Automated approach for accurate CPU power modelling

Tomé Maseda, Jonatan Enes, Roberto R. Expósito, Juan Touriño
Universidade da Coruña, CITIC, Computer Architecture Group, Campus de Elviña, 15071 A Coruña, Spain
Email: {tome.maseda,jonatan.enes,roberto.rey.exposito,juan}@udc.es

***Abstract*—Power supply is a limiting factor when increasing the computing capacity of supercomputers. As a consequence, power consumption has become one of the biggest challenges in the field of High Performance Computing (HPC). In order to develop energy-efficient tools (e.g., frameworks, applications), it is essential to have an accurate power consumption modelling. Although previous works proposed a wide variety of approaches to model CPU power consumption, building models in an automated and adaptable way to changing scenarios and predicting power with high precision remains complex due to multiple factors (e.g., training and test workloads, model variables). In this paper, we present a set of tools to fully automate the process of modelling power consumption using CPU time series data. More specifically, our proposal includes two tools: (1) CPUPowerWatcher, which gathers CPU metrics during the execution of user-configurable workloads; and (2) CPUPowerSeer, which builds models to predict CPU power consumption (e.g., polynomial regression) from different CPU variables (e.g., usage, clock frequency) using time series data. Thus, multiple models can be created and evaluated easily, allowing the selection of an optimal model for a specific workload. The experiments conducted by combining these tools allow analys ing the impact of novel factors on CPU power consumption, such as the type of CPU usage generated by different workloads or how the CPU cores are allocated to them. In addition, the accuracy of six regression models is compared when predicting CPU- and I/O-intensive workloads using two different core allocations.**

***Index Terms*—CPU power modelling, Time series, Energy consumption**

I. Introduction

Power consumption has become one of the biggest challenges in many fields of computer science, such as High Performance Computing (HPC). Frequently, the economic cost of the energy consumed by supercomputers represents a significant portion of the total expenses over their lifespan (e.g., initial acquisition, staff, maintenance). Therefore, energy consumption is a constraint on the expansion or acquisition of computing infrastructures, not only because of its costs but also due to its environmental impact. Similarly, this also affects software developers, who seek to enhance the energy efficiency of their codes before deploying them on larger infrastructures.

To address this challenge, both manufacturers (hardware) and administrators/developers (software) are making a joint effort to provide more efficient approaches. On the one hand, hardware designers implement methods to reduce consumption such as Dynamic Voltage and Frequency Scaling (DVFS) [35], automatic shutdown techniques for inactive processor units (power gating), or clock masking techniques (clock gating). On the other hand, the scientific community explores different ways to efficiently manage energy, such as task scheduling based on potential power consumption [26] or more groundbreaking techniques such as power budgeting [21]. A common aspect to all of these approaches is the need for obtaining accurate estimations of power consumption from different hardware resources. In this work we will focus on the CPU, considering it as one of the most energy-demanding resources of a server, along with the GPU. One approach to estimate and predict CPU power consumption is through modelling, that is, the creation of mathematical models that correlate energy consumption with other CPU metrics (e.g., usage percentage, clock frequency, cache hits). Once implemented, this method can be easily integrated into other tools with different objectives related to improve energy management. However, building such models requires the efficient collection and storage of a set of metrics to approximate the relationship between the variables involved as accurately as possible. To perform these tasks, it is necessary to extract, process and analyse time series data using various technologies and tools, as well as to carefully select the variables (i.e., CPU metrics) and the training workloads involved in the models to fine-tune their accuracy. Performing these tasks manually makes power consumption modelling a potentially tedious and error-prone process, which complicates the agile creation of multiple models in order to use them with particular workloads. To overcome these limitations, this paper presents an all-in-one solution to seamlessly generate models to predict a wide range of workloads of different types and intensities.

We summarise our main contributions as follows:

- A toolkit to fully automate the whole process of modelling power consumption (i.e., data collection and fitting). Our solution eases the building of models by monitoring the CPU to be modelled during the execution of a set of synthetic workloads. The resulting models can then be used to estimate the power consumption of any workload executed on that CPU with high accuracy (MAPE lower than 10% across all evaluated scenarios).
- An experimental analysis of the impact of novel factors that directly affect CPU power modelling when using high-level CPU metrics, such as considering the different CPU states (e.g., user, iowait) and the specific cores allocated to workloads (e.g., physical or logical) from a single or multiple CPUs. Up to our knowledge, these

factors have not been taken into account in the literature.
- An experimental evaluation of six polynomial regression models that use different variables and/or training workloads. Our goal is to determine whether it is possible to accurately estimate CPU power consumption relying only on high-level CPU metrics together with modelling methods that are easy to implement and interpret, as well as being computationally efficient.

The remainder of the paper is organised as follows. First, Section II discusses the related work. Next, Section III describes the implementation of the tools developed, while Section IV analyses the impact of novel factors on CPU power modelling and presents the proposed models. Section V compares the accuracy of the models; and finally, Section VI summarises our concluding remarks.

II. Related Work

This section presents the characteristics and methods used in previous works to implement power consumption modelling (Section II-A), as well as those works related to software-based power monitoring and management (Section II-B).

A. Power consumption modelling

When it comes to model generation, there is a wide variety of approaches described in the current literature. First, it is important to distinguish between modelling a whole infrastructure, such as a distributed computing system [14] or a data centre [19], and modelling a specific component, as described for a server [33], CPU [17], memory [18] or GPU [16].

Considering the heavy role that the CPU plays in the energy consumption of current servers [19], particularly in the context of a supercomputer, we focus on CPU power models, and more specifically, on architecture-level models that work at a high level of abstraction. The level of abstraction of a model is based on the data used to predict power consumption and the prediction target. Gate-level models work at a low level and typically predict the consumption of digital circuits (e.g., an adder) based on the activity of their logic gates [20]. Similarly, register-transfer-level models are commonly used to predict the energy consumption of CPU units (e.g., an ALU) based on the activity of signals and registers [38]. Finally, architecture-level models work at a higher level and are often used to predict the whole CPU consumption based on microarchitecture events and/or high-level CPU metrics (e.g., cache hits, CPU usage) [23]. Low abstraction levels require very long simulation times and involve many challenges to deploy the necessary simulation tools across different CPUs (e.g., compatibility between microarchitectures). Therefore, architecture-level models have been selected in order to build them in a more flexible and adaptable way to changing scenarios.

Focusing now on the creation of architecture-level models, some previous approaches are based on event counters integrated into the CPU hardware called Performance Monitoring Counters (PMC). These counters determine the activity factor of specific components [15], essentially measuring how frequently they are accessed to estimate their power consumption. However, a major drawback of PMC-based models is their lack of portability, since the available PMCs vary with each CPU microarchitecture. Statistical models have also been extensively used, many of them developed considering a linear correlation between power consumption and system resource activity factors (e.g., CPU usage, CPU frequency, I/O bandwidth). While some of these models are simple and based on just a few factors [37], others are more complex, even combining these factors with microarchitecture-level events using PMC counters [30]. Aiming to further increase the accuracy, other models include novel parameters such as the concurrency level of a workload along with the average power dissipation of each thread [32], or the complexity of an algorithm in addition to the number of parallel memory accesses it performs [31]. It is also possible to incorporate non-linearity through the explicit use of methods such as polynomial regressions, where power consumption is expressed as a non-linear function of model variables [17]. Finally, recent works make use of machine learning to express this non-linearity using artificial neural networks and support vector regressions [36], graph neural networks [41] or convolutional neural networks [42], amongst others. However, these methods require more computational resources, and they must use larger training datasets along with a set of tuned hyperparameters in order to obtain good accuracy, which reduces their portability. In addition, obtaining optimal hyperparameters is also a difficult task.

Our work proposes non-linear statistical models based on second-degree polynomial regressions that predict CPU power consumption from its usage and frequency. This approach provides a good balance between accuracy, portability and computational efficiency, allowing for the construction of simple and understandable models that can be easily integrated into other energy management workflows.

B. Power monitoring and management

It must be mentioned that hardware vendors offer software interfaces to measure and manage energy, such as the Running Average Power Limit (RAPL) interface [11]. This API reports energy measurements of various system-on-chip power domains that can be accessed through model specific registers [28]. From this point, similar works emerged using RAPL as a data source to model CPU power consumption [27]. RAPL has been widely used to monitor power consumption and its accuracy has been validated across different architectures [29] and workloads [39], demonstrating its ability to isolate CPU power consumption from the rest of the system with low overhead and high scalability [24]. Energy capping techniques through RAPL have also been implemented to enforce memory [18] or CPU [40] power limits.

III. Implementation

Building regression-based models for CPU power consumption mainly requires two essential steps: (1) data collection, that is, monitoring the target CPU to obtain a set of metrics (i.e., regression variables) and store them properly and

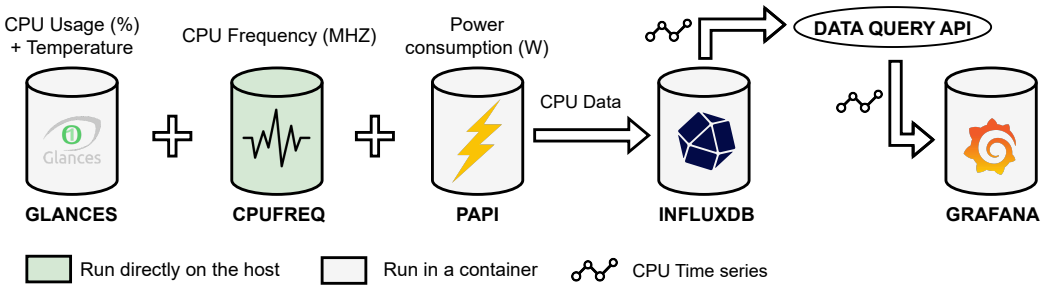


Fig. 1: Monitoring infrastructure for CPU data collection

efficiently; and (2) data fitting, which consists of estimating the correlation between power consumption and the previously retrieved variables. This fact motivates the use of different technologies and tools to perform each step. This section describes in detail the implementation of both steps, as well as the automation of the whole process, which is one of the advantages of our proposal.

A. Data collection and generation

In order to gather CPU metrics several monitoring tools were deployed and orchestrated, as shown in Fig. 1. Glances [5] is in charge of monitoring CPU usage and temperature, the latter being useful to detect thermal throttling. At the same time, the CPUFREQ script retrieves CPU clock frequency from the cores used by the workload being modelled. Finally, a C program using the PAPI library [10] provides access to different RAPL hardware counters that contain information about the CPU energy consumption.

This monitoring solution consists of lightweight agents that can be deployed on different architectures and environments in a simple way, which contributes to enhance the portability of our proposal. These agents can also be executed inside Docker [4] or Apptainer [1] containers, although the latter was used in the experiments of this paper due to its better compatibility with HPC systems. The monitoring agents read and send data with a default sampling period of one second, thus accurately capturing the power variations and their correlation with the CPU metrics when running different operations. To store the metrics, InfluxDB [7] is deployed as a time series database, which can efficiently ingest the received data even when a large number of agents are sending them. To avoid the need for perfectly synchronised update intervals between metrics, InfluxDB data is retrieved using 2-second time windows (i.e., higher than the sampling period). Thus, slightly desynchronised metrics are still retrieved within the same time window [27]. Finally, Grafana [6] is deployed as a visualisation tool, so that InfluxDB time series can be manually analysed when necessary.

When it comes to establishing a relationship between CPU metrics and its power consumption, it is crucial to collect representative data. These data must reflect CPU power consumption over the largest possible range of values for each regressor variable. Therefore, the target CPU must be stressed using different numbers of cores and running diverse types and intensities of workloads, which overall results in varying levels of CPU usage and frequency. To achieve this purpose, a CLI tool named ‘stress-system’ was developed [13], which stresses the target CPU using the cores and usage percentage specified by the user. The CPU usage (-l option) must be set in absolute terms (i.e., 700% usage corresponds to 7 cores being used at 100%). Additionally, the user can provide a list of specific cores to be stressed (-c option), so that the user-defined usage is sequentially distributed among the selected cores. For instance, if the user specifies a 530% usage and a list of six cores, 100% of the usage is assigned to the first five cores and the remaining 30% to the last one. To actually stress the target CPU, our tool executes stress-ng [12] in the background by translating the parameters specified by the user into the required stress-ng executions. Finally, the type and intensity of the workload executed to stress the CPU can also be specified (-s option), currently supporting different stress mechanisms or ‘stressors’ from stress-ng (e.g., ‘all’, ‘sysinfo’, or a mix of both).

B. Data fitting

This process involves determining the specific coefficients of the polynomial regression that are used to model CPU power consumption using the previously obtained metrics. To do so, the CPUPowerSeer tool [2] has been developed to ease the burden of creating regression models from a given dataset using the Scikit-Learn library. Our tool extracts two datasets from InfluxDB to train and test the model, respectively. To identify the start and end periods of the experiments used to obtain the metrics (e.g., running stress-system), CPUPowerSeer requires two timestamp files as input (one per dataset). While each experiment corresponds to a single workload execution under a specific usage percentage, it is worth noting that the training and evaluation of the models involves the execution of multiple experiments. Consequently, timestamp files include several start and end periods. For each period, CPUPowerSeer conducts some queries to InfluxDB in order to gather the required CPU variables for the training or evaluation of the models.

C. Automated model building

In order to be able to automatically create the models we developed another tool: CPUPowerWatcher [3]. This tool is

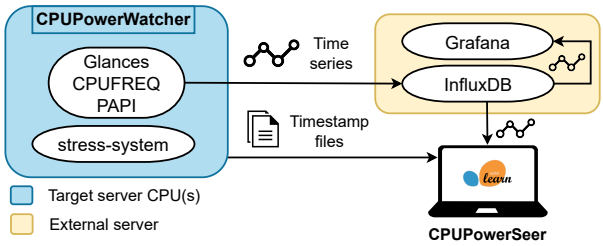


Fig. 2: Architecture for automated model building

in charge of automatically deploying the CPU monitoring infrastructure during the execution of experiments running stress-system or any other workload (see Section III-A). Additionally, CPUPowerWatcher automatically logs the start and end timestamps of each experiment, so that they can be later used by CPUPowerSeer to train and test the models (see Section III-B). As shown in Fig. 2, the proposed methodology involves running Glances, PAPI and CPUFREQ on the target CPU while running the workloads using CPUPowerWatcher. Ideally, InfluxDB and Grafana containers are deployed on separate hardware to reduce noise. Finally, CPUPowerSeer can be executed on any computer with access to the InfluxDB container, in order to be able to retrieve the time series data and automatically build the models.

IV. CPU POWER MODELLING

Leveraging the automated model building previously described, several models can be proposed to explore the most effective approach to estimate power consumption for a wide range of workloads using high-level metrics and straightforward modelling methods. However, for them to be accurate and comprehensive, it is important to take into account some key factors that directly affect CPU power modelling such as the different ways of distributing a workload across available cores, the different CPU usage types, the correlation between power and frequency, the training/test datasets used, or the variables selected to predict power.

A. CPU core distribution

In multisocket servers, workloads can be distributed across the available cores in many different ways, as these systems feature several multicore CPUs. Accordingly, processes or threads of a given workload can use cores from a single or multiple CPUs. Furthermore, these cores can also be physical or logical, with a second thread/process running on the same core through the use of the simultaneous multithreading technology, supported by most current CPUs. Considering that any CPU that has at least one active core incurs a base power consumption and that running two processes on the same physical core through simultaneous multithreading potentially consumes less energy than running them on two separate physical cores, grouping CPU resources as much as possible should reduce consumption and increase power efficiency. Therefore, the specific order in which CPU cores are allocated to workloads, referred to as 'core distribution' from now on, is an important aspect to consider when modelling.

To that end, our stress-system tool was used to conduct a preliminary experiment where we analysed the impact on CPU power consumption when using different core distributions by stressing the CPU incrementally, initially using two cores and allocating two new cores at each two-minute step. To do so, a two-socket server with 16 physical cores per CPU (i.e., 32 logical cores) was stressed running the 'all' stressor for each core distribution shown in Table I (one after the other). It is worth noting that if single-socket servers are used only a subset of distributions are possible (i.e., Group_P, Group_P&L and Group_1P_2L). The results obtained for these experiments are shown in Fig. 3, where CPU usage (user+system) is computed as the average across all core distributions (see the blue line). Group_P has been omitted from the results because, as it only uses physical cores one CPU at a time, it is a subset of Group_PP_LL and their time series would overlap up to half of the execution. Spread_P results are limited to 32 cores (3200% usage) since this distribution only uses physical cores (i.e., 16 per CPU). As can be seen, there is an initial jump in power consumption between 0% and 200% usage (from 50 W to 80 W approximately) due to the differences between an idle and an active CPU. The two distributions that initially use only a single CPU (Group_P&L and Group_1P_2L) find a local maximum (around 130 W) before 3200% usage (maximum value for a single 16-core CPU). While Group_P&L (violet circles) reaches this peak at around 2600% usage, Group_1P_2L (red squares) reaches it earlier (around 1600%), just when all the physical cores of a single CPU have been allocated. Note that both distributions converge to around 120 W from their maximums when they reach 3200% usage, due to operating at slightly lower frequencies since thermal throttling was not detected. The other three distributions, which use both CPUs from the beginning, present significantly higher power values than the previous ones at 3200% usage. At that point, Spread_P&L (orange crosses) is using 16 physical and 16 logical cores taken from both CPUs and consuming about 150 W, while Group_PP_LL (green diamonds) and Spread_P (brown stars), both using 32 physical cores, consume even more (about 180 W).

Overall, the distributions that assign pairs of physical and logical cores increase their consumption more linearly, while those that initially use only physical cores show a more logarithmic behaviour, as adding them leads to a greater consumption increase than just adding logical cores.

B. CPU states

When analysing CPU usage, it is crucial to distinguish between its subcategories: (1) 'idle', which is the percentage of time the CPU is running the special OS idle task when the CPU has nothing to do; (2) 'user' and 'system', which are the percentages of time the CPU is running user- and kernel-space code, respectively (i.e., the CPU is active); and (3) 'iowait', the percentage of time the CPU is idle but there is at least one I/O operation in progress (i.e., the CPU is waiting for I/O).

TABLE I: CPU core distributions used to train the models

Distribution	Description
Group_P	Only physical cores, one CPU at a time
Group_P&L	Pairs of physical and logical cores, one CPU at a time
Group_1P_2L	Physical cores first, then logical cores, one CPU at a time
Group_PP_LL	Physical cores first, one CPU at a time, then logical cores
Spread_P	Only physical cores, alternating between CPUs
Spread_P&L	Pairs of physical and logical cores, alternating between CPUs

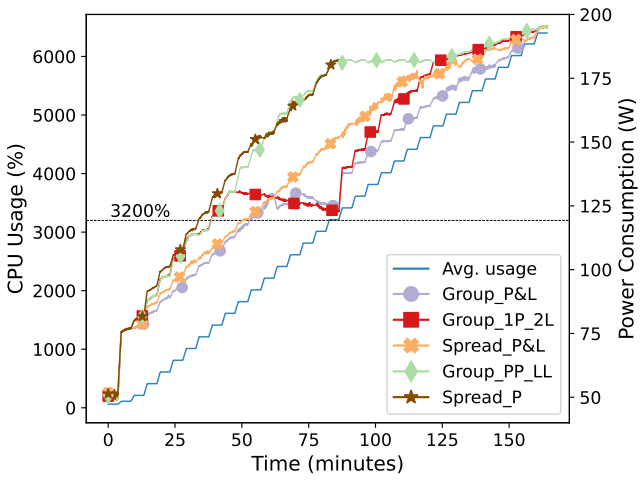


Fig. 3: CPU average usage and power consumption for different core distributions

Considering the huge differences in power consumption between an 'idle' CPU (i.e., usage is close to 0%) and an 'active' CPU (e.g., during workload execution), we decided to model these two states separately to increase the precision of the proposed models using exclusively time series data corresponding to active CPU periods. However, idle consumption was also determined as the average consumption during idle CPU periods. Although it has not been used, these idle measurements can still be useful for evaluating resource and infrastructure management policies, such as the potential benefits of consolidating an infrastructure rather than decentralising its resources.

To analyse the potential differences in power consumption among user, system and iowait states (idle is ignored as previously explained), our stress-system tool was executed using three different stressors from stress-ng: (1) 'all', that iterates over all CPU stress methods to run a balanced load that uses all the processor units, mainly contributing to increase CPU user; (2) 'sysinfo', which runs an OS system call (i.e., times), thus increasing CPU system values; and (3) 'iomix', which performs a mix of sequential, random and memory-mapped I/O operations that contribute to CPU iowait.

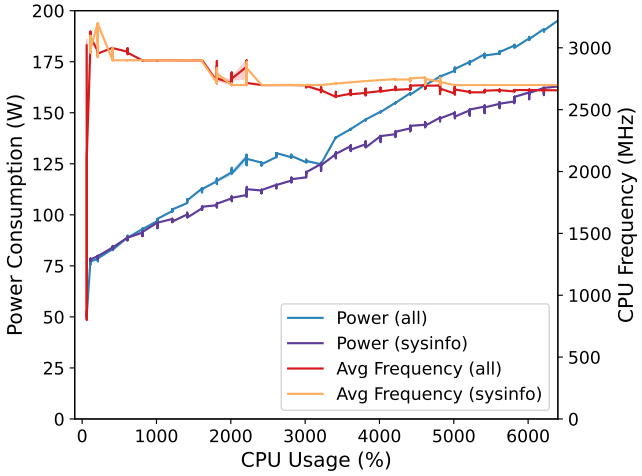
Fig. 4a shows the power consumption for the executions of 'all' and 'sysinfo' stressors at different levels of CPU usage and frequency using the two-socket server described in Section IV-A and Group_P&L as core distribution. CPU usage is presented as the sum of both user+system usages (X-axis), and frequency as the average of all CPU cores being used by the corresponding stressor (right axis). It can be seen that system state consumes less energy than user at comparable frequency and usage levels. For instance, the 'all' stressor consumes around 195 W when using all CPU cores with a sustained frequency of 2700 MHz, while 'sysinfo' consumes about 18% less energy (160 W) under the same conditions.

Fig. 4b shows the power consumption for the executions of 'all' and 'iomix' stressors using the same hardware and Group_P&L as core distribution. It should be noted that 'iomix' usage levels do not exceed 1500%, as this stressor mainly generates CPU iowait usage (instead of user or system). It can be seen that while iowait does not directly affect CPU power consumption, it does have a significant impact on frequency. High iowait values are associated with low frequencies and, consequently, reduced power consumption. These frequency variations are directly related to disk speed. Each disk I/O operation is handled by the OS (i.e., using the CPU), which can account for a significant portion of the total access time (I/O latency). According to [25], as fast disks perform more I/O operations per second than slower ones, the CPU, in turn, must increase its frequency to keep I/O latencies low (i.e., low CPU iowait). This behaviour was confirmed by comparing iomix executions on HDD and SSD disks, as shown in Figs. 5a and 5b. Note that Fig. 5b displays iowait usage on the right axis instead of frequency. From both figures, it can be concluded that SSDs show lower iowait values but higher frequencies and power consumptions at similar CPU usage levels.

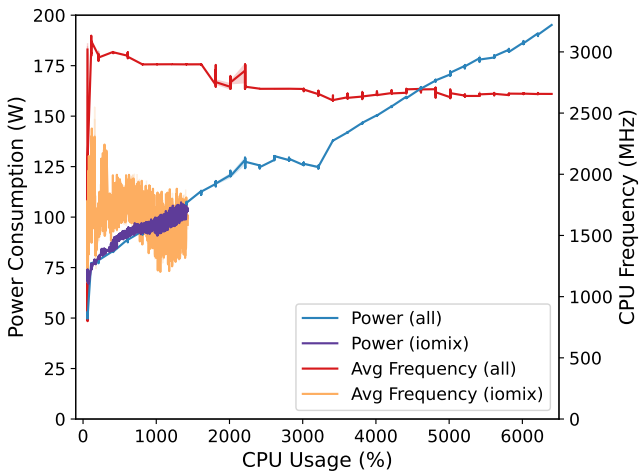
Therefore, workloads that cause high iowait are typically associated with low user and system values, the latter reflecting the I/O operations processed by the OS. This fact motivated the exclusion of iowait from the models proposed in this work, as its effects are captured in the frequency and the other usage metrics.

C. Correlation between CPU power and frequency

The relationship between CPU clock frequency and power consumption is complex. Generally speaking, as the number of active cores increases, power consumption likewise rises, but their average frequency usually decreases following an inversely proportional relationship. This scenario can be seen in Fig. 4a, where frequency decreases and consumption increases as usage does (from left to right). This behaviour is due to the dynamic frequency scaling performed by modern CPUs, where cores can run at different maximum frequencies depending on the number of active cores and the current processor conditions (e.g., power and thermal limits). For instance, increasing the number of active cores on an Intel Xeon Silver 4216 from 1

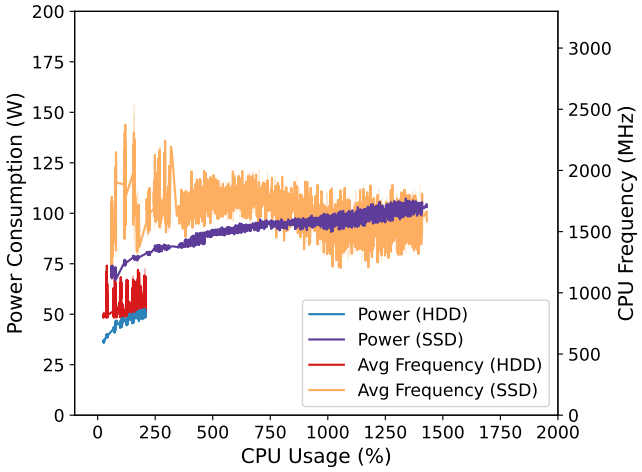


(a) 'all' and 'sysinfo'

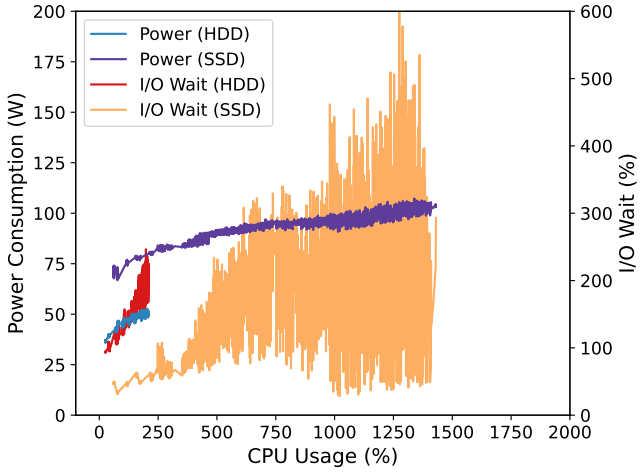


(b) 'all' and 'iomix'

Fig. 4: Power consumption and frequency for different CPU usage levels during stressor executions



(a) Behaviour of CPU frequency (see right axis)



(b) Behaviour of CPU iowait (see right axis)

Fig. 5: Power consumption for different CPU usage levels during the 'iomix' stressor execution on HDD and SSD disks

to 16 automatically triggers a slow-down of their maximum average frequency from 3.2 to 2.7 GHz [8]. Conversely, when active cores are running mainly I/O-intensive workloads, there is a decrease in the CPU usage, frequency and power consumption. In such scenarios, power consumption follows a direct and proportional relationship with frequency. This pattern can be seen in Fig. 5a, where at some points of the execution (e.g., from 0% usage to 750%) all resources increase linearly. In the end, this relationship between consumption and frequency may be directly or indirectly proportional depending on the workload used, so the modelling could be non-trivial. In fact, our preliminary models using average frequency as a variable showed lower accuracy with I/O-intensive workloads. To overcome this issue, we propose the use of the summation of all the frequencies of the active cores as an alternative metric for CPU frequency. This value has a directly proportional relationship with power consumption in both scenarios, as it sums the frequencies of more cores when the CPU usage increases. This approach simplifies power consumption modelling using regression techniques and will be validated in the experiments presented in Section V.

D. Training/test datasets and variable selection

The workload used to generate the training dataset must be representative (similar) to the one used in the actual scenario being modelled, as this typically leads to high accuracy. Furthermore, it is essential for the test datasets to depict realistic scenarios where modelling the CPU power consumption is relevant. This implies not only the execution of CPU-intensive workloads such as scientific simulations, but also I/O-intensive ones, since many real-world codes involve

TABLE II: Proposed CPU power models

Model	CPU model variables	Training workload
M1	U_{user}, U_{system}	all, sysinfo, all+sysinfo
M2	U_{user}, U_{system}, F_{avg}	
M3	U_{user}, U_{system}, F_{sum}	
M4	U_{user}, U_{system}	all, sysinfo, all+sysinfo, iomix
M5	U_{user}, U_{system}, F_{avg}	
M6	U_{user}, U_{system}, F_{sum}	

heavy disk I/O operations while still making intensive CPU usage (e.g., Big Data applications). Once the training and test datasets are known, it is necessary to choose the appropriate model variables. Considering that regression models adjust the weight of each variable based on training data, it is critical that the selected variables are sufficiently represented in these data. Otherwise, the model may assign inconsistent weights to missing variables in the training data. For instance, models trained with workloads that involve both user and system usage are supposed to include both types as independent variables, while avoiding the use of other variables not represented in the data (e.g., iowait).

Considering all these constraints, both user and system usage have always been selected, adding frequency for some models, as they are resources strongly correlated with CPU power consumption according to the literature [17], [30], [36], [37]. Note that iowait was excluded from the proposed models, as its role in defining power consumption is captured by the CPU frequency and usage metrics (see Section IV-B). Other variables, such as temperature, were also discarded as they can be influenced by external factors (e.g., the facilities where the server is hosted).

E. Proposed models

Taking into account all the previously explained factors, we rely on polynomial regressions because they are simple and easy to understand and integrate, as well as low demanding in terms of computational resources and training time. These methods are preferable over more complex ones, such as neural networks, which have several drawbacks: they require careful hyperparameter selection and optimisation, which is difficult and complicates automation; they demand more computational resources for training, even including specific hardware such as GPUs; and our initial tests during implementation showed no significant improvements.

This work proposes six models using a second-degree polynomial regression, which are divided into two groups according to their training workloads (see Table II). These workloads were named after the stress-ng stressors that were executed during data collection, where the 'all+sysinfo' workload represents an equal mix from both stressors. All the models were trained with data corresponding to the execution of these stressors, while the CPU usage was gradually increased according to each of the core distributions shown in Table I. Regarding model variables, U_{user} and U_{system} stand for CPU user and system usage, respectively, while F_{avg} and F_{sum} represent the average and summation frequencies of active cores. While the first model (M1) is the most basic one, including only user and system usage, M2 and M3 add F_{avg} and F_{sum} as an additional variable, respectively. M3 uses F_{sum} instead of F_{avg} to check if solving the CPU power-frequency inconsistencies can improve the results, as explained in Section IV-C.

To achieve better accuracy, the variables of a regression model must be independent. The existence of a strong correlation between regression variables is known as multicollinearity, which can both reduce the accuracy of a model and make it difficult to interpret. To avoid this, interaction terms can be removed for those variables that are strongly correlated. In our scenario, interaction terms for F_{sum} with U_{user} and U_{system} were excluded in M3 and M6, considering that F_{sum} directly depends on the number of active cores ($F_{sum} \approx F_{avg} * \frac{U_{user}+U_{system}}{100}$). The quadratic term was also excluded for F_{sum}, as it was observed to worsen the results in our preliminary experiments. Finally, to assess potential accuracy variations when adding workloads that present high iowait values, the 'iomix' stressor using an SSD disk was added to the training workloads of M4, M5 and M6.

V. Experiments

The models proposed in this paper were evaluated using two multisocket servers that have 2 CPUs with 16 physical cores each (i.e., 32 logical), 256 GiB of memory and one SATA SSD disk. The first server (Server 1) has Intel Xeon Silver 4216 CPUs running at 2.1/3.2GHz base/turbo frequencies, whereas the second one (Server 2) has Intel Xeon Gold 5218 CPUs running at 2.3/3.9GHz. In addition to CPU-intensive workloads, I/O-intensive ones were also evaluated to analyse their impact on power consumption. To test the models under CPU-intensive workloads, the NAS Parallel Benchmarks (NPB) suite [9] was used, and specifically the kernels IS, FT, MG, CG, and BT. Regarding I/O-intensive workloads, the models were tested using the BTIO kernel from NPB and SMusket [22], a DNA error correction tool that represents a real-world Big Data application implemented with Apache Spark. All the workloads were run using two different core distributions: Group_PP_LL and Group_1P_2L (see Table I), in order to assess their impact on the accuracy of the models. These distributions were selected as they are widely used to run HPC workloads. This accuracy was measured using the Mean Absolute Percentage Error (MAPE), an easy-to-interpret metric that does not depend on the scale of the data. We also report the adjusted R-Squared (R^2_{adj}) as a complementary metric, as it has been widely used in previous

TABLE III: Average model accuracies (MAPE in % and R^2_{adj}) using Group_PP_LL

Model	CPU-intensive						I/O-intensive						Total Average	
	Server 1		Server 2		Average		Server 1		Server 2		Average			
M1	6.68	0.89	5.91	0.91	**6.30**	0.90	6.20	0.87	6.28	0.87	**6.24**	0.87	**6.27**	0.89
M2	6.37	0.89	6.18	0.89	**6.28**	0.89	7.75	0.81	10.43	0.51	**9.09**	0.66	7.69	0.78
M3	6.63	0.89	6.30	0.89	**6.47**	0.89	7.31	0.76	7.56	0.79	**7.44**	0.78	6.96	0.83
M4	7.04	0.89	6.83	0.90	**6.94**	0.90	6.32	0.87	5.96	0.87	**6.14**	0.87	6.54	0.89
M5	6.69	0.88	6.27	0.89	**6.48**	0.89	7.80	0.81	7.87	0.83	**7.84**	0.82	7.16	0.86
M6	6.75	0.89	6.63	0.90	**6.69**	0.90	6.47	0.86	5.74	0.88	**6.11**	0.87	6.40	0.88

TABLE IV: Average model accuracies (MAPE in % and R^2_{adj}) using Group_1P_2L

Model	CPU-intensive						I/O-intensive						Total Average	
	Server 1		Server 2		Average		Server 1		Server 2		Average			
M1	8.18	0.82	8.32	0.76	**8.25**	0.79	5.55	0.90	5.51	0.87	**5.53**	0.88	**6.89**	0.84
M2	7.52	0.83	7.40	0.82	**7.46**	0.82	6.64	0.84	9.12	0.42	**7.88**	0.63	7.67	0.73
M3	8.62	0.77	8.24	0.77	**8.43**	0.77	7.01	0.75	6.84	0.74	**6.93**	0.75	7.68	0.76
M4	8.54	0.82	9.35	0.73	**8.95**	0.77	5.67	0.89	5.49	0.87	**5.58**	0.88	7.27	0.83
M5	7.32	0.84	7.54	0.82	**7.43**	0.83	6.39	0.86	6.46	0.87	**6.43**	0.87	6.93	0.85
M6	8.44	0.80	9.08	0.74	**8.76**	0.77	5.74	0.89	5.38	0.86	**5.56**	0.88	7.16	0.82

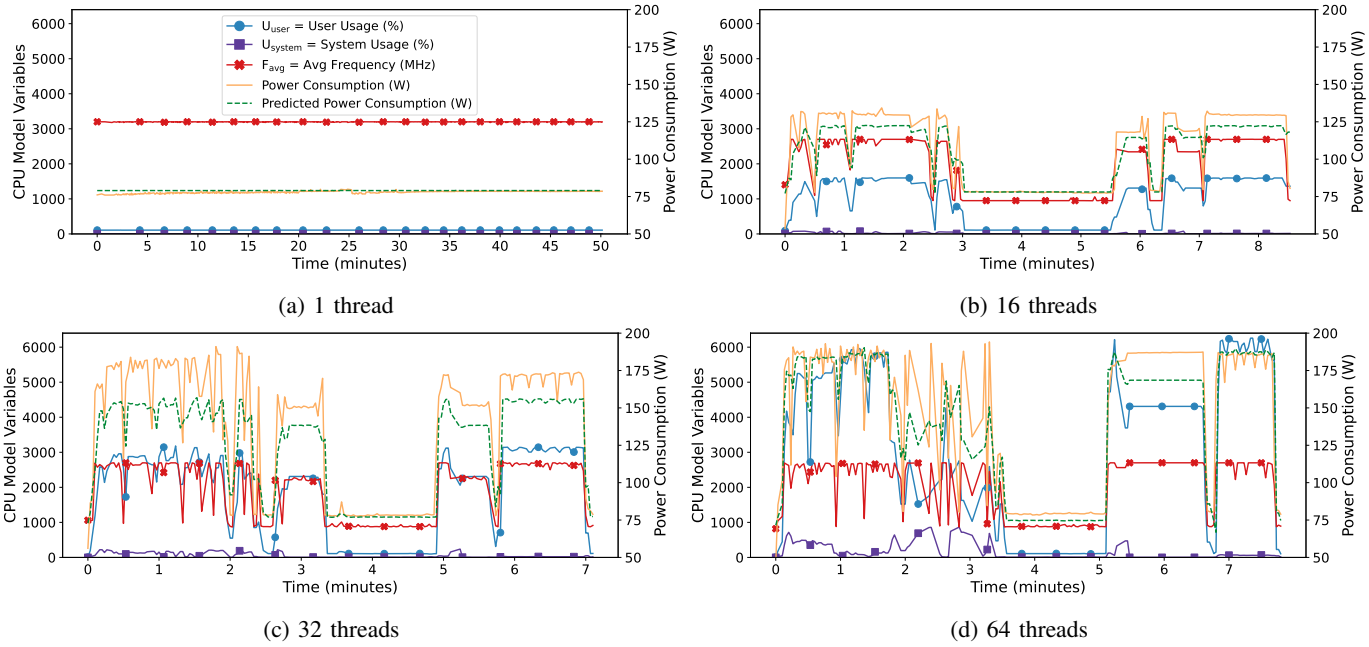
works, even though this metric is not suitable for non-linear model evaluation [34].

A. Results

Table III presents the average accuracies of the proposed models across all CPU- and I/O-intensive workloads on both servers, and using Group_PP_LL core distribution. The inclusion of the 'iomix' stressor in the training workloads (M4, M5 and M6 models) led to decreased accuracy for CPU-intensive codes, as these workloads became less representative of the NPB kernels being predicted. In both servers, M4-M6 models achieve a higher average MAPE (lower accuracy) than those not using 'iomix' (M1-M3). Conversely, their accuracy increases for I/O-intensive codes due to the inclusion of the 'iomix' stressor. It is worth noting that the decrease between M1 and M4 was notably smaller compared to the others (M2 compared to M5, and M3 to M6), as these models do not use frequency, which is the most affected variable when running I/O-intensive workloads (see Section IV-C). When comparing models that define frequency as F_{avg} (M2 and M5) against F_{sum} (M3 and M6), we find that models using F_{sum} are more accurate for I/O-intensive workloads, but less accurate for CPU-intensive ones, as the CPU power-frequency inconsistencies solved using F_{sum} only appear when the CPU runs operations that involve both CPU user/system and iowait (see Section IV-C). Similar results are obtained when using Group_1P_2L, as shown in Table IV. Including the 'iomix' stressor in the training workloads generally worsens the predictions of CPU-intensive codes (except for M2-M5), while improving those of I/O-intensive ones (except for M1-M4). Again, M3 and M6 show increased accuracy with I/O-intensive codes compared to M2 and M5, but they underperform with CPU-intensive ones. Overall, M1 has the lowest average MAPE: 6.27% and 6.89% for Group_PP_LL and Group_1P_2L, respectively, being the best choice for unknown workloads. M2 seems optimal for CPU-intensive codes, with errors of 6.28% and 7.46%, respectively. Finally, M6 is the most accurate model for I/O-intensive workloads, reporting errors of 6.11% and 5.56%.

To further analyse the predictions when running real-world codes, Fig. 6 shows the results for SMusket using 1, 16, 32 and 64 threads and the Group_PP_LL core distribution on the first server, along with the predicted values by the most accurate model for I/O-intensive workloads (M6). Note that F_{avg} is shown in the plots even though M6 relies on F_{sum}, just to avoid having a very large scale on the left axis (i.e., CPU model variables), since F_{sum} can achieve values up to 172800 MHz. The MAPEs for each execution are 1.27% (1 thread), 6.12% (16 threads), 10.80% (32 threads), and 10.06% (64 threads). M6 increases its accuracy for low usage and power values.

When using one thread (Fig. 6a), the model variables and power consumption show low variability throughout the whole execution, and the model fits power behaviour perfectly. However, when increasing the number of threads to 16, 32, or 64 (Figs. 6b, 6c and 6d), the variability of the metrics and the errors of the model increase. In general, M6 tends to achieve better predictions for 16 and 64 threads than for 32. Using 32 threads, a first computing phase can be identified (around minutes 0-2), where the consumption is higher and the model predicts worse. In later phases, the consumption decreases while the predictions improve. When 64 threads are used, five phases can be distinguished. The first, fourth and fifth phases, corresponding to minutes 0-2, 5-7 and 7-8, respectively, show low variability, especially the first and fifth phases, where M6 predictions are very accurate. In contrast, the second phase (minutes 2-4) is characterised by high variability, worsening the model predictions. Finally, in the third phase (minutes 4-5) U_{user} and F_{avg} show minimal values, which are coherently correlated with low power consumption. This third phase corresponds to stages in the SMusket execution where all threads are performing network operations (e.g., 'collectAsMap' Spark jobs). Overall, M6 predictions seem to follow the pattern of power consumption, although they may



(a) 1 thread

(b) 16 threads

(c) 32 threads

(d) 64 threads

Fig. 6: Prediction results for the M6 model running the SMusket application on Server 1

exhibit some offset when variability increases, highlighting its impact on prediction accuracy.

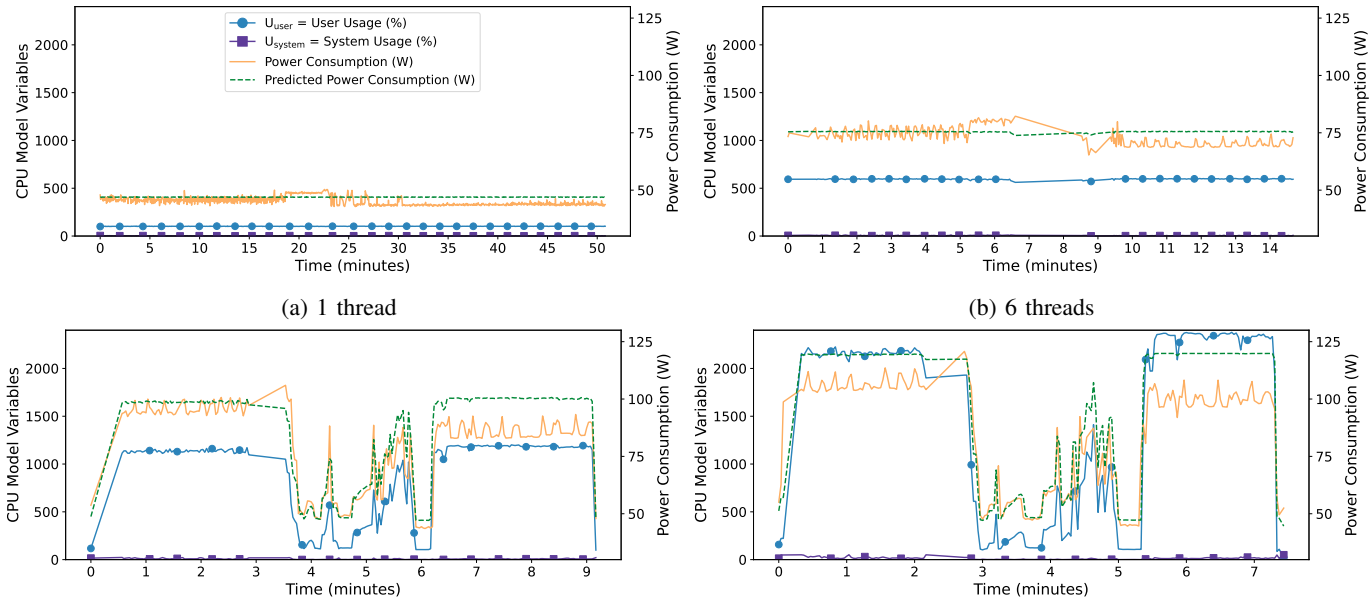
Regarding CPU model variables, there is a direct correlation between them and power consumption, since the latter increases or decreases as U_{user}, U_{system} or F_{avg} do. An example of such behaviour can be seen from minute 0 to 2 during the SMusket execution with 16 threads (see Fig. 6b), where there are two sharp drops for both U_{user} and F_{avg}, leading to a decrease in consumption as well, which is also accurately replicated by M6. In contrast, the relationship between F_{avg} and power consumption varies across different executions. When using one thread (Fig. 6a), F_{avg} keeps values close to 3200 MHz, which corresponds to the turbo frequency, while power remains around 80 W (i.e., the highest frequency but low power consumption). When increasing the number of threads to 16, 32 and 64 (Figs. 6b, 6c and 6d), F_{avg} does not exceed 2700 MHz in any scenario, while power reaches values around 130 W, 180 W and 190 W, respectively (i.e., lower frequency but higher consumption). Therefore, F_{avg} shows a directly proportional relationship with power consumption for a constant number of threads, while this relation becomes inversely proportional across executions with different number of threads. However, M6 uses F_{sum} precisely to overcome this issue, as explained in Section IV-C.

Finally, in order to prove the effectiveness of the solution on other CPU architectures, the proposed models were also trained and evaluated on an AMD Ryzen 9 3900X with 12 physical cores (24 logical), 64 GiB of memory and one SATA SSD disk. To further analyse the predictions when using this CPU, Fig. 7 shows the results for SMusket using 1, 6, 12 and 24 threads, along with the predicted values by the M1 model, which was the most accurate for SMusket on this architecture. Note that frequency is not shown in this figure as the M1 model only relies on U_{user} and U_{system} variables. The MAPEs for each execution are 5.31% (1 thread), 5.57% (6 threads), 10.13% (12 threads), and 13.13% (24 threads).

Similarly to Server 1, the model variables and power consumption show low variability at low usage levels. However, power varies slightly between minutes 20 and 30 during the execution with one thread on this CPU (Fig. 7a) and between minutes 6 and 10 when using six threads (Fig. 7b), resulting in lower model accuracy. These variations correspond to the execution of the 'collectAsMap' jobs previously mentioned. As the number of threads increases, the variability caused by the execution of these jobs increases as well (see Fig. 7c, minutes 4-6, and Fig. 7d, minutes 3-5). Nevertheless, the model accurately follows the variable patterns of power consumption, while it shows higher inaccuracies when predicting power for high usage levels (see Fig. 7c, minutes 6-9, and Fig. 7d, minutes 5-7). Regardless, the model shows a relatively low MAPE considering that it requires low training times and few computing resources, which further proves that the proposed solution can be adapted to different CPU vendors with ease. Additionally, the main advantage of this methodology lies in its ability to evaluate and compare new models in an agile way, thus choosing the one that best fits the target CPU.

VI. Conclusions

Modelling CPU power consumption from time series data is influenced by several complex factors. Previous works have explored aspects such as the correlation between CPU power consumption, usage and frequency. This paper introduces additional factors, including the relationship between



(a) 1 thread

(b) 6 threads

(c) 12 threads

(d) 24 threads

Fig. 7: Prediction results for the M1 model running the SMusket application on an AMD Ryzen 9 3900X

the workload's power consumption and its core distribution and the differences in power consumption across CPU usage types (i.e., user, system, iowait). Considering the various factors involved in modelling, a generic model is preferable for unknown workloads, aiming for good average accuracy across different scenarios. Otherwise, if workloads are known in advance, using multiple specific models trained on similar workloads can yield optimal accuracy in each scenario. In both instances, agile model building and evaluation is critical to select the most appropriate model for a particular case. Therefore, this work also proposes a set of tools to fully automate the whole process, enabling the evaluation of multiple models using different variables (e.g., CPU usage and frequency), prediction methods (e.g., linear or polynomial regression), core distributions (e.g., Group_PP_LL, Group_1P_2L), and training/test workloads (e.g., benchmarks, real applications).

This toolkit is designed to be flexible, allowing easy modification of any of these features and providing support for modelling CPUs from different vendors. One potential future work of the resulting models may be the dynamic control of the energy consumed by an application based on limiting its CPU usage. Furthermore, in scenarios where multiple applications are running on the same CPU, different models could be applied to each one based on its core distribution. The tools developed and presented in this work are publicly available at https://github.com/UDC-GAC/AutoPowerModeling.

Acknowledgment

This work was supported by grant PID2022-136435NB-I00, funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", EU. CITIC, as a centre accredited for excellence within the Galician University System and a member of the CIGUS Network, receives subsidies from the Department of Education, Science, Universities, and Vocational Training of the Xunta de Galicia. Additionally, it is co-financed by the EU through the FEDER Galicia 2021-27 operational program (ref. ED431G 2023/01). This work was also funded by Xunta de Galicia through a predoctoral fellowship (ref. ED481A-2023-035).

References

[1] Apptainer, https://apptainer.org/, [Visited July 2024]
[2] CPUPowerSeer, https://github.com/TomeMD/CPUPowerSeer.git
[3] CPUPowerWatcher, https://github.com/TomeMD/CPUPowerWatcher.git
[4] Docker, https://www.docker.com/, [Visited July 2024]
[5] Glances, https://nicolargo.github.io/glances, [Visited July 2024]
[6] Grafana, https://grafana.com, [Visited July 2024]
[7] InfluxDB, https://www.influxdata.com, [Visited July 2024]
[8] Intel Xeon Silver 4126 frequencies, https://en.wikichip.org/wiki/intel/xeon_silver/4216#Frequencies, [Visited July 2024]
[9] NAS Parallel Benchmarks, https://www.nas.nasa.gov/software/npb.html, [Visited July 2024]
[10] Performance Application Programming Interface (PAPI), https://icl.utk.edu/papi, [Visited July 2024]
[11] Reading RAPL energy measurements from Linux, https://web.eece.maine.edu/~vweaver/projects/rapl/, [Visited July 2024]
[12] stress-ng, https://github.com/ColinIanKing/stress-ng, [Visited July 2024]
[13] stress-system, https://github.com/TomeMD/stress-system.git
[14] Almeida, F., et al.: Energy monitoring as an essential building block towards sustainable ultrascale systems. Sustainable Computing: Informatics and Systems **17**, 27–42 (2018)
[15] Bertran, R., et al.: Energy accounting for shared virtualized environments under DVFS using PMC-based power models. Future Generation Computer Systems **28**(2), 457–468 (2012)
[16] Bridges, R.A., Imam, N., Mintz, T.M.: Understanding GPU power: A survey of profiling, modeling, and simulation methods. ACM Computing Surveys **49**(3), 41:1–41:27 (2016)
[17] Dargie, W.: A stochastic model for estimating the power consumption of a processor. IEEE Transactions on Computers **64**(5), 1311–1322 (2014)

[18] David, H., Gorbatov, E., Hanebutte, U.R., Khanna, R., Le, C.: RAPL: Memory power estimation and capping. In: 16th ACM/IEEE Intl. Symp. on Low Power Electronics and Design (ISLPED 2010). pp. 189–194. Downton, TX, USA (2010)

[19] Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications Surveys & Tutorials **18**(1), 732–794 (2015)

[20] Ding, C.S., Tsui, C.Y., Pedram, M.: Gate-level power estimation using tagged probabilistic simulation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **17**(11), 1099–1107 (1998)

[21] Enes, J., Fieni, G., Expósito, R.R., Rouvoy, R., Touriño, J.: Power budgeting of Big Data applications in container-based clusters. In: IEEE Intl. Conf. on Cluster Computing (CLUSTER 2020). pp. 281–287. Virtual (2020)

[22] Expósito, R.R., González-Domínguez, J., Touriño, J.: SMusket: Spark-based DNA error correction on distributed-memory systems. Future Generation Computer Systems **111**, 698–713 (2020)

[23] Fieni, G., Rouvoy, R., Seiturier, L.: Smartwatts: Self-calibrating software-defined power meter for containers. In: 20th IEEE/ACM Intl. Symp. on Cluster, Cloud and Internet Computing (CCGrid 2020). pp. 479–488. Virtual (2020)

[24] Ilsche, T., Hackenberg, D., Graul, S., Schöne, R., Schuchart, J.: Power measurements for compute nodes: Improving sampling rates, granularity and accuracy. In: Sixth Intl. Green and Sustainable Computing Conf. (IGSC 2015). pp. 1–8. Las Vegas, NV, USA (2015)

[25] Imamura, S., Yoshida, E.: Reducing CPU power consumption for low-latency SSDs. In: IEEE 7th Non-Volatile Memory Systems and Applications Symp. (NVMSA). pp. 79–84. Hakodate, Japan (2018)

[26] Jaiantilal, A., Jiang, Y., Mishra, S.: Modeling CPU energy consumption for energy efficient scheduling. In: 1st Workshop on Green Computing (GCM'10). pp. 10–15. Bangalore, India (2010)

[27] Kavanagh, R., Djemame, K.: Rapid and accurate energy models through calibration with IPMI and RAPL. Concurrency and Computation: Practice and Experience **31**(13), e5124 (2019)

[28] Khan, K.N.: Energy measurement and modeling in high performance computing with Intel's RAPL. Ph.D. thesis, Aalto University (2018)

[29] Lawson, G., Sosonkina, M., Shen, Y.: Towards modeling energy consumption of Xeon Phi. arXiv:1505.06539 (2015)

[30] Rivoire, S., Ranganathan, P., Kozyrakis, C.: A comparison of high-level full-system power models. In: USENIX Workshop on Power Aware Computing and Systems (HotPower'08). p. 5. San Diego, CA, USA (2008)

[31] Roy, S., Rudra, A., Verma, A.: An energy complexity model for algorithms. In: 4th Conf. on Innovations in Theoretical Computer Science (ITCS 2013). pp. 283–304. Berkeley, CA, USA (2013)

[32] Shi, W., Wang, S., Luo, B.: CPT: An energy efficiency model for multi-core computer systems. In: 5th Workshop on Energy-Efficient Design. pp. 1–6. Tel-Aviv, Israel (2013)

[33] Sîrbu, A., Babaoglu, O.: Power consumption modeling and prediction in a hybrid CPU-GPU-MIC supercomputer. In: 22nd Intl. Conf. on Parallel and Distributed Computing (Euro-Par 2016). pp. 117–130. Grenoble, France (2016)

[34] Spiess, A.N., Neumeyer, N.: An evaluation of R2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach. BMC Pharmacology **10**, 6:1–6:11 (2010)

[35] Suleiman, D., Ibrahim, M., Hamarash, I.: Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction. In: 4th Intl. Conf. on Electrical and Electronics Engineering (ICEEE 2005) (2005)

[36] Tarafdar, A., Sarkar, S., Das, R.K., Khatua, S.: Power modeling for energy-efficient resource management in a cloud data center. Journal of Grid Computing **21**, 10:1–10:29 (2023)

[37] Wang, H., Jing, Q., Chen, R., He, B., Qian, Z., Zhou, L.: Distributed systems meet economics: Pricing in the cloud. In: 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10). p. 6. Boston, MA, USA (2010)

[38] Xie, Z., et al.: APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors. In: 54th Annual IEEE/ACM Intl. Symp. on Microarchitecture (MICRO-54). pp. 1–14. Athens, Greece (2021)

[39] Zhai, Y., Zhang, X., Eranian, S., Tang, L., Mars, J.: HaPPy: Hyperthread-aware power profiling dynamically. In: 2014 USENIX Annual Technical Conf. (USENIX ATC 14). pp. 211–217. Philadelphia, PA, USA (2014)

[40] Zhang, H., Hoffman, H.: A quantitative evaluation of the RAPL power control system. In: 10th Intl. Workshop on Feedback Computing. pp. 1–6. Seattle, WA, USA (2015)

[41] Zhang, Y., Ren, H., Khailany, B.: GRANNITE: Graph neural network inference for transferable power estimation. In: 57th ACM/IEEE Design Automation Conf. (DAC'20). pp. 1–6. Virtual (2020)

[42] Zhou, Y., Ren, H., Zhang, Y., Keller, B., Khailany, B., Zhang, Z.: PRIMAL: Power inference using machine learning. In: 56th ACM/IEEE Design Automation Conf. (DAC'19). pp. 1–6. Las Vegas, NV, USA (2019)