MDPI

Article

Analytical Energy Model Parametrized by Workload, Clock Frequency and Number of Active Cores for Share-Memory High-Performance Computing Applications

Vitor Ramos Gomes da Silva [1,†,‡], Carlos Valderrama Sakuyama [1,‡], Pierre Manneback [1,‡], and Samuel Xavier-de-Souza [2,‡]

[1] Department of Electronics and Microelectronics (SEMi), University of Mons, 7000 Mons, Belgium;
[2] Department of Computer Engineering and Automation, Universidade Fedral do Rio Grande do Norte, Natal, Brazil; samuel@dca.ufrn.br
* Correspondence: vitor.ramosgomesdasilva@umons.ac.be
‡ These authors contributed equally to this work.

Abstract: Energy consumption is crucial in High-Performance Computing (HPC), especially to enable the next exascale generation. Hence, modern systems implement various hardware and software features for power management. Still, due to numerous different implementations, we can always push the limits of software to get the most efficient use of our hardware. To be energy efficient, the software relies on the dynamic frequency and voltage scaling (DVFS) as well as dynamic power management (DPM). Yet, none have privileged information on the hardware architecture and application behavior, which may lead to energy-inefficient software operation. This work proposes analytical modeling for architecture and application behavior that can be used to estimate energy-optimal software configurations and provide knowledgeable hints to improve DVFS and DPM techniques for single-node HPC applications. Additionally, model parameters such as the level of parallelism and dynamic power provide insights into how the modeled application consumes energy, which can be helpful for energy-efficient software development and operation. This novel analytical model takes the number of active cores, the operating frequencies, and the input size as inputs to provide energy consumption estimation. We present the modeling of 13 parallel applications employed to determine energy-optimal configurations for several different input sizes. The results show that up to 70% of energy could be saved on the best scenario compared to the default Linux choice and, 14% on average. We also compare the proposed model with standard machine-learning modeling concerning training overhead and accuracy. The results show that our approach generates about 10 times less energy overhead for the same level of accuracy.

Keywords: Energy Model; Dynamic Frequency and Voltage Scaling; Dynamic Power Management; High Performance Computing

Citation: Silva, V.; Valderrama, C.; Manneback, P. and Souza, S. Analytical Energy Model Parametrized by Workload, Clock Frequency and Number of Active Cores for Share-Memory High-Performance Computing Applications. *Energies* **2021**, *1*, 0. https://doi.org/

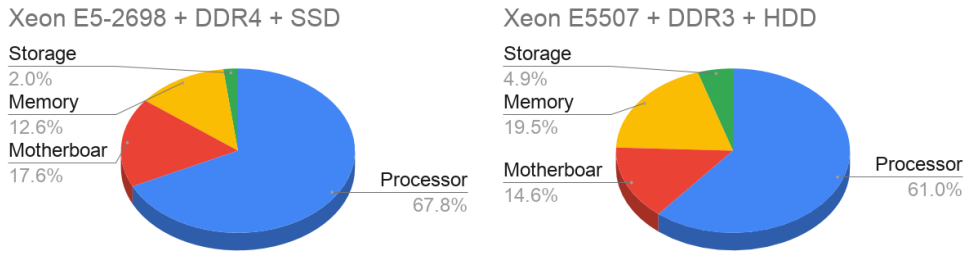
Received:
Accepted:
Published:

1. Introduction

Data center's energy efficiency has attained crucial importance in recent years due to its high economic, environmental, and performance impact. For example, the leading Petaflop supercomputers consume a range of 1–18 MW of electrical power, with 1.5 MW on average, which can be easily translated into millions of dollars per year in electricity bills [1]. Datacenter energy consumption was estimated to be between 1.1% and 1.5% of worldwide electricity usage in 2010 [2,3], generating as much pollution as a nation like Argentina [4]. In some cases, the power costs exceed the cost of purchasing hardware [5]. Furthermore, the energy costs of powering a typical data center doubles every five years [6]. Therefore, with such a steep increase in power use, electricity bills have become a significant expense for today's data centers [7,8]. Due to these reasons, data center

energy efficiency is now considered a chief concern for data center operators, often ahead of the traditional considerations of availability and security.

There are several approaches for green computing, from electrical materials to circuit design, systems integration, and software. These techniques may differ, but they share the same goal: substantially reduce overall system energy consumption without a corresponding negative impact on delivered performance. The processor and main memory are usually the components that dominate power consumption, as shown in Figure 1. The processor can consume as much as 50% of the total energy [9–11]. For that reason, modern processors incorporate several features for power management [12–14], such as Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVFS). DPM encompasses a set of techniques for obtaining energy-efficient computing by deactivating or reducing the system component's performance when they are idle or partially utilized [15,16]. DVFS allows the frequency and voltage to be adjusted in run-time depending on current needs.



(a) This work. **(b)** Study case in [11].

Figure 1. Power breakdown of a typical node of an HPC cluster at full use. The system used in this work (a) was built in 2016 and equipped with two Intel Xeon E5-2698, 128 GB of DDR4 memory and SSD as storage, while (b) the case study in [11] was built in 2012 and equipped with two Xeon E5507, 32GB of DDR3 memory and HDD as storage.

DVFS is motivated by the well-known fact that frequency and power have a near-cubic relationship [1,2] this implies that running the CPU at a lower frequency causes a linear reduction in performance and a near-cubic reduction in power, which could lead to a near-square reduction in CPU energy. Because of that, it is possible to archive dramatic energy savings just with frequency control depending on the system and its architecture. Although very promising, the system software has yet to determine when and what voltage and frequency to use when running applications. Otherwise, not only will performance deteriorate, but in the worst case, energy consumption would also increase [1]. Indeed, reducing the frequency results in a longer execution time, which increases the energy consumption of other system components such as memory and disks. There is also an overhead of time and energy associated with a voltage and frequency switch that need to be considered. Thus, finding the most appropriate voltage and frequency to use in all circumstances is not easy. Therefore, since its introduction in 1994, there has been a tremendous amount of research on DVFS algorithms.

The DPM technique can achieve substantial energy savings on systems where the static power is high or the system remains inactive for a long time. In that case, the problem is to determine when and which components to turn on/off. With DPM, energy savings of 70% have been reported [15,16].

However, at the same time these power-saving techniques reduce system energy, they can compromise the performance leading to a lead to a complex trade-off that needs to be carefully exploited to produce more energy-efficient algorithms. Indeed, this work investigates whether the construction of an energy consumption model of an application can lead to significant energy savings.

We propose an analytical energy model for a given application in the function of the two control variables present in most HPC systems: CPU operating frequency and number of active cores. The model is composed of three application-dependent parameters and three parameters relating to the architecture of the system. The application parameters incorporate characteristics of the percentage of parallelism and the input size. The system architecture parameters include power-related and technology-dependent components such as dynamic, static, and leakage power.

The proposed model can help to improve DVFS and DPM methods since it estimates the contribution of each of the parameters to the total energy consumption.

We have organized the rest of this paper in the following way. In Sections 2 and 3, we present a general review of existing models showing the differences between each approach and its applications. In Section 4, we propose our model and derive its parameters alongside its constraints. In Section 5, we validate the model with the PARSEC benchmark applications. Forward, in Section 5.8, we present use cases of the model as well as how we applied it in DVFS algorithms. Finally, we conclude with a discussion in Section 6.

2. Theoretical background

A model is a formal representation of a natural system. Computer system models representation includes equations, graphical models, rules, decision trees, representative collections of examples, and neural networks. The choice of representation affects the model's accuracy, as well as its interpretability by people [17–19]. Accurate energy and power consumption models are essential for many energy efficiency schemes employed in computing equipment [5], and they can have multiple uses, including the design, forecasting, and optimization of data center systems. This work focuses on analytical models that could aid energy optimizations and analyses of crucial factors in the total energy draw.

The desirable properties of a full-system model of energy consumption include accuracy, speed, generality and portability, inexpensiveness, and simplicity [20]. However, modeling the exact energy consumption behavior of an HPC system is not straightforward, either at the whole-system level or at the level of individual components. Data centers' patterns of energy consumption depend on multiple factors such as hardware specifications, workload, cooling requirements, or the type of the applications. Some of these factors cannot be measured easily. Furthermore, it is impractical to perform detailed measurements of the energy consumption of lower-level components without additional overhead.

Several proposed models have already been classified concerning its input parameters, as shown by Dayarathna et al. [2], who analyzed more than 200 models according to their characteristics and limitations and classified them into categories where the model is more suited to its objectives:

- System Utilization or Workload
- Frequency
- Other system states such as cache miss, branch prediction, number of instructions executed, and more

Often, energy models are described as a combination of two main parts, the power model of the system and the performance model of the application. This is because the concept of Energy (E) is the total amount of work performed by a system over a period of time (T), while power (P) is the rate at which the system performs the work. The relation between these three amounts can be expressed as:

$$E = \int_0^T P(t)dt. \tag{1}$$

2.1. Power models

The modeling of system parameters is becoming popular nowadays with the advantage of performance counters provided by the CPU or the operating system. These counters can measure micro-architectural events, such as instructions executed, cache hits, miss-predicted branches, and more. Thus, providing a base for many different estimations of power usage. This makes this type of model very suitable for power estimation because it can use information about several internal states of the computer.

Frequency-based models are the most common kind of model. They serve as a base for many power models [21–23]. These models utilize the fact that every digital circuit (including modern processors) is composed of transistors. Thus, modeling one transistor's interaction and scaling this to the chip can give a reasonable estimate of the entire system's energy. One of the most common frequency base model approximations is defined as follows:

$$P = \alpha + \beta f^3, \tag{2}$$

where α and β are model parameters, and f is the operating frequency (details of this equation are covered in Section 4). This type of models is suitable for optimization problems since they are a function of the operating frequency, which can be easily controlled.

2.2. Performance models

The most common way to model the application performance is using the workload. The workload is an abstract representation of the amount of work done in a given time and speed. The workload (W) can be defined in many different ways. One common way used in many works such as Paolillo et al. [24], Francis et al. [1], and Kim et al. [25] is the following:

$$W = \int_0^\tau s(t)dt = s\tau, \tag{3}$$

where τ is total active time, and s is the execution speed in instructions/second.

Utilization models [1,26] are also found in the literature, defined as the ratio between the time that the system is active and the total time (idle and active). These models are present in many DVFS algorithms present in Linux. They can be seen as a good alternative to the workload since it is impossible to measure workload in real-time. The Eq. (4) defines workload in terms of CPU utilization (u):

$$u = \frac{\tau}{T} = \frac{W/s}{T}, \tag{4}$$

where T the total execution time (idle and active), and τ is the active time, meaning when the processor was executing instructions. Models based on CPU utilization are the basis for DVFS algorithms. Even though this is not an controllable parameter, it is straightforward to measure system utilization with almost no overhead and it is also very portable in terms of operating systems and architectures.

3. Related work

Merkel et al. [27] developed an energy model for processors based on events. Their model assumes a fixed energy consumption α_i for each activity, and by counting the number of occurrences c_i of every activity they estimate the total energy as:

$$E = \sum_{i=1}^{n} \alpha_i c_i. \tag{5}$$

.

Another event-based model introduced by Roy et al. [28], described the computational energy consumed by a CPU for an algorithm A as the Eq. (6):

$$E(A) = P_{clk}T(A) + P_wW(A), \tag{6}$$

where P_{clk} is a processor clock leakage power, $T(A)$ is the total execution time, $W(A)$ is the total time taken by non-I/O operations, and P_w is used to capture the power consumption per operation performed by the CPU. $T(A)$ and $W(A)$ are estimated using performance features.

Models based on events present some drawbacks, they are highly dependent on the operating system and its architecture, making them problematic to port for another platforms. There are also limitations regarding the number of simultaneous events that can coexist without adding a non-negligible overhead. Additionally, there are cases where events need multiplexing, for example, when using more hardware events that the CPU can provide. There are also some well know problems regarding the precision of some events as shown in many works [29–34]. Some events that should be exact and deterministic (such as the number of executed instructions) show run-to-run variations and over-count on various architectures, even when running in strictly controlled environments. Because of that, our proposed model is not dependent on events, therefore, not vulnerable to those drawbacks.

An instruction-level energy model was also proposed in [35] by Yakun et. al. Where they proposed an energy per instruction (EPI) characterization made on Xeon Phi. Their model is expressed as:

$$E(f) = \frac{(p_1 - p_0)(c_1 - c_0)/f}{N}, \tag{7}$$

where N is the total number of dynamic instructions, p_0 is the initial idle power, p_1 is the average dynamic power, and (c_1-c_0) refers to the cumulative number of cycles the micro-benchmark performs. This model is suitable for estimating the energy after the application finishes executing when it is possible to count the total cycles. However, it is challenging to use for optimization or forecasting since it does not have an application model to predict the cycles. Our model integrates the behavior of the application, taking into account the execution time.

Lewis et al. [36] described the overall system energy consumption using the following equation:

$$E = A_0(E_{proc} + E_{mem}) + A_1E_{em} + A_2E_{board} + A_3E_{hdd}, \tag{8}$$

where, A_0, A_1, A_2, and A_3 are unknown constants that are calculated via linear regression analysis and those remain constant for a specific server architecture. This model, as the previous one, relies on knowledge of energy spent on each component, being a suitable option for estimation after the application has already ran, but not for optimization of the run itself, which is the aim of our model.

In another energy consumption model based on system utilization, Mills et al. [37] modeled the energy consumed by a compute node with CPU (single) executing at speed σ as Eq. (9),

$$E(\sigma, [t_1, t_2]) = \int_{t_1}^{t_2} \sigma^3 + \rho\sigma_m ax^3 dt, \tag{9}$$

where ρ stands for the overhead power consumed regardless of the processor speed, t_1 and t_2 are the initial and final execution time of the application. The overhead includes power consumption by all other system components such as memory, network, and more. For this reason, although the authors mentioned the energy consumption of a socket, their power model is generalized to the entire server. This model lacks of a closed-form i.e. it depends on the definition of $\alpha(t)$ to be complete. Our model has a closed-form which facilitates analyses.

Liu et al. [38] adopted another approach to model energy, where the model uses a Markov chain to determine energy state transitions. However, they approach needs heuristic algorithms to search for the best transitions and does not provide insights about the problem as an analytical equation does.

Although much work has been done in DVFS, the focus is still on the consumer electronics and laptop markets. For HPC, the notion of energy perception is relatively new [39]. Moreover, the operational characteristics of non-HPC and HPC systems are significantly different. First, the workload on non-HPC systems is very interactive with the end-user, but the workload on the HPC platform is not. Second, activities conducted on a non-HPC platform tend to share more machine resources. In contrast, in HPC, each job often runs with dedicated resources. Third, an HPC system usually is much larger than non-HPC systems, making it more challenging to gather information, organize decisions, and execute global decisions. Therefore, it is worthwhile to investigate whether a DVFS scheduling algorithm, which works well for conventional computing, remains effective for HPC.

Our work proposes a full-system energy model based on the CPU frequency and the number of cores. The model aims to understand and optimize the energy behavior of parallel applications in HPC systems according to application parameters such as the degree of parallelism and CPU parameters related to dynamic and static power. The proposed model differs from existing ones for including the frequency and number of cores in the same equation for estimating the energy for a specific application in a given configuration. This model can serve as a base for DVFS and DPM optimization problems that include frequency and active cores. It can also be used to analyze the contribution of each parameter (ex: level of parallelism) to energy consumption. The number of cores is an essential factor in HPC since applications are designed to run on multiple cores.

The proposed energy model is the product of an application-agnostic power model and an architecture-specific application performance model. The power model is based on the CMOS logic gates power draw as a function of the frequency [21,22] augmented to include the number of cores. The performance model is based on Amdahl's law [40–42], which can be used to estimate runtime in multi-core systems. In addition, this model has been extended to include execution frequency and input size, characterizing the application on the target architecture.

4. Modeling energy with performance and power

This section describes the models proposed for power, performance, and energy.

4.1. Power Model

The developed power model is based on the developed frequency models [43–46]. In this approach the idea is to reduce the complexity of the processor dynamics by looking only at the main element that it is composed of, the transistor. Thus, modeling the power consumption can be resumed to model the logic gates and multiplying this by the total number of gates, reducing the complexity of the modeling process.

FINFET and MOSFET compose the main techniques to manufacture transistors. However, FINFET is the more recent one and has gradually replaced the mature technology MOSFET. Despite having different characteristics, they have aspects in common that can be modeled [43–46]. Namely, static power P_{static}, dynamic power P_{dynamic}, and leakage power P_{leak}, that accumulated compose and approximation the total power draw.

The dynamic power and leakage power behavior can be approximated by the following equations, respectively, as shown by Sarwar et al. [21] and Butzen et al. [22].

$$P_{dynamic} = CV^2 f, \tag{10}$$

$$P_{leak} \propto V, \tag{11}$$

where C is the load capacitance, V the voltage applied to the circuit, and f the switching frequency.

Another common approximation is to expect a linear relationship between the voltage and the applied frequency [23] such that:

$$f \propto V, \tag{12}$$

Thus, the proposed model for one processing core of a multi-core processor is derived by using Eq. (10), Eq. (11) and Eq. (12) to write Eq. (13).

$$P(f) = c_1 f^3 + c_2 f + c_3, \tag{13}$$

where c_1 c_2, and c_3 are the model's parameters associated with the dynamic, leakage and static power aspects, respectively. Including the number of active cores p, the proposed estimation of the power consumption of the whole processor becomes Eq. (14)

$$P(f,p) = p(c_1 f^3 + c_2 f) + c_3, \tag{14}$$

4.2. Performance Model

We consider a program as a set of instructions executed on a mean frequency f with c_k instructions per cycle to model the application execution time. The time T_f that this program will take to complete at a given frequency is devised as follows:

$$T_f = \frac{I}{c_k f}, \tag{15}$$

where I is the total number of instructions and c_k the ratio of instructions per unit of time.

The next step is to include the number of cores in the equation. Amdahl's law [40], gives the theoretical background for that. It describes the speedup in latency of the execution of a task at a fixed workload.

$$S = \frac{T_s}{T_p} = \frac{1}{1 - w + \frac{w}{p}}, \tag{16}$$

where T_s is the serial time, T_p the parallel time, S is the theoretical speedup of the execution of the whole task, w is the proportion of the execution time that benefits from improving system resources, and p is the speedup part of the task that benefits from improved system resources. Combining this with Eq. (15), the parallel time at frequency f can be written as:

$$T_p = \frac{T_s}{S} = \frac{T_f}{\frac{1}{1-w+\frac{w}{p}}}, \tag{17}$$

We can then write the equation of the program execution time as a function of frequency, number of cores and parallelism as Eq. (18) and subsequently derive Eq. (19):

$$T(f,p) = \frac{I}{\frac{c_k f}{1-w+\frac{w}{p}}}, \tag{18}$$

$$T(f,p) = \frac{d_1(p - wp + w)}{fp}, \tag{19}$$

where d_1 is a constant.

Finally, to fully characterize the application, a parameter representing the application's workload, called input size N, is introduced, representing the number of basic operations need to complete a problem [47]. In Oliveira et al. [48], they showed that this parameter could generally be described as exponential. Therefore the proposed

performance model is presented in Eq. (20). This resulting equation describe the behavior of the execution time of a program for an input N, frequency f, and active cores p:

$$T(f,p,N) = \frac{d_1 N^{d_2}(p - wp + w)}{fp}, \tag{20}$$

where d_1, d_2 and w are constants that depend on the application.

4.3. Energy Model

Combining the power model output described in Eq. (4.1) and the characterization of the application performance described in Eq. (4.2), the total energy can be modeled as:

$$E(f,p,N) = P(f,p) \times \mathrm{T}(f,p,N), \tag{21}$$

where $P(f,p)$ is the total power modeled by Eq. (14), $T(f,p,N)$ is the execution time estimated by the Eq. (20), f is the frequency, p is the number of active cores, and N is the input size. The final equation can be written as:

$$E(f,p,N) = \frac{d_1 N^{d_2}(p - wp + w)(p(c_1 f^3 + c_2 f) + c_3)}{fp}. \tag{22}$$

5. Experimental validation

In this section, the models presented in 4.1 and 4.2 were validated with a benchmark specific for multi-core architectures. Additionally, in order to assess the modeling overhead and accuracy, our proposal was then compared to machine learning approaches. We compared against Support Vector Regression (SVR) [49], Decision Tree [50], k-nearest neighbors [51], Multilayer perceptron [52], and some new methods such as Gao et al. [53]. However, SVR was chosen as the most representative because it performed best in our tests without aggressive fine-tuning.

5.1. Case-Study Architecture

The experiments were executed in one computer node equipped with two Intel Xeon E5-2698 v3 processors with sixteen cores each and two hardware threads for each core. The overall view of the architecture is shown in Figure 2. The maximum non-turbo frequency is 2.3GHz, and the total physical memory of the node is 128GB (8×16GB). Turbo frequency and hardware multi-threading were disabled during all experiments. The operating system used was Linux CentOS 6.5, kernel 4.16.

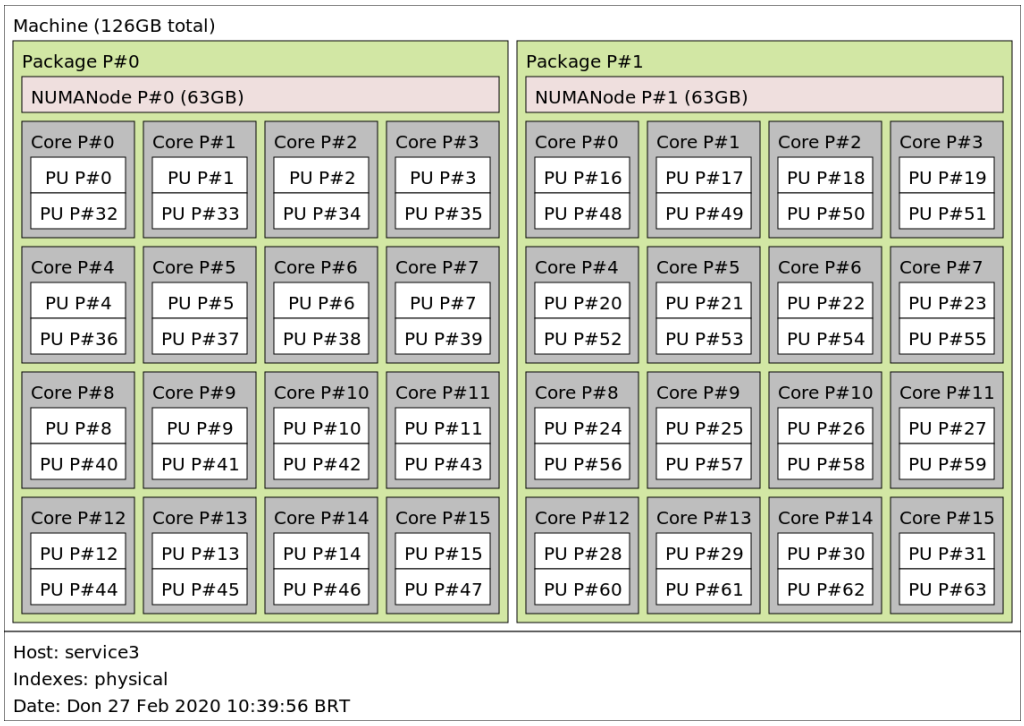


Figure 2. Node architecture (the image was made with the lstop application).

The Linux kernel has many different policies for power management, depending on the driver. In the default driver, the acpi-cpufreq, the options are:

- Powersave
- Performance
- Ondemand
- Conservative
- Userspace

In this work, the frequency control was performed using the Userspace governor, and the core control was accomplished by modifying the appropriate system files with the default CPU-hotplug driver.

The architecture is equipped with the Intelligent Platform Management Interface (IPMI), a set of interfaces allowing out-of-band management of computer systems and platform-status monitoring via the local network [54]. It can monitor variables and resources such as the system's temperature, voltage, fans, and power supplies, with independent sensors attached to the hardware.

5.2. Verifying Hypothesis

In this section we validate whether the assumptions of our model are valid for the system used.

5.2.1. Frequency and voltage relation

One of the assumptions was that the frequency and the voltage have a linear relationship as mentioned in Eq. (12). To verify that, we build an experiment that set the frequency to a specific value while sampling the voltage using the APERF and MPERF registers that provide feedback on the current CPU frequency. The average result of the sampling voltages are shown in the Figure 3.

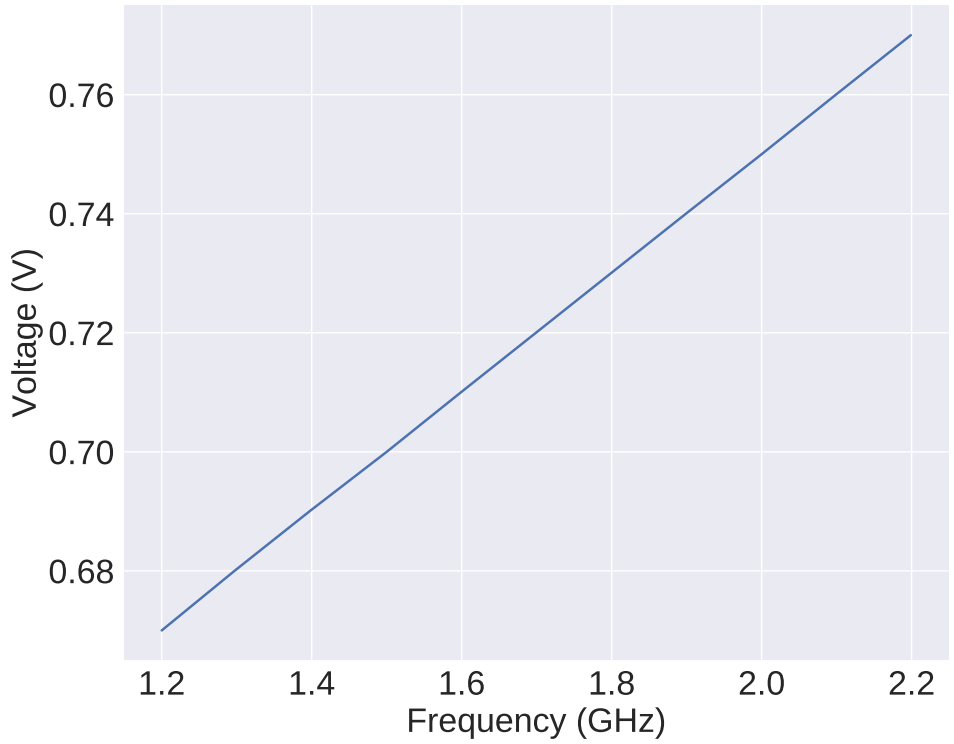
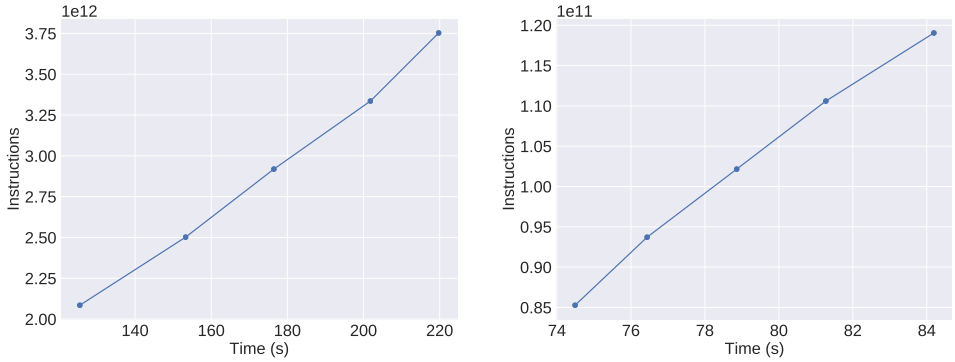


Figure 3. Frequency voltage relation

As we can see in the Figure 3, there is a linear relation between them. Manufacturers implement this curve in the processors using tables that relate ranges of frequencies to voltages. Because of that, they can precisely define any curve that will better suit their design.

5.2.2. Input size and instructions

We ran the applications with different inputs assuming a linear growth in the amount of work for one input to the other when building our model. However, measuring and controlling the amount of work would require much instrumentation and tuning to find an input that corresponds to a certain amount of work. Therefore, to build our models, we use the time as reference to the amount of work, assuming that the work is proportional to the executing time.Figure 4 correspond to the verification of this supposition.

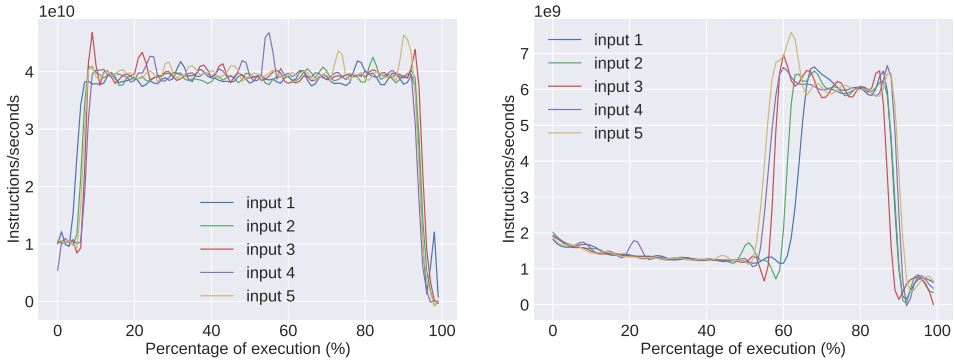


(a) Blackscholes. **(b)** Canneal.

Figure 4. Relation between time and instructions for each input size.

Figure 4 shows that the assumption was reasonable. This was the case for all applications that we ran in our benchmark, and should hold for any data parallelism type of application.

The next assumption was that the behavior of the application was the same when varying the workload. This condition is necessary for using the model with an unknown input size because if the behavior is the same, we can simply interpolate the known inputs. One way to verify this is to measure the rate of instructions per second normalized by the frequency as shown in the Figure 5.



(a) Blacksholes. **(b)** Canneal.

Figure 5. Rate of instructions per second varying the input size normalized by the frequency.

Figure 5 shows that the applications have roughly the same curve when normalized, this also happens for all others applications in our benchmark.

The final assumption is that the workload should also not vary depending the number of cores or frequency. To verify we measure the total number of executed instructions while varying the cores from 1 to 32. Table 1 show the results.

Table 1: Variation of the number of instructions when changing the number of cores for the same input.

Application	Instructions	Dev.	Dev. (%)
Vip	7.97e+11	7.16e+06	0.00%
Openmc	8.17e+07	1.65e+04	0.02%
Rtview	9.91e+12	1.55e+09	0.02%
X264	4.52e+11	5.81e+07	0.01%
Bodytrack	1.86e+12	3.95e+10	2.13%
Fluidanimate	2.09e+12	8.44e+10	4.04%
Xhpl	1.14e+08	1.24e+05	0.11%
Blackschole	3.75e+12	1.40e+09	0.04%
Dedup	1.02e+11	5.74e+07	0.06%
Swapti	2.43e+12	8.87e+08	0.04%
Canneal	1.19e+11	4.46e+07	0.04%
Freqmine	1.27e+12	4.78e+08	0.04%
Ferret	4.76e+11	7.04e+07	0.01%

The Table 1 show the standard deviation and what that correspond to the total number of instructions in percentage.

The same test was performed for the frequency, varying from 1.2 to 2.2 GHz with 100 MHz steps. The results are found in Table 2.

Table 2: Frequency Variation

Application	Instructions	Dev.	Dev. (%)
Vip	7.97e+11	1.16e+06	0.00%
Openmc	8.17e+07	4.52e+03	0.01%
Rtview	9.91e+12	6.64e+05	0.00%
X264	4.52e+11	1.54e+05	0.00%
Bodytrack	1.84e+12	2.54e+05	0.00%
Fluidanimate	2.38e+12	1.70e+09	0.07%
Xhpl	1.14e+08	5.95e+03	0.01%
Blackschole	3.75e+12	4.36e+05	0.00%
Dedup	1.02e+11	8.32e+07	0.08%
Swapti	2.43e+12	1.48e+05	0.00%
Canneal	1.19e+11	3.01e+05	0.00%
Freqmine	1.27e+12	3.70e+08	0.03%
Ferret	4.76e+11	5.63e+07	0.01%

Table 3: Variation of the number of instructions when changing the frequency for the same input.

These results show that all the assumptions were reasonable, and we can safely move to the validation of the model's prediction.

5.3. Case-Study Applications

The PARSEC parallel benchmark suite, version 3.0 [55], OpenMC [56] and LINPACK (HPL) [57], were chosen as case studies. The PARSEC benchmark focused on emerging workloads and was designed to represent the next-generation shared-memory programs for chip-multiprocessors. It covers an ample range of areas such as financial analysis, computer vision, engineering, enterprise storage, animation, similarity search, data mining, machine learning, and media processing. The OpenMC and the LINPACK are two classic HPC programs.

5.4. Fitting the Models

To find the parameters of the Eq. (22), 10 uniformly random configurations of frequencies (f), cores (p) and inputs (N) were chosen from the range $1 <= p <= 32$, $1.2 <= f <= 2.2$ and $1 <= N <= 5$ respectively. The application was executed for each chosen configuration, and the measured energy and time values were collected. For the input size, if we assume that all CPU instructions are executed at approximately the same time, the number of basic operations will be directly correlated with the time. Thus, we can estimate the input size by looking at the execution time, allowing us to divide a large input size into several smaller ones knowing their relationship as done in the work of Oliveira [48]. The unity can also vary depending on the definition. For simplicity, we assign numbers from 1 to 10, increasing the problem linearly, so it is also possible to interpolate any input in between these values.

For each configuration, samples of the power were collected using IPMI every 1 second. This sampling rate was chosen based on the magnitude of the mean run time of the applications, which are in the order of minutes. Therefore, this rate provides enough samples to measure average power. Additionally, timestamps and the total run time were collected. The total energy spent on each configuration is estimated by first interpolating the power samples using the first-order method and then integrating this function in the time.

The model's parameters are calculated by solving an optimization problem of finding the values that minimize the squared error of the prediction to the measured values using the non-linear least-squares method.

The python library scikit-learning was used to build the SVR model [58]. The SVR was trained using the same data used for parameters estimation of Eq. (22) with a grid search used to find the best kernel function and the best values for the hyper-parameters penalty for the wrong (C) and (γ). For this data, the best function was the Radial Base Function (RBF), and the hyper-parameters were $C = 10^4$ and $\gamma = 0.5$.

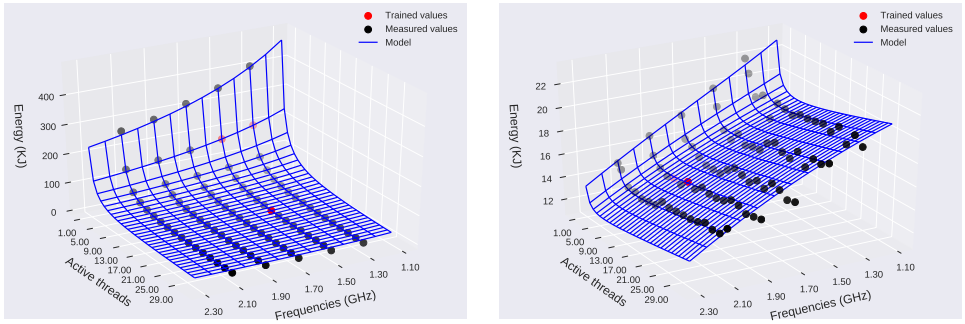
5.5. Measured versus Modeled Energy

To validate the model, we ran all possible configurations in the tested machine, varying the cores in a range of $1 <= p <= 32$, the frequency in $1.2 <= f <= 2.2$, and the input in $1 <= N <= 5$. The total number of configurations varies from 400 to over 1000 depending on the application, as some applications have restrictions on the number of cores that they can run. Once the data was collected, we computed the mean percentage error (MPE) according to the following equation:

$$MPE = \frac{1}{N}\sum_{i}^{N}\frac{|y_{\text{estimated}} - y_{\text{measured}}|}{y_{\text{measured}}}. \tag{23}$$

5.5.1. Frequency x Cores

The Figure 6 plot the measured and modeled energy consumption for some of the applications modeled. In addition, they show some of the possible shapes that the model can take while varying the number of active cores, operating frequency.

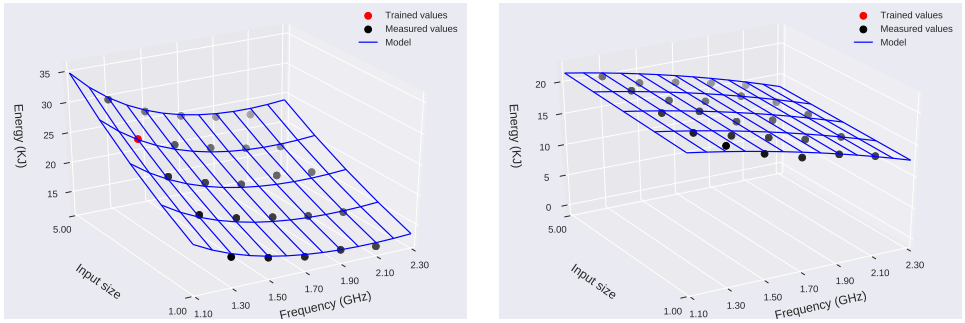


(a) Blackscholes. **(b)** Canneal.

Figure 6. Example fit for a specific input size: Blackscholes (a) and Canneal (b). "measured values" are the sensor data, and "minimum energy" is the minimum energy model prediction.

5.5.2. Frequency x Input

The Figure 7 plot the measured and modeled energy consumption for some of the applications modeled. They show some of the possible shapes that the model can take while varying the operating frequency, and input size.

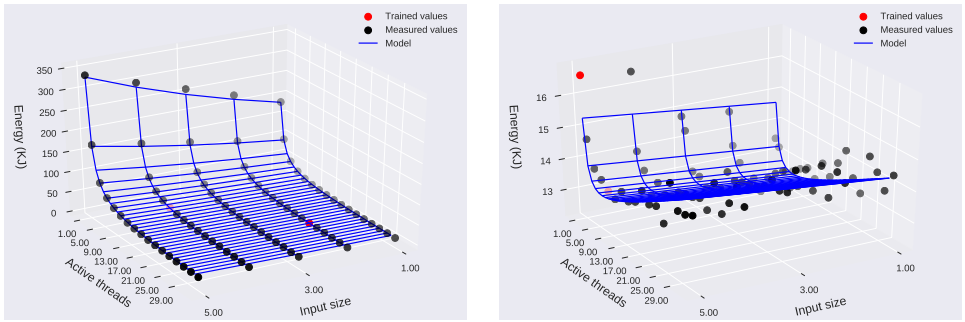


(a) Blackscholes. **(b)** Canneal.

Figure 7. Example fit for a specific input size: Blackscholes (a) and Canneal (b). "measured values" are the sensor data and "minimum energy" is the minimum energy model prediction.

5.5.3. Cores x Input

The Figure 8 plot the measured and modeled energy consumption for some of the applications modeled. They show some of the possible shapes that the model can take while varying the number of active cores, and input size.



(a) Blackscholes. **(b)** Canneal.

Figure 8. Example fit for a specific input size: Blackscholes (a) and Canneal (b). "measured values" are the sensor data and "minimum energy" is the minimum energy model prediction.

5.5.4. Validation

The average results for each application were calculated using a model trained with only 10 configurations, and the comparison is displayed Figure 9. We calculated the raw MPE values shown in Table 4.

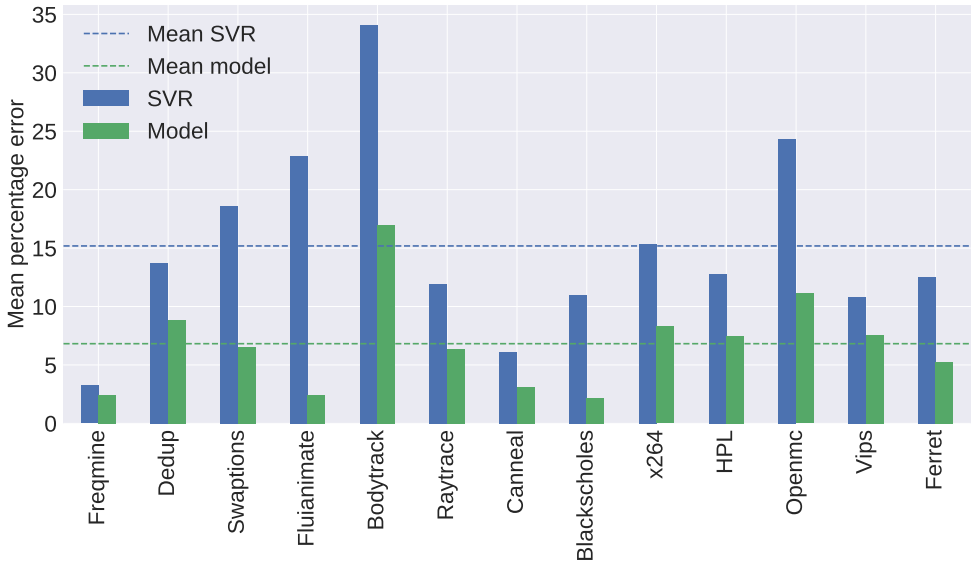


Figure 9. Comparison of the Mean Percentage Error between the proposed model and SVR. "Model mean" and "SVR mean" are the average of all MPE values for all applications.

Application	Model	SVR
Ferret	5.25	12.49
Raytrace	6.36	11.95
Fluianimate	2.44	22.90
x264	8.28	15.33
Vips	7.54	10.80
Swaptions	6.54	18.57
Canneal	3.12	6.13
Dedup	8.85	13.70
Freqmine	2.44	3.24
Blackscholes	2.18	11.00
HPL	7.47	12.75
Bodytrack	16.98	34.12
Openmc	11.15	24.34

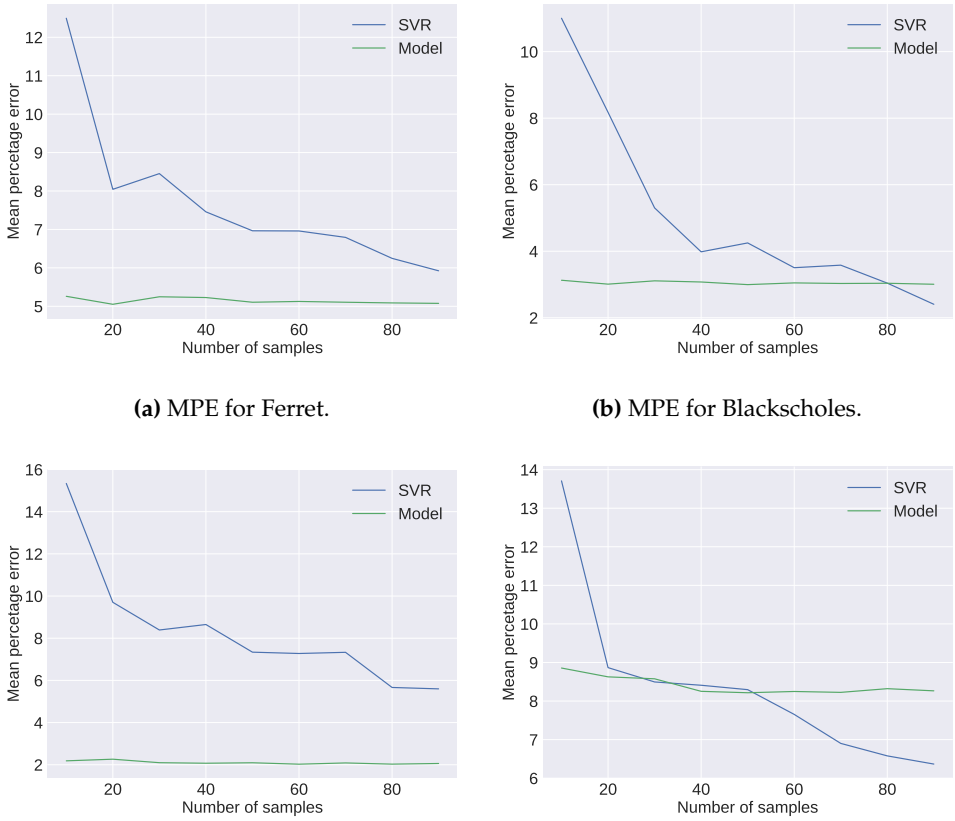
Table 4: Comparison of the Mean Percentage Error between the proposed model and SVR: raw values.

Figure 9 shows that the proposed model always performed better, with a lower MPE, than SVR when we were limited to 10 training points. This result is further explored in 5.6 where we compare with different training sizes.

5.6. Overheads on training

It is known that machine learning is data-driven, in that sense, the SVR model obtained using only 10 configurations could be improved, but what about the analytical model? To answer that question, the proposed model and the SVR were also trained with a varying number of configurations. We then compared the MPE and the amount of energy spent to create each model. This accuracy-energy trade-off is crucial since the

energy overhead when building models defeats the primary goal of saving power when running applications.



(a) MPE for Ferret.

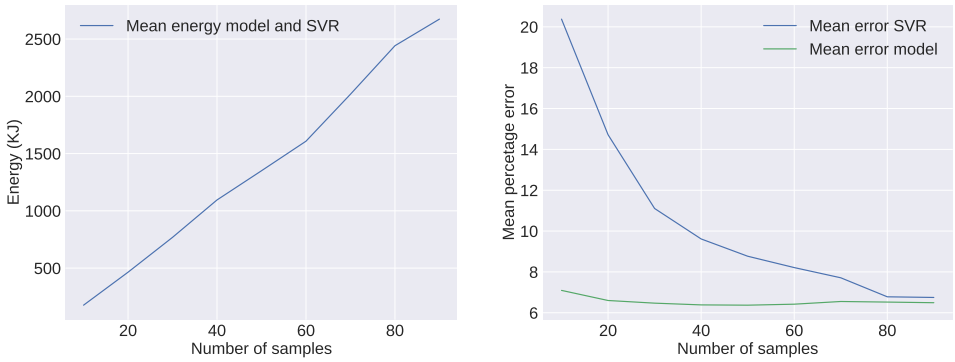
(b) MPE for Blackscholes.

(c) MPE for x264.

(d) MPE for Dedup.

Figure 10. MPE of the case studies versus training size, comparing how many training points is necessary to reach an acceptable result.

Figure 10 shows the comparisons of MPE and energy spent to create each model for two selected applications. According to the results, the analytical model is found to be very stable, not changing much as more data is added, while the SVR keeps reshaping to adapt to the data. The error of the analytical model is almost constant but that of the SVR, initially very high, drops as more data is used in the training process.



(a) Average energy spent on all applications during model creation. The two curves are identical because the same data were used to adjust the SVR and the model.

(b) MPE of all applications: SVR needs 10 times more data to have an MPE lower than the proposed model.

Figure 11. Overall results for energy and MPE for each training size.

Figure 11 presents the overall results, with the mean energy overhead and MPE for all applications. The meeting point of the MPE for the SVR and the proposed model can be extracted from Figure 11b. There it shows that in around 90 configurations, the SVR starts to have a smaller error. The cost of that is the linear increase in energy spent on training. The increase in energy, about 10 times more, can be observed in Figure 11a.

5.7. Analysis

One of the most significant advantages of using an analytical model is the understanding of the problem that an equation provides, making many different kinds of analysis possible that are otherwise impossible with a machine learning model. In this section, we discuss one of the possible analyses. In the following figures, we try to understand the contribution of each parameter of the equation to the total energy consumption.

For this analysis, we took the model of one of the applications and, varying one parameter of the equation, we display the energy versus performance (time) for all configurations. After that, we computed the Pareto frontier, a set of all Pareto efficient allocations, i.e., all the configurations where resources cannot be reallocated to make one individual better off without making at least one individual worse off. This gives us all the configurations where we have an optimal trade-off of performance and energy to choose from.

Figure 12 show the Pareto frontier for several values for the static power parameter (c_3 in the Eq. (22)) with configurations of frequency ranging from 1.2 to 5 GHz and cores from 1 to 64 so that we can also have an idea of what is the tendency when we increase the frequency and number of cores.

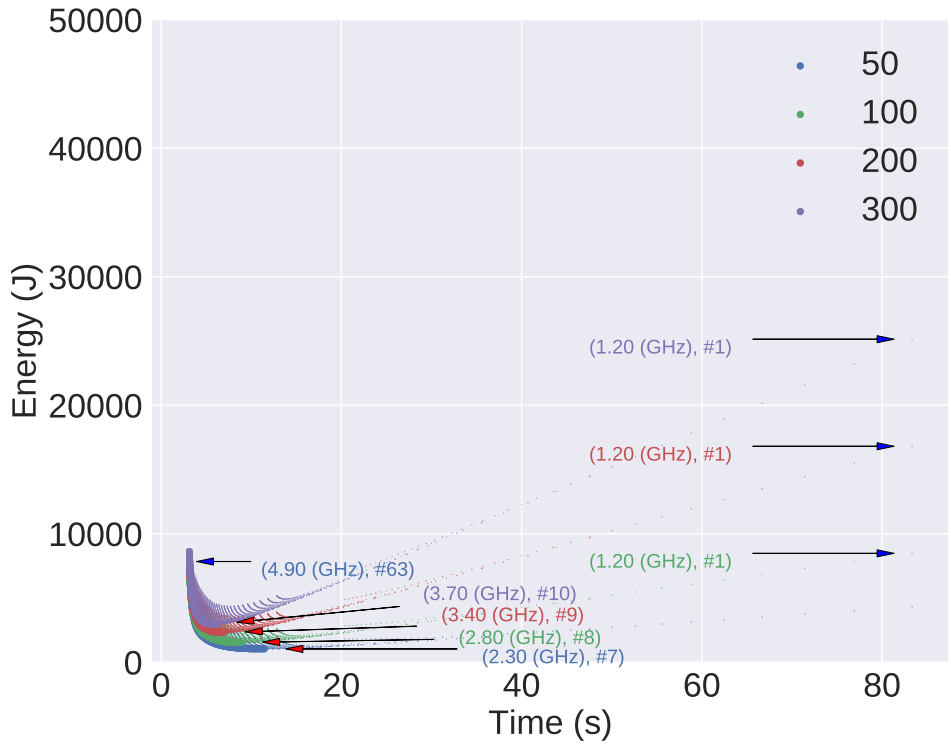


Figure 12. Pareto frontier for several values of static power parameter. The arrows with blue heads indicate the maximum energy, while the arrows with a red head the minimal for each corresponding curve.

From this figure, we can see that when increasing the value of the static power parameter, the total energy consumption increases as expected. We can also observe that the values that minimize the total energy consumption tend to be high frequency and multiple cores. This is one of the consequences of increasing the static power factor. As the dynamic factor proportionally decreases, the variables linked to it tend to have less impact on total consumption, enabling configurations with high frequency and several cores. This also enables chip-level optimization for choosing components that change the ratio between static and dynamic power.

Figure 13 shows the Pareto frontier in the same ranges described before but for the parameter corresponding to the level of parallelism of the application (w in the Eq. (22)).

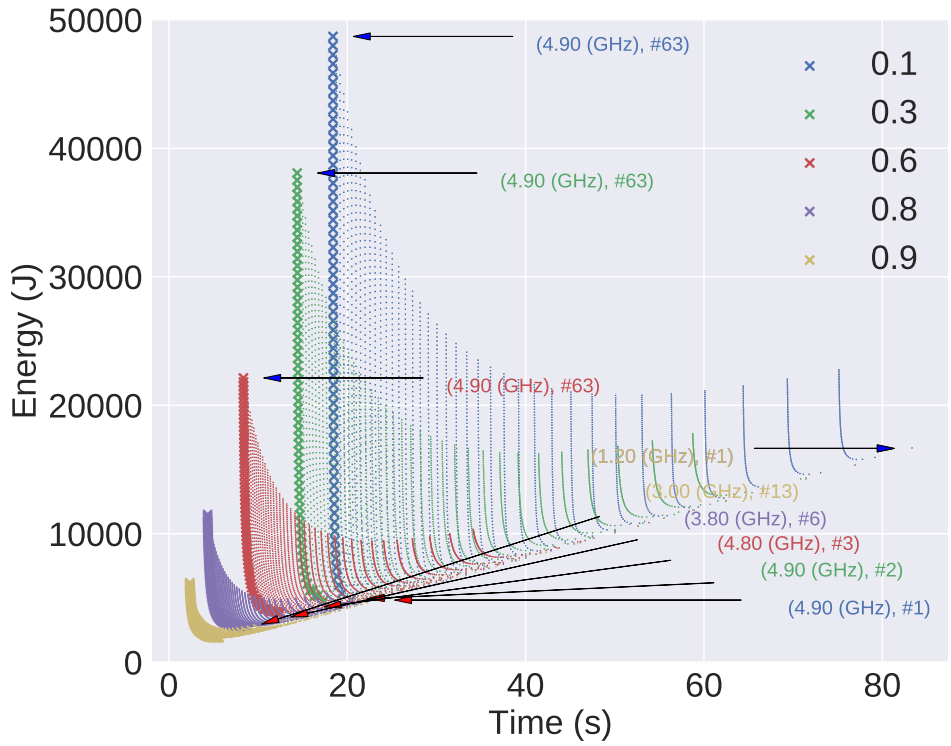


Figure 13. Pareto frontier for several values of static power parameter. The arrows with blue heads indicate the maximum energy, while the arrows with a red head, the minimal, for each corresponding curve.

In the Figure 13 we observe that as the parallelism level increases the total energy decreases. The number of cores tends to be higher with a higher level of parallelism as expected and the frequency shows an inverse relation.

5.8. DVFS and DPM optimization

The effectiveness of the proposed approach during optimization was evaluated with a simple algorithm that finds the optimal frequency and number of active cores from the proposed equation. The results were then compared to the Linux default choices for power management.

With Eq. (22), it is possible to calculate energy consumption estimates for each possible configuration since there is a finite range of possible values for the frequency and number of cores. It is also possible to apply constraints on the execution time, frequency, and number of active cores. Then, the configuration that minimizes energy consumption for a given input can be selected.

Current HPC managers leave to the user the choice of how many cores to use. On this basis, three situations were analyzed with relation to the number of cores:

1. Worst choice: number of cores that maximize the total energy consumed;
2. Random choice: energy consumed for a random choice of number of cores;
3. Best choice: number of cores that minimize the total energy consumed (oracle).

The default option for the Linux governor is Ondemand, and by default, it has no DPM control for the number of active cores. As Ondemand only does DVFS, for comparison, each application was executed with all available cores in the system, from 1 to 32.

Figures 14, 15, and 16, show the energy savings with respect to Ondemand, i.e. $\frac{Ondemand - Model_{min}}{Ondemand}$ for the three cases described above. The savings and losses for each case are:

1. Worst choice: save 69.88% on average;
2. Random choice: save 12.04% on average;
3. Best choice: lost 14.06% on average.

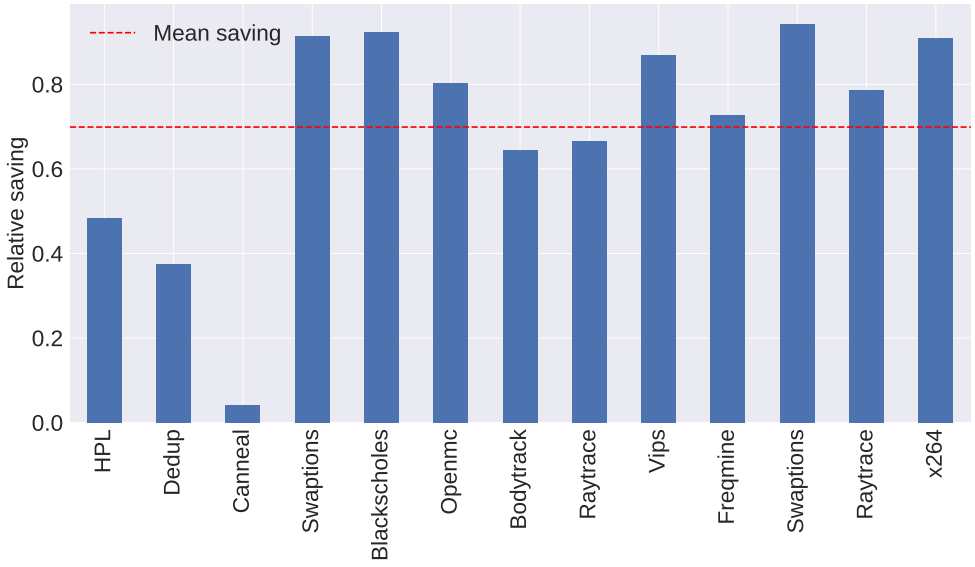


Figure 14. Energy savings comparisons between the proposed model and the Worst case.

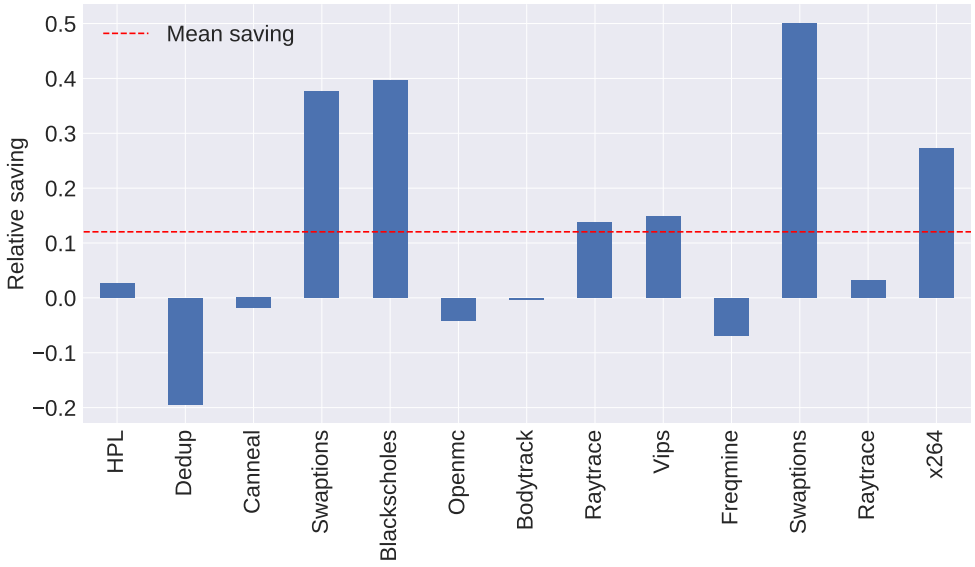


Figure 15. Energy savings comparisons between the proposed model and the Random case.

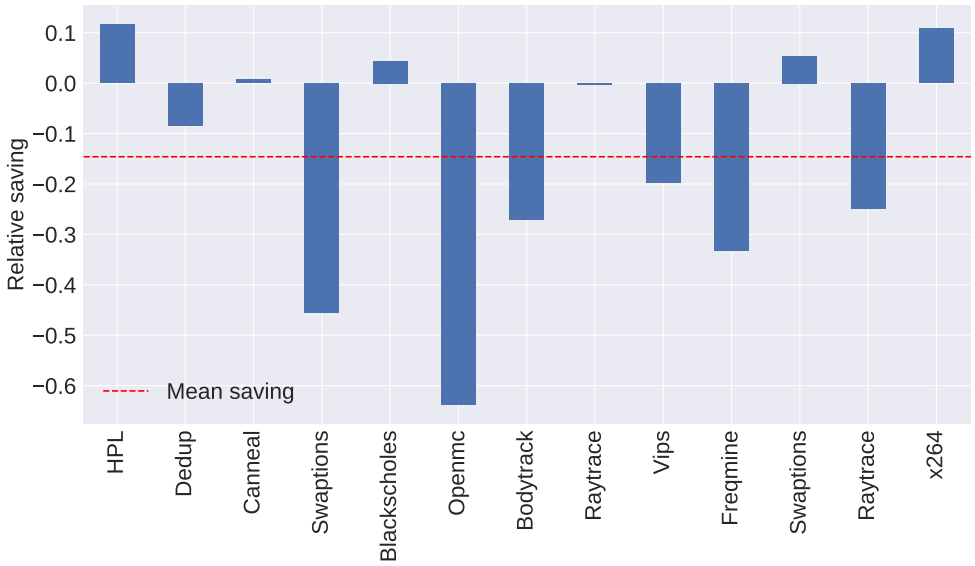


Figure 16. Energy savings comparisons between the proposed model and the Best case.

By default, operating systems do not implement DPM at the core level, and in HPC the user usually explicitly chooses the number of cores to run their job. To give a better idea of the impact on the energy consumption of DPM at the core level, we analyzed the choices of the number of cores over a period of one year in the HPC center at UFRN. The result is plotted in Figure 17.

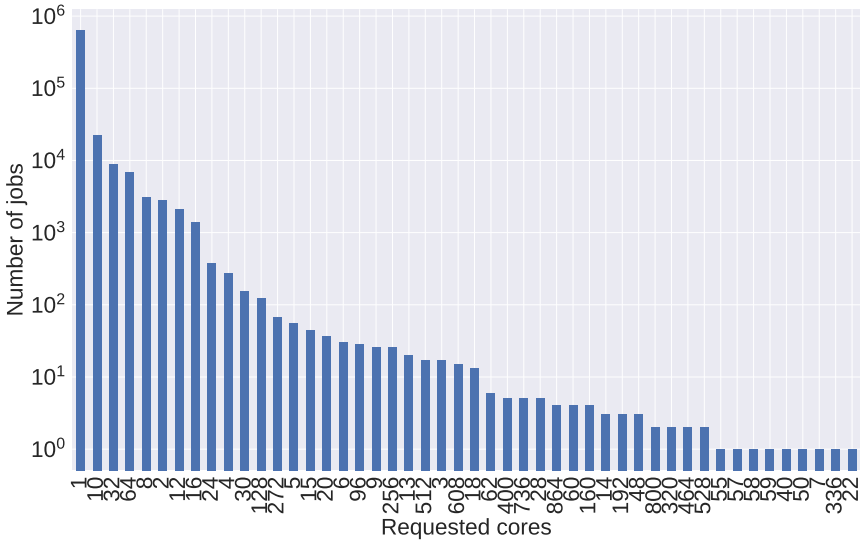


Figure 17. Number of CPU requests during one year in HPC cluster, sorted by the number of cores requested per job.

Observe that the most common choice of many regular users is a single core requested per job, matching the worst case choice for all application analysed in this work. The best choice was quite often 32 cores, which is the third most popular choice among users, but it is 72 times less frequent than 1 core. This leads us to envision how much energy could be saved and encourage us towards future work using the proposed model for DPM or more advanced optimization algorithms.

In practice, this approach can be brought into production by allowing the resource manager to perform these changes for the user using pre-scripts and post-scripts for high energy consumption job submissions.

6. Conclusion

We have proposed an energy model based on the operating frequency and the number of cores for a shared memory system in this work. This model serves as a reference for DVFS and DPM optimization problems.

Results from 3 different HPC benchmarks demonstrate the potential of the proposed model while consuming 10 times less energy than a machine learning approach, such as SVR, to characterize applications. Moreover, it can provide knowledgeable hints to improve DVFS and DPM algorithms by enabling analysis of the contribution of each model parameter (e.g., level of parallelism) to the energy consumption. Indeed, as shown in Section 5.8, when no oracle is available to choose the frequency and the number of cores the application should use, the proposed model can save around 12% of energy for a random choice and up to 70% for the worse possible choice. Considering the job history of our own HPC center, which shows the prevalence of worse choices made by users, the potential energy savings are very significant and encourages further research.

Future work will demonstrate all possible analyzes achievable with the proposed equation as more advanced DVFS models can be derived from it. For instance, identifying the different phases of a target program will allow more subtle changes in frequency and, perhaps, in the number of active cores to improve further the results presented here.

Author Contributions: All authors conceived and planned the experiments, V. R. G. Silva contributed to the implementation and test.

Acknowledgments: The experiments performed in this work, we used the compute nodes of the High-Performance Computing Center (NPAD/UFRN)[1].

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DVFS Dynamic Frequency and Voltage Scaling
DPM Dynamic Power Management
SVR Support Vector Regression
RBF Radial Base Function
HPC High Performance Computing
IPMI The Intelligent Platform Management Interface
RBF Radial Base Function
MPE Mean Percentage Error

References

1. Group, F. *Handbook of Energy-Aware and Green Computing Series Editor : Sartaj Sahni*; Vol. 1, 2012.
2. Dayarathna, M.; Wen, Y.; Fan, R. Data center energy consumption modeling: A survey. *IEEE Communications Surveys and Tutorials* **2016**, *18*, 732–794. doi:10.1109/COMST.2015.2481183.
3. Corcoran, P.; Andrae, A. Emerging Trends in Electricity Consumption for Consumer ICT **2017**. pp. 1–56.
4. Mathew, V.; Sitaraman, R.K.; Shenoy, P. Energy-aware load balancing in content delivery networks. *Proceedings - IEEE INFOCOM* **2012**, pp. 954–962. doi:10.1109/INFCOM.2012.6195846.
5. Rivoire, S.; Shah, M.A.; Ranganathan, P.; Kozyrakis, C.; Meza, J. Models and metrics to enable energy-efficiency optimizations. *Computer* **2007**, *40*, 39–40. doi:10.1109/MC.2007.436.

[1] NPAD is the name of the High Performance Computing Center (from Portuguese: Núcleo de Processamento de Alto Desempenho) of the Universidade Federal do Rio Grande do Norte (UFRN)

6. Buyya, R.; Vecchiola, C.; Selvi, S.T. *Mastering Cloud Com- puting: Foundations and Applications Programming*; 2013; pp. 3–27. doi:10.1016/b978-0-12-411454-8.00001-2.
7. Poess, M.; Nambiar, R.O. Energy cost, the key challenge of today's data centers: A power consumption analysis of TPC-C results. *Proceedings of the VLDB Endowment* **2008**, *1*, 1229–1240. doi:10.14778/1454159.1454162.
8. Gao, Y.; Guan, H.; Qi, Z.; Wang, B.; Liu, L. Quality of service aware power management for virtualized data centers. *Journal of Systems Architecture* **2013**, *59*, 245–259. doi:10.1016/j.sysarc.2013.03.007.
9. Fan, X.; Weber, W.D.; Barroso, L.A. Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News* **2007**, *35*, 13. doi:10.1145/1273440.1250665.
10. Barroso, L.A.; Hölzle, U. The case for energy-proportional computing. *Computer* **2007**, *40*, 33–37. doi:10.1109/MC.2007.443.
11. Malladi, K.T.; Nothaft, F.A.; Periyathambi, K.; Lee, B.C.; Kozyrakis, C.; Horowitz, M. Towards energy-proportional datacenter memory with mobile DRAM. *Proceedings - International Symposium on Computer Architecture* **2012**, pp. 37–48. doi:10.1109/ISCA.2012.6237004.
12. Rotem, E.; Naveh, A.; Ananthakrishnan, A.; Weissmann, E.; Rajwan, D. Power-management architecture of the intel microarchitecture code-named Sandy Bridge. *IEEE Micro* **2012**, *32*, 20–27. doi:10.1109/MM.2012.12.
13. Brown, L.; Moore, R.; Li, D.S.; Yu, L.; Keshavamurthy, A.; Pallipadi, V. ACPI in Linux. *Symposium A Quarterly Journal In Modern Foreign Literatures* **2005**, *51*, 51.
14. Hackenberg, D.; Schöne, R.; Ilsche, T.; Molka, D.; Schuchart, J.; Geyer, R. An Energy Efficiency Feature Survey of the Intel Haswell Processor. *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2015* **2015**, pp. 896–904. doi:10.1109/IPDPSW.2015.70.
15. Shuja, J.; Madani, S.A.; Bilal, K.; Hayat, K.; Khan, S.U.; Sarwar, S. Energy-efficient data centers. *Computing* **2012**, *94*, 973–994. doi:10.1007/s00607-012-0211-2.
16. Benini, L.; Bogliolo, A.; De Micheli, G. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **2000**, *8*, 299–316. doi:10.1109/92.845896.
17. Hypothesis, M.I. *Encyclopedia of the Sciences of Learning*; Number 1988, 2012; pp. 223–242. doi:10.1007/978-1-4419-1428-6.
18. Roy, P.; Mahapatra, G.S.; Dey, K.N. Forecasting of software reliability using neighborhood fuzzy particle swarm optimization based novel neural network. *IEEE/CAA Journal of Automatica Sinica* **2019**, *6*, 1365–1383. doi:10.1109/JAS.2019.1911753.
19. Zhu, W.; Liu, X.; Xu, M.; Wu, H. Predicting the results of RNA molecular specific hybridization using machine learning. *IEEE/CAA Journal of Automatica Sinica* **2019**, *6*, 1384–1396. doi:10.1109/JAS.2019.1911756.
20. Rivoire, S.; Ranganathan, P.; Kozyrakis, C. A comparison of high-level full-system power models. *Conference on Power aware computing and systems (HotPower 2008)* **2008**, pp. 1–5.
21. Sarwar, a. Cmos power consumption and cpd calculation. *Proceeding: Design Considerations for Logic Products* **1997**. doi:SCAA035B.
22. Butzen, P.; Ribas, R. Leakage Current in Sub-Micrometer CMOS Gates. *Universidade Federal do Rio Grande do Sul* **2007**, pp. 1–30.
23. Usman, S.; Khan, S.U.; Khan, S. A comparative study of voltage/frequency scaling in NoC. *IEEE International Conference on Electro Information Technology* **2013**. doi:10.1109/EIT.2013.6632716.
24. Paolillo, A. Optimisation of Performance Metrics of Embedded Hard Real-Time Systems using Software / Hardware Parallelism **2018**.
25. Kim, D.H.K.; Hoffmann, H. Racing and Pacing to Idle: Theoretical and Empirical Analysis of Energy Optimization Heuristics. *International Conference on Cyber-Physical Systems, Networks, and Applications* **2015**, pp. 21–23.
26. Fu, C.; Chau, V.; Li, M.; Xue, C.J. Race to idle or not: balancing the memory sleep time with DVS for energy minimization. *Journal of Combinatorial Optimization* **2018**, *35*, 860–894. doi:10.1007/s10878-017-0229-7.
27. Merkel, A.; Bellosa, F. Balancing power consumption in multiprocessor systems. *Proceedings of the 2006 EuroSys Conference* **2006**, pp. 403–414. doi:10.1145/1217935.1217974.
28. Roy, S.; Rudra, A.; Verma, A. An energy complexity model for algorithms. *ITCS 2013 - Proceedings of the 2013 ACM Conference on Innovations in Theoretical Computer Science* **2013**, pp. 283–303. doi:10.1145/2422436.2422470.
29. Weaver, V.M.; McKee, S.A. Can hardware performance counters be trusted? *2008 IEEE International Symposium on Workload Characterization, IISWC'08* **2008**, pp. 141–150. doi:10.1109/IISWC.2008.4636099.
30. Weaver, V.M.; Terpstra, D.; Moore, S. Non-determinism and overcount on modern hardware performance counter implementations. *ISPASS 2013 - IEEE International Symposium on Performance Analysis of Systems and Software* **2013**, pp. 215–224. doi:10.1109/ISPASS.2013.6557172.
31. Das, S.; Werner, J.; Antonakakis, M.; Polychronakis, M.; Monrose, F. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. *Proceedings - IEEE Symposium on Security and Privacy* **2019**. doi:10.1109/ESSDERC.2003.1256886.
32. Mc Guire, N.; Okech, P.; Schiesser, G. Analysis of Inherent Randomness of the Linux Kernel. *Eleventh RealTime Linux Workshop* **2009**.
33. Ramos, V.; Valderrama, C.; Xavier De Souza, S.; Manneback, P. An accurate tool for modeling, fingerprinting, comparison, and clustering of parallel applications based on performance counters. *Proceedings - 2019 IEEE 33rd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019* **2019**, pp. 797–804. doi:10.1109/IPDPSW.2019.00130.
34. Silva-De-Souza, W.; Iranfar, A.; Bráulio, A.; Zapater, M.; Xavier-De-Souza, S.; Olcoz, K.; Atienza, D. Containergy-a container-based energy and performance profiling tool for next generation workloads. *Energies* **2020**, *13*, 1–19. doi:10.3390/en13092162.

35. Characterization, E.; Model, E. Energy characterization and instruction-level energy model of Intel's Xeon Phi processor **2013**. pp. 1–6.
36. Lewis, A.; Ghosh, S.; Tzeng, N.F. Run-time energy consumption estimation based on workload in server systems. *Proceedings of the 2008 conference on Power aware computing and systems* **2008**, pp. 4–4.
37. Mills, B.; Znati, T.; Melhem, R.; Ferreira, K.B.; Grant, R.E. Energy consumption of resilience mechanisms in large scale systems. *Proceedings - 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014* **2014**, pp. 528–535. doi:10.1109/PDP.2014.111.
38. Liu, X.; Han, T.; Ansari, N. Intelligent battery management for cellular networks with hybrid energy supplies. *IEEE Wireless Communications and Networking Conference, WCNC* **2016**, *2016-Septe*. doi:10.1109/WCNC.2016.7564759.
39. Beckman, D.; Hirsch, D. Making a case for efficient supercomputing. *ABA Journal* **2005**, *91*.
40. Amdahl, G.M. Validity of the single processor approach to achieving large scale computing capabilities. Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring); ACM Press: New York, New York, USA, 1967; p. 483. doi:10.1145/1465482.1465560.
41. Eyerman, S.; Eeckhout, L. Modeling critical sections in Amdahl's law and its implications for multicore design. *Proceedings - International Symposium on Computer Architecture* **2010**, pp. 362–370. doi:10.1145/1815961.1816011.
42. Shi, Y. Reevaluating Amdahl's Law and Gustafson's Law **2015**.
43. Rauber, T.; Rünger, G.; Schwind, M.; Xu, H.; Melzner, S. (P1) Energy measurement, modeling, and prediction for processors with frequency scaling. *The Journal of Supercomputing* **2014**, *70*, 1451–1476. doi:10.1007/s11227-014-1236-4.
44. Goel, B.; McKee, S.A. A Methodology for Modeling Dynamic and Static Power Consumption for Multicore Processors. *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* **2016**, pp. 273–282. doi:10.1109/IPDPS.2016.118.
45. Du, Z.; Ge, R.; Lee, V.W.; Vuduc, R.; Bader, D.A.; He, L. Modeling the Power Variability of Core Speed Scaling on Homogeneous Multicore Systems. *Hindawi Scientific Programming* **2017**, p. 13. doi:10.1155/2017/8686971.
46. Gonzalez, R.; Gordon, B.; Horowitz, M. Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of Solid-State Circuits* **1997**, *32*, 1210–1216. doi:10.1109/4.604077.
47. Kumar, V.; Gupta, A. Analyzing Scalability of Parallel Algorithms and Architectures, 1994. doi:10.1006/jpdc.1994.1099.
48. Oliveira, V.H.; Georgiou, K.; Furtunato, A.F.; Eder, K.; Silveira, L.F.; Xavier-De-Souza, S. Application speedup characterization: Modeling parallelization overhead and variations of problem size and number of cores. *ICPE 2018 - Companion of the 2018 ACM/SPEC International Conference on Performance Engineering* **2018**, *2018-Janua*, 43–44. doi:10.1145/3185768.3185770.
49. Smola, A.J.; Schölkopf, B. A Tutorial on Support Vector Regression. *Statistics and Computing* **2004**, *14*, 199–222. doi:Doi 10.1023/B:Stco.0000035301.49549.88.
50. Kitts, B. Regression Trees Lecture. *Data Mining* **2006**, p. 7.
51. Altman, N.S. An introduction to kernel and nearest-neighbor nonparametric regression. *American Statistician* **1992**, *46*, 175–185. doi:10.1080/00031305.1992.10475879.
52. Murtagh, F. Multilayer perceptrons for classification and regression. *Neurocomputing* **1991**, *2*, 183–197. doi:10.1016/0925-2312(91)90023-5.
53. Gao, S.; Zhou, M.; Wang, Y.; Cheng, J.; Yachi, H.; Wang, J. Dendritic Neuron Model with Effective Learning Algorithms for Classification, Approximation, and Prediction. *IEEE Transactions on Neural Networks and Learning Systems* **2019**, *30*, 601–614. doi:10.1109/TNNLS.2018.2846646.
54. Schwenkler, T. Intelligent Platform Management Interface **2006**. pp. 169–207. doi:10.1007/3-540-31287-0{_}6.
55. Bienia, C.; Kumar, S.; Singh, J.P.; Li, K. The PARSEC benchmark suite: Characterization and architectural implications. *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques* **2008**, pp. 72–81. doi:10.1145/1454115.1454128.
56. Romano, P.K.; Horelik, N.E.; Herman, B.R.; Nelson, A.G.; Forget, B.; Smith, K. OpenMC: A state-of-the-art Monte Carlo code for research and development. *Annals of Nuclear Energy* **2015**, *82*, 90–97. doi:10.1016/j.anucene.2014.07.048.
57. Dongarra, J.J. The LINPACK Benchmark: An explanation, 1988. doi:10.1007/3-540-18991-2{_}27.
58. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; .; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; .; Weiss, R.; Dubourg, V.; V.; erplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.