum of the active power usage of the applications when each is run individually. Instead, it becomes less than or equal to the sum, i.e. $A_S \leq \sum_{P_i \in S} A_{P_i}$. All applications in the scenario bear responsibility for the execution context and ensuing decrease in active consumption compared to sequential executions. Once again, policy statements could be made to define a share allocation of consumption. However, it can be argued that a model must remain consistent in various contexts, regardless of whether hyperthreading and turboboost are enabled or disabled. To achieve this, active power should be allocated in a consistent manner. Consequently, the estimated active power consumption of applications should maintain the same ratio for parallel and sequential execution as it does when hyperthreading and turboboost are disabled. This is formalized by Equation 3.

$$\forall P_i \in S, \forall t \in T_S,$$
$$Ae^{P_i}_{S,t} = A_{S,t} \times \frac{A_{P_i,t}}{\sum_{P_j \in S} A_{P_j,t}} \quad (3)$$

It can be argued that alternative allocations may be preferable. However, due to the lack of consensus in the literature on power division definition in this specific context, we suggest our definition as a fair proposal for state-of-the-art model comparison.

D) To summarize, – Defining the notion of *energy consumption of a software* is not an easy task, and there is no apparent consensus in the state of the art. From Figure 3, three challenges regarding power division models can be extracted. The first challenge **(C1)** - which is generally raised in the literature - is to consider the accuracy of the power model to account for different types of application behavior. This can be considered a *technical* challenge, since the observed variation is due to variability in application behavior, which results in variation in power consumption. The second challenge **(C2)** is to take into account the context of execution to provide stable results for a given application, since its consumption depends mostly on the deployment context. This second challenge is often overlooked in the literature on power division models, and with such vagueness as to suggest that it is possible to use them for software optimization (comparison of two versions of the same application in production context) without clear evaluation. The next challenge **(C3)** - deciding how to allocate the residual consumption, deciding how to allocate the power efficiency gain at higher CPU usage and so on -, defines a list of policy decisions. These decisions needs to be arbitrated, as no single option stands out as inherently better. This could lead to the creation of families of models (**(F1)**, **(F2)**,...). These decisions must be correctly defined when presenting a power division model in order to be comparable and repeatable as required by the scientific method. The next Sections IV and V present evaluations of the ability of the power division models to meet these challenges.

E) Protocol - To evaluate the accuracy of a power division model, a protocol consisting of three main parts must be defined. Given a list of applications $P_i \in \mathbb{P}$ that execute a set of repeatable instructions with stable and predictable power usage, first compute the active consumption of each application $A_{P_i,0}$ by running them alone on the machine and removing the residual consumption R from the acquired power consumption. Second, generate a list of scenarios consisting of two applications and simultaneously execute them, while ensuring there is no contention on the machine, i.e., the parallel and sequential executions have the same execution time. Obtain the estimated consumption $Ce^{P_i}_{S,0}$ of the applications according to the models and compare them with the active consumption acquired during phase (1).

Because the state-of-the-art models (*Scaphandre*, *PowerAPI*) use the policy of applying the same ratio for residual and active consumption, it is easy to extract the estimated

active consumption when disabling hyperthreading and turbo boost, following Equation 4.

$$\forall P_i \in S, \forall t \in T_S,$$
$$Ae^{P_i}_{S,t} = Ce^{P_i}_{S,t} - R \times \frac{Ce^{P_i}_{S,t}}{C_{S,t}} \tag{4}$$

When both hyperthreading and turboboost are activated, the estimated active power usage of an application is not equivalent in parallel and sequential execution. Therefore, another ratio must be calculated as outlined in Equation 3. Since this equation is applicable even if hyperthreading or turboboost are not activated, the model's absolute error can be calculated for both contexts by the following Equation 5.

$$AE_S = \frac{\sum\limits_{\substack{P_i \in S \\ t \in T_S}} \left| \frac{Ce^{P_i}_{S,t}}{C_{S,t}} - \frac{A_{P_i,t}}{\sum\limits_{P_j \in S} A_{P_j,t}} \right|}{dim(S) \times T_S} \tag{5}$$

IV. Evaluation of the models on stress applications

In this section, a first evaluation of the models from the literature is presented. The applications in the set $\mathbb{P}$ are selected to stress the CPU cores of the selected machines. These applications are functions within the popular benchmarking suite *stress-ng*, which provides more than 300 functions to generate specific loads on various components of the system: CPU, memory, file system, etc. The selection of *stress-ng* stems from its ability to produce consistent benchmarks, whereby each function always produces exactly the same load. We selected a subset of 12 tests that target the CPU, as listed in Table III. This evaluation was conducted on the DAHU and the SMALL INTEL machines (see Table II). All scripts and traces are available on a git repository[3].

Name	Description
ACKERMANN, QUEENS, FIBONACCI	Implementations of well-known algorithms
FLOAT64, INT64, DECIMAL64, DOUBLE	Perform operations of a given type
INT64FLOAT, INT64DOUBLE	Convert a given type to another
MATRIXPROD	Computes matrix product
RAND	Generates random numbers
JMP	Performs conditional jumps

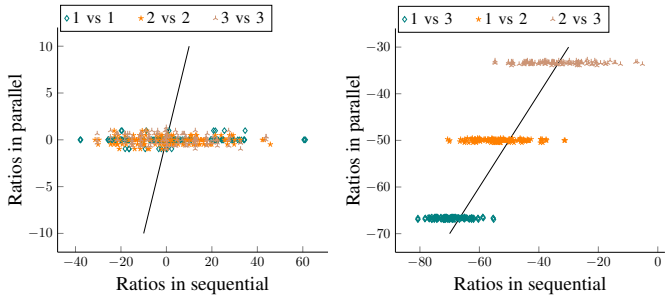
TABLE III: List of stress-ng tests

Multiple sets of applications were created with a varying number of threads. Note that the two largest applications (with the higher number of threads) can run on the machines without competing for CPU. Following the protocol outlined in Section III, we obtained values for all stress functions separately on each machine ($C_{P_i}, \forall P_i \in \mathbb{P}$) using RAPL. Then, we ran all possible combinations of the 12 functions with varying numbers of threads in parallel, collecting values for each combination ($Ce^{P_i}_{P_i \parallel P_j}, \forall P_i, P_j \in \mathbb{P}$).

[3]https://github.com/davidson-consulting/software-energy-model-evaluation

A) Evaluation in laboratory context - In this section, we evaluate the *PowerAPI* and *Scaphandre* models from the literature in a laboratory context. In this context, the turboboost and hyperthreading are disabled in order to stabilize the power consumption of the machines, as we have seen in Section III. *PowerAPI* and *Scaphandre* are the models we selected for evaluation and discarded *Kepler* as it targets a kubernetes environment, yet we wanted the simplest context with as little overhead and side effects as possible. However, because *Kepler* operates on a model that is relatively similar to the one utilized by *Scaphandre*, the conclusions presented in this section are applicable to it as well. Because hyperthreading was disabled, SMALL INTEL was capable of running 6 threads simultaneously without any contention while DAHU was able to handle 32. Consequently, the largest application on SMALL INTEL consisted of 3 threads, while on DAHU it consisted of 16 threads.

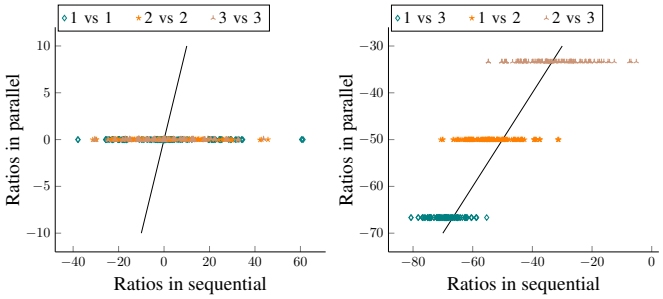
Figure 4 and 5 illustrate the power consumption ratio difference of two applications on the SMALL INTEL node for *Scaphandre* and *PowerAPI*, respectively. The ratio on the x-axis is calculated using the active power consumption of the two applications when executed independently on the machine $100 - \left(\frac{A_{P_i}}{A_{P_j}} \times 100\right)$. The ratio on the y-axis is calculated using the estimated consumptions, represented by $100 - \left(\frac{Ce^{P_i}_{P_i \parallel P_j}}{Ce^{P_j}_{P_i \parallel P_j}} \times 100\right)$. The ratios have been subtracted from 100 to center them around 0, so a result equal to 0 means two applications with the same power consumption, a negative result means P_1 consumes more than P_0, and a positive result means the opposite. Applications of different sizes were executed in parallel and are presented in Figures 4b and Figures 5b. The gray line represents the target value $y = x$.



(a) *Same size stress* (b) *Different size stress*

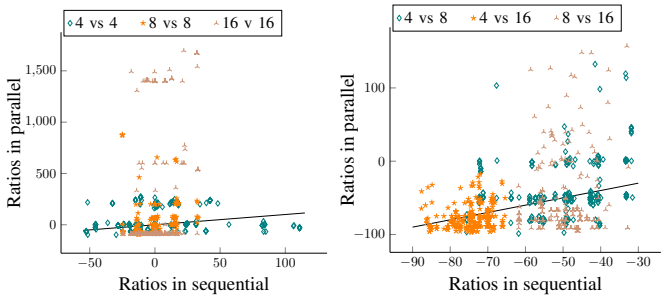
Fig. 4: Ratio for stress without HT according to *Scaphandre* on SMALL INTEL

During the evaluation, stress applications ran for 30 seconds, and the 10 seconds with the least extreme values were selected to ensure a variation of less than 2 watts at most in the estimations. It should be noted that slight variations may occur due to delays in the start of the stress execution, so that some points at the beginning and end of a scenario are not relevant to the behavior of the power division model. With *Scaphandre*, typically more than ten data points were usable. However,



(a) *Same size stress* (b) *Different size stress*

Fig. 5: Ratio for stress without HT according to *PowerAPI* on SMALL INTEL

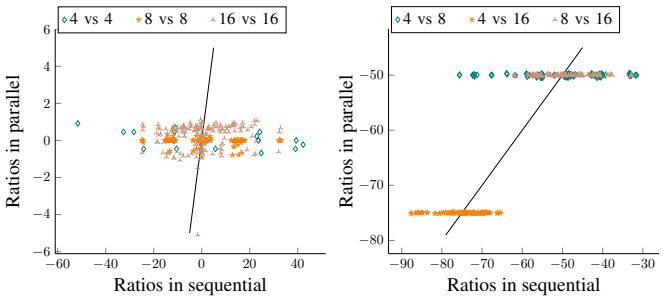


(a) *Same size stress* (b) *Different size stress*

Fig. 7: Ratio for stress without HT according to *PowerAPI* on DAHU

because of the nature of *PowerAPI*, which requires a learning phase, the first ten seconds of test execution are disregarded by the model, generating no estimations. It is worth noting that these estimation drops occur whenever there is a change in context. For our evaluation, we elected to remove these drops from consideration.

Both models seem to consider that applications with the same number of running threads consumes roughly the same amount of power, and omit the differences in terms of instruction and thus the differences in term of consumptions between different stress functions. In result, the maximum error that can be observed is 11.7% for both *PowerAPI* and *Scaphandre*, and occurs for the same pairs involving FIBONACCI (being the less consuming function) and one of the top consuming application (MATRIXPROD, INT64FLOAT, JMP). This maximum error ratio is almost the same as the width of the power consumption curve that was observed in Figure 1 and correspond to approximately 8 Watts on SMALL INTEL. However, because applications are relatively well scattered in term of power consumptions within the width of the power consumption curve of the machine, the overall error of the models are respectively 3.12% for *PowerAPI* and 3.15% for *Scaphandre* (cf. Equation 5).

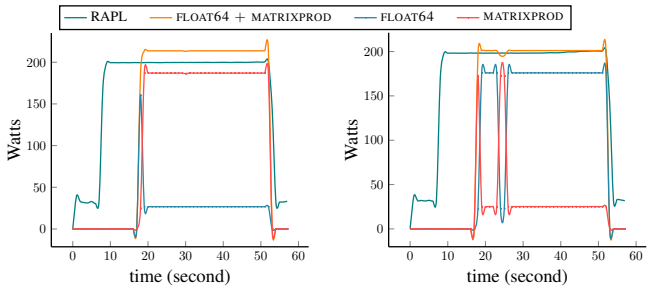


(a) *Same size stress* (b) *Different size stress*

Fig. 6: Ratio for stress without HT according to *Scaphandre* on DAHU

Figures 6 and 7 illustrate the power ratio difference of two applications when run individually and in parallel on the DAHU node for *Scaphandre* and *PowerAPI*, respectively. The behavior displayed by *Scaphandre* is similar to that observed on SMALL INTEL, with an average error rate of 2.7%. The maximum error of 17.4% can be observed between QUEENS and FLOAT64, which are near the minimum and maximum limits of the consumption curve of DAHU.

Alternatively, the *PowerAPI* produces varying results on DAHU compared to SMALL INTEL, with an average error rate of 16.23% and a maximum error rate of 49.1%. Such disparities could arise due to technical issues with the model, resulting in inconsistencies during runs. For instance, the model tends to define FLOAT64 as one of the least consuming applications, generally accounting for only about 10% of the machine's consumption, when in fact this application is one of the most consuming based on its isolated execution. Furthermore, the model occasionally changes its decisions, even if the applications' behavior remains stable. For example, we conducted two identical tests comparing FLOAT64 and MATRIXPROD. In the first test, *PowerAPI* assigned 90% of the consumption to FLOAT64, while in the second test, it assigned 90% of the consumption to MATRIXPROD. This results are presented in Figure 8. This behavior was not observed on SMALL INTEL, which suggests possible implementation issues (note that the latest stable official version 2.1.2 of *PowerAPI* was used).

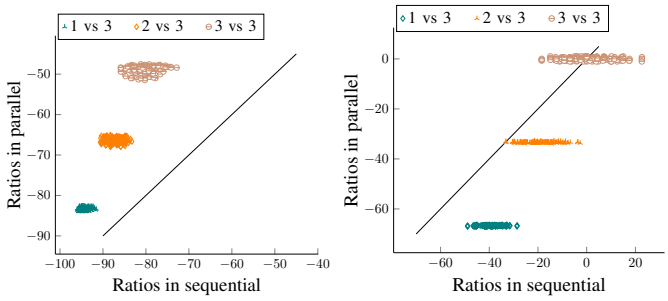


(a) *First run* (b) *Second run*

Fig. 8: Attribution between MATRIXPROD and FLOAT64 according to *PowerAPI* on DAHU

B) Considering Residual consumption - In these initial experiments, we did not include residual consumption as part of the application's power usage. Our decision was based on the fact that existing models use a linear approach that excludes this factor. However, as discussed in Section III,

residual consumption is not an idle consumption but is actually determined by the frequency of the cores and is generated by the applications themselves. In the following evaluation, we consider a workload that caused the residual consumption to vary. On node SMALL INTEL, we compared stress functions with CPU time capped at 50% (using cgroup) to stress functions without capping. To prevent context switching and ensure that some cores run for 50% of the time, we pinned the stress processes to specific CPU cores, with one process per core. In this scenario, the cores performing a stress function at a maximum of 50% usage operated at an average speed of 2.4 GHz. A residual consumption of 15 watts was produced when these stresses were executed independently. Meanwhile, the remaining cores were running at 3.6 GHz, resulting in 28 watts of residual power when the stresses were executed alone, the same residual consumption was observed when capped and uncapped applications where running in parallel. Since the machine's total power consumption when running capped applications was about 20 watts on average (for a residual of 15 watts), which is lower than its residual power consumption when running uncapped applications (28 watts, for a global consumption of about 74 watts on average), it can be concluded that the residual power consumption variability is indeed caused by application behavior. The figure 9a shows the results of the execution of capped applications against uncapped applications according to *Scaphandre*, where P_0 is capped to 50% of cpu time. The residual consumption of P_0 being lower than the residual consumption observed for P_1 when executed in isolation, the difference between the two residual consumption is allocated to P_1 in the objective value. In this context, the average error rate of *Scaphandre* is 13.7% with a maximum error rate of 21.64%.



(a) with P_0 capped at 50% (b) with P_0 uncapped

Fig. 9: Ratio for stress without HT according to *Scaphandre* considering part of R as application consumption

Because residual consumption is generated by applications, one should assign it to the consumption of the applications, even in isolation. However, this statement does not accurately reflect the actions of *Scaphandre* or *PowerAPI*. Figure 9b presents the error rate of the model for the execution presented in last subsection based on *Scaphandre*, while taking into account an expected ratio of $100 - \left(\frac{C_{P_i} - R_0}{C_{P_j} - R_0} \times 100\right)$, with R_0 being the residual consumption of the machine at nominal frequency. In this case, the maximum error rate is 19.3%, and the global error rate averages 6.19%. These rates are primarily due to applications of the same size (same number of threads). However, by removing them from the evaluation set, the average error rate increases to 11.3%. Similar results were observed for *PowerAPI* with an average error rate of 5.6% (and 10.7% without same size applications) and a maximum error rate of 19.1%.

In our opinion, this reflects the incapacity of *Scaphandre* and *PowerAPI* to consider the variability of frequencies in application behavior. If this variability could be considered by the family of models of *PowerAPI* and *Scaphandre* (family **(F1)**), defining this power model family becomes difficult as it changes in varying contexts. This is due to the need for a complex definition for residual consumption. Note that it becomes even more complex when taking into account context with Hyperthreading and turboboost as depicted in Section III. A simpler and, in our opinion, more justifiable solution would be to take into account this variability by defining another family of power division models, the family **(F2)** that tries to keep the same ratio in isolated and parallel executions. One approach that could prove promising, would be to construct such a model using a state-of-the-art model that estimates the consumption of each application individually as isolated at the machine level, and uses these estimations to compute a ratio to allocate the actual consumption to each application; constructing such a model is beyond the scope of this paper, but could be the subject of interesting future work.

C) Conclusion - In section III challenges of power division models were raised. In this section, two of them, namely **(C1)** and **(C3)** have been evaluated. It is found that state-of-the-art power division models have difficulty meeting these challenges. First they do not account for the variability in power consumption of different applications, and only CPU time, which is not fully correlated with power consumption, seems to have an impact on the results. Second, because *PowerAPI* and *Scaphandre* don't consider the existence of residual consumption and consider it as an idle consumption allocation, they don't take into account its dynamic aspect and in our opinion fail to meet the challenge **(C3)** by having different allocation policy depending on application behaviors.

V. USING POWER MODELS AS ENERGY MODELS

In the previous section, the evaluation of power division models was discussed. In this section, we will examine the fundamental differences between power and energy division models. To illustrate these distinctions, we selected a set of applications more complex than the stress function from the Phoronix test suite[4]. The selected applications are presented in Table IV. They were selected for their minimal variance since they behave nearly identically when run multiple times within the same context. This characteristic is crucial for the application employed in this experimental protocol since it is necessary to minimize deviations to ensure that the variations

[4]https://openbenchmarking.org/suites/pts

observed are due only to the division algorithm and not to the applications themselves.

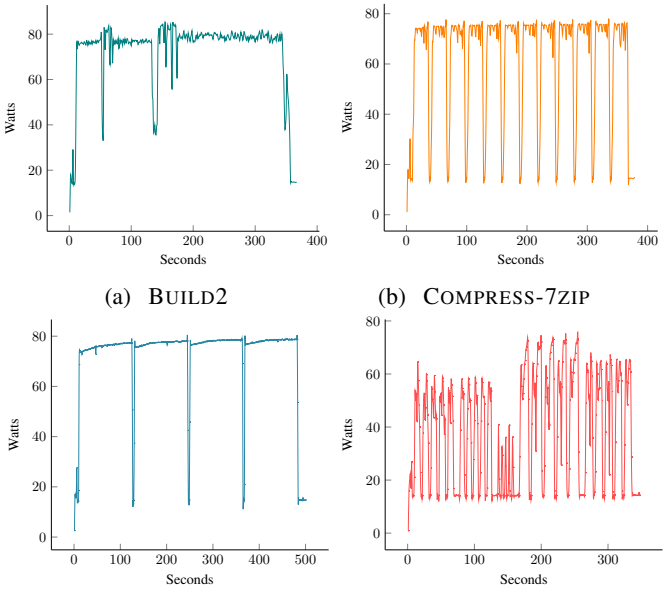
Name	Description	Type
COMPRESS-7ZIP	7zip compression, decompression	CPU
BUILD2	Compilation of the build2 toolchain	CPU , I/O
DACAPO	Java benchmark	CPU , system
CLOVERLEAF	Hydrodynamics benchmark	CPU

TABLE IV: List of Phoronix tests

Application	CLOVERLEAF	DACAPO	BUILD2	COMPRESS-7ZIP
C_s (RAPL, kJ)	36.46 (0.3 %)	13.51 (0.4 %)	26.75 (0.6%)	23.53 (0.2 %)
Execution time (s)	516 (0.4 %)	364 (0.3%)	384 (0%)	396 (0.3 %)

TABLE V: Reference values of Phoronix applications on SMALL INTEL on 6 vcpus

The benchmarks in the Phoronix test suite aim to fully exploit all available resources on the server running them. To execute multiple applications at once, it is essential that each application uses a consistent number of resources- regardless of whether they are executed individually or in parallel. Therefore, VMs are provisioned with a subset of the allocated resources to run the applications. Provisioned virtual machines operate on the Ubuntu 22.04 operating system and have 4GB of RAM and 6 fixed vCPUs. When applications are run within virtual machines, no division is made between applications within a single virtual machine. As a result, for this particular experiment, VMs are regarded as regular applications without further division. To provide a relevant context for production, hyperthreading and turboboost are enabled in this experiment.



(a) BUILD2 (b) COMPRESS-7ZIP

(c) CLOVERLEAF (d) DACAPO

Fig. 10: Behavior of phoronix tests on the SMALL INTEL server according to RAPL

A) Understanding application's behavior from power consumption - In this experiment involving actual applications, the virtual machines (VMs) are set up to utilize 6 vCPUs each. Since the selected host machine (SMALL INTEL) has 12 CPU cores and at most two VMs are active at a time, there is no risk of overloading the machine. Before running together, the applications are first run individually on the machine three times to establish a baseline in Joules. The program was executed three times to confirm its low variability and ensure any future differences in results stem from the energy models. Table V contains the reference values for these runs on the SMALL INTEL node. The first row shows the average consumption of the machine running the application and its variability, and the second row shows the average execution time and its variability. In Figure 10, each chosen application's performance is exhibited while being executed on node SMALL INTEL during a single run. It's noteworthy that the machine consumption variability is really low.

To comply with the protocol outlined in Section III, applications are run a second time in parallel with another application. State-of-the-art models are power division models, and are run at a given frequency to divide the current power consumption of the machine among the running applications. These models could also be utilized as energy division models by calculating the integration of the power curve of each respective application. In a production environment, contexts often change due to the arrival and departure of applications or changes in their behavior. Consequently, power consumption of the machine may fluctuate, significantly impacting the allocation of consumption among applications. For instance, let's consider three applications P_0, P_1 and P_2, all three applications are running a stable workload (with the same isolated residual consumption), but are started and stopped at different times, as shown in Figure 11. As a result of changes in the execution context, power consumption allocation varies over time, despite the application's consistent behavior throughout its execution. The incorporation of changes in power consumption curves will alter the overall energy consumption estimation of applications and their perceived behavior. It is essential to keep this aspect in mind while utilizing energy division models.

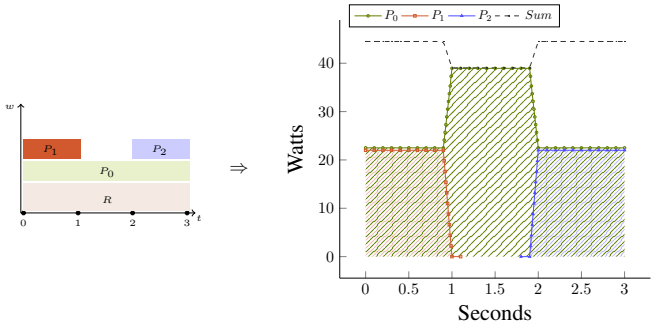
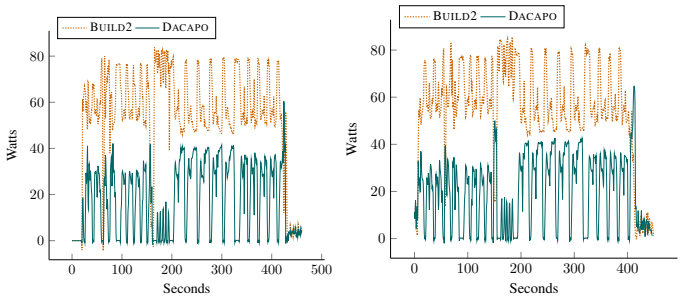


Fig. 11: Illustration of how power division leads to context-dependent results

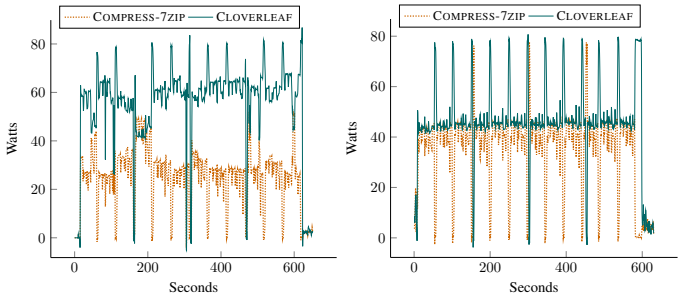
This can be observed using *PowerAPI* and *Scaphandre*. Figure 12 displays the power consumption estimations for BUILD2 and DACAPO when running in parallel. When running sequentially, these two applications require a total of 39 kJ of energy whereas when running in parallel they require 33 kJ, or a reduction of 13% (due to the logarithmic consumption curve and residual consumption of SMALL INTEL); this reduction in consumption is necessarily reported in the estimation of the

consumption of the two applications. For example, when run in parallel with DACAPO, according to *PowerAPI* BUILD2 consumed 24.5 kJ instead of 26 kJ (a decrease of 6%), and DACAPO consumed 8.4 kJ instead of 13 kJ (a decrease of 35%). *Scaphandre* gives matching results. This decrease in energy consumption can vary greatly depending on the execution context. For instance, when running the CLOVERLEAF application on DAHU under different conditions (without contention), energy consumption ranged from 60 kJ to 26 kJ (as reported by *Scaphandre* when executed in parallel of 9 other similar VMs), resulting in a decrease of over 56% in consumption. Any comparison of the energy consumption of two applications in a production context becomes uncertain as the context may have a greater impact on their energy consumption than the applications themselves.



(a) *PowerAPI* (b) *Scaphandre*

Fig. 12: Behavior of BUILD2 vs DACAPO on the SMALL INTEL server



(a) *PowerAPI* (b) *Scaphandre*

Fig. 13: Behavior of COMPRESS-7ZIP vs CLOVERLEAF on the SMALL INTEL server

In addition to the change in overall energy consumption, the behavior of applications cannot be solely explained by the power consumption curve. Although the behavior of DACAPO appears distinguishable in Figure 12, the behavior of BUILD2 is entirely contextual, mirroring the behavior of DACAPO and mistaking its consumption troughs for peaks. A similar observation can be made for CLOVERLEAF when executed against COMPRESS-7ZIP in Figure 13. Notice that *PowerAPI* has difficulty staying stable in its estimations. In the last experiment, CLOVERLEAF and COMPRESS-7ZIP executed the same workload repeatedly, resulting in similar contexts at multiple instants ($t = 150$, $t = 300$, etc.). Nonetheless, *PowerAPI* provided varying estimations in these similar contexts.

B) Conclusion - Assuming that there is a perfect energy allocation model (for any given family **(F1), (F2)**, etc.), this model would be able to capture an abstract vision of the infrastructure by allocating parts of its energy consumption to running applications (for example, to construct *Life Cycle Assessments*). However, we believe, on the other hand, that it won't be applicable to understand the behavior of running applications, nor will it be applicable to support energy optimizations at the application level in a production context, despite suggestions to do so in [4]. These experiments lead to a key result of this paper; power division models produce measurements that are highly context-dependent and, as a consequence, fail to meet the challenge **(C2)**. At this point, we have no reason to believe that this limitation is not inherent to the power division approach, and therefore that a power division model could meet this challenge.

VI. CONCLUSION AND FUTURE WORK

The literature has demonstrated limited attention towards validating power division models, as there were no clear benchmarks for comparison. This lack of clarity in our opinion reflects the lack of a clear definition of the problem of power division. Our paper aims to address this by proposing a new definition of power division models that introduces the possibility of multiple valid divisions that can be grouped into families of models. We didn't aim to provide an exhaustive evaluation (as there are an infinite number of deployment contexts, CPU architectures, applications, etc.), but to show that it is possible to give a formal definition of a power division model, and that the lack of such a definition in the state of the art already leads to significant limitations even in very controlled situations.

Our proposed definition was supported by an analysis of the power consumption of physical machines. Various power division families exhibit distinct characteristics. In our viewpoint, a certain family is noteworthy due to its simplicity in definition and justification, as opposed to the family selected by state-of-the-art models. Based on this definition, we designed a protocol to evaluate power division according to its model family. Our protocol was applied to two separate models, *Scaphandre* and *PowerAPI*, on two machines with different levels of computational power. This evaluation highlights the limitations of current models in accounting for the variability of energy consumption due to the different instruction costs, and their difficulty in accounting for the variability of consumption due to the frequencies of the CPU cores. A discussion of the conceptual differences between a power division model and an energy division model has been presented. If power division models can be utilized to allocate the power consumption of an infrastructure among running applications, this type of model does not seem to be applicable for energy optimization at the application level, as the behavior of applications is completely lost in context.

Because optimizing software in terms of energy consumption remains an important and challenging objective, we propose the following future work. Investigate the possibility of

statically adding debugging information to a program in order to extract the path taken during its execution (e.g. number and type of instructions) and estimate the energy consumption it would have generated if executed alone on a given physical machine. Such a power model would be suitable for creating a power division model of the second family and would also help in optimizing energy consumption.

ACKNOWLEDGEMENT

This work is partly supported by the French government through the *Agence Nationale de la Recherche* (ANR) under the DISTILLER (ANR-21-CE25-0022) grant.

Some 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 (see https://www.grid5000.fr).

REFERENCES

[1] C. M. Olschanowsky, T. Rosing, A. Snavely, L. Carrington, M. M. Tikir, and M. Laurenzano, "Fine-grained energy consumption characterization and modeling," in *2010 DoD High Performance Computing Modernization Program Users Group Conference*, 2010.

[2] M. Jarus, A. Oleksiak, T. Piontek, and J. Weglarz, "Runtime power usage estimation of hpc servers for various classes of real-life applications," *Future Generation Computer Systems*, 2014.

[3] P. K. D. Pramanik, N. Sinhababu, B. Mukherjee, S. Padmanaban, A. Maity, B. K. Upadhyaya, J. B. Holm-Nielsen, and P. Choudhury, "Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage," *IEEE Access*, 2019.

[4] G. Fieni, R. Rouvoy, and L. Seinturier, "SmartWatts: Self-Calibrating Software-Defined Power Meter for Containers," in *CCGRID 2020*, Melbourne, Australia, 2020.

[5] M. Colmant, R. Rouvoy, M. Kurpicz, A. Sobe, P. Felber, and L. Seinturier, "The next 700 cpu power models," *Journal of Systems and Software*, 2018.

[6] 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*, Bangalore, India, 2023.

[7] S. Georgiou, S. Rizou, and D. Spinellis, "Software development lifecycle for energy efficiency: Techniques and tools," *ACM Comput. Surv.*, 2019.

[8] N. Rashid and S. U. Khan, "Agile practices for global software development vendors in the development of green and sustainable software," *Journal of Software: Evolution and Process*, 2018.

[9] [Online]. Available: https://sustainable-computing.io/

[10] J. Pastor and J. M. Menaud, "Seduce: a testbed for research on thermal and power management in datacenters," in *2018 26th international conference on software, telecommunications and computer networks (SoftCOM)*, 2018.

[11] M. D. De Assuncao, J.-P. Gelas, L. Lefevre, and A.-C. Orgerie, "The green grid'5000: Instrumenting and using a grid with energy sensors," in *Remote Instrumentation for eScience and Related Aspects*, 2012.

[12] S. Delamare and L. Nussbaum, "Kwollect: Metrics collection for experiments at scale," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021.

[13] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, 2010, pp. 189–194.

[14] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 2018.

[15] A. Venkatesh, K. Kandalla, and D. K. Panda, "Evaluation of energy characteristics of mpi communication primitives with rapl," in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, 2013.

[16] J. Phung, Y. C. Lee, and A. Y. Zomaya, "Modeling system-level power consumption profiles using rapl," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, 2018.

[17] R. Schöne, T. Ilsche, M. Bielert, M. Velten, M. Schmidl, and D. Hackenberg, "Energy efficiency aspects of the AMD zen 2 architecture," *CoRR*, 2021.

[18] Y. Arafa, A. ElWazir, A. Elkanishy, Y. Aly, A. Elsayed, A. A. Badawy, G. Chennupati, S. J. Eidenbenz, and N. Santhi, "Verified instruction-level energy consumption measurement for NVIDIA gpus," *CoRR*, 2020.

[19] W. Lin, F. Shi, W. Wu, K. Li, G. Wu, and A.-A. Mohammed, "A taxonomy and survey of power models and power modeling for cloud servers," *ACM Comput. Surv.*, 2020.

[20] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010.

[21] A. E. Husain Bohra and V. Chaudhary, "Vmeter: Power modelling for virtualized clouds," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010.

[22] W. Lin, H. Wang, Y. Zhang, D. Qi, J. Z. Wang, and V. Chang, "A cloud server energy consumption measurement system for heterogeneous cloud environments," *Information Sciences*, 2018.

[23] [Online]. Available: https://hubblo-org.github.io/scaphandre-documentation/

[24] L. F. W. Anthony, B. Kanding, and R. Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," *CoRR*, 2020.

[25] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the carbon emissions of machine learning," *CoRR*, 2019.

[26] V. Schmidt, K. Goyal, A. Joshi, B. Feld, L. Conell, N. Laskaris, D. Blank, J. Wilson, S. Friedler, and S. Luccioni, "Codecarbon: estimate and track carbon emissions from machine learning computing (2021)," 2021.

[27] R. Schöne, T. Ilsche, M. Bielert, A. Gocht, and D. Hackenberg, "Energy efficiency features of the intel skylake-sp processor and their impact on performance," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*, 2019.