Effectiveness of Neural Networks for Power Modeling for Cloud and HPC: It's Worth It!

GEORGES DA COSTA, JEAN-MARC PIERSON, and
LEANDRO FONTOURA-CUPERTINO, IRIT, University of Toulouse

Power consumption of servers and applications are of utmost importance as computers are becoming ubiquitous, from smart phones to IoT and full-fledged computers. To optimize their power consumption, knowledge is necessary during execution at different levels: for the Operating System to take decisions of scheduling, for users to choose between different applications.

Several models exist to evaluate the power consumption of computers without relying on actual wattmeters: Indeed, these hardware are costly but also usually have limits on their pooling frequency (usually a one-second frequency is observed) except for dedicated professional hardware. The models link applica-tions behavior with their power consumption, but up to now there is a 5% wall: Most models cannot reduce their error under this threshold and are usually linked to a particular hardware configuration.

This article demonstrates how to break the 5% wall of power models. It shows that by using neural networks it is possible to create models with 1% to 2% error. It also quantifies the reachable precision obtainable with other classical methods such as analytical models.

Additional Key Words: Power models, machine learning, analytical models, Cloud, HPC

1 INTRODUCTION

In the Green IT community there is a very large number of researchers proposing approaches for estimating power. Power consumption knowledge is crucial in many aspects of the life of a computer infrastructure: It allows system administrator to make sure their computers will not exceed the maximum power fed from the energy provider; it allows infrastructure providers to

The results presented in this article are partially funded by the European Commission under contract 288701 through the project CoolEmAll and by the French ANR project DATAZERO (ANR-15-CE25-0012).
Authors' addresses: G. Da Costa, J.-M. Pierson, and L. Fontoura-Cupertino, IRIT, University of Toulouse; emails: {georges.da-costa, jean-marc.pierson, leandro.fontoura-cupertino}@irit.fr.

https://doi.org/10.1145/3388322

bill for the actual consumption of the hosted customers and these latter to be conscious of their power consumption; it can allow autonomous systems to adapt the placement of services [27, 39] as a matter of these estimates to avoid creating hot spots in the infrastructure or to optimize the power consumption of the diverse equipment at hand (favoring the most energy-efficient ones, for instance); it also provides means for users to choose the less-energy-consuming application and the developers to choose the most energy-efficient code library.

To know the power consumption of a system, there are basically two techniques: Either having power meters and monitoring the power consumption regularly or building a model for estimating the power on the fly. The first solution needs the installation and management of a monitoring infrastructure on every single computer; the second creates a generic model and instantiates it on all equipment. However, the quality of the estimation finally depends on the quality of the model. The main trend of approaches is based on analytical models, where all components are known in advance and the contribution of each to the power consumption is weighted (manually calibrated or via linear regression). Knowing in advance the internal architectures allows for good precision but for general workloads, any error on the power estimation below a 5% wall is unseen. While this precision can be considered enough in some cases, better precision is required to make smart and profitable decisions.

Another approach can give better results in the general case, i.e., better precision, and with a higher level of adaptation to unknown architectures. In formula-learned models, the components contributing to the power consumption are unknown. Some early works [16, 40] showed the interests of such approach, but in limited environment and configuration. Using properly artificial neural networks to build prediction models is a common technique: We propose a detailed methodology integrating proper data curation and neural network hyper-parameters selection. The methodology by itself is actually a contribution allowing other researchers to reproduce the same experimental protocol and therefore to improve the reproducibility of the results. We applied it to the estimation of power consumption at runtime. In the experiment section, we focus on high-density servers with low power consumption. The reason is twofold: First, there is a trend in investigating the possibilities of using such systems in different environments, from Cloud to HPC [25, 30, 44]. Second, being able to model the power consumption of low-power servers with high precision is more difficult than with high consumption servers (i.e., a 5% error on 300 watts is not the same as on 100 watts: the former is precise at 15 watts, while the latter is precise at 5 watts). However, the method is not limited to this use case and could have been applied on different types of hardware. To the best of our knowledge, this work is the first to use neural networks to estimate power consumption at runtime [38, 46].

To sum up, the contributions of this article are:

- A comprehensive state-of-the-art on power modeling, highlighting pros and cons of the different approaches.
- A validated in-depth methodology for power modeling based on four steps using artificial neural networks (preprocessing, learning dataset, topology optimization, variable reduction).
- An in-depth performance evaluation of analytical and machine-learning power models.

The rest of the article is organized as follows: Section 2 provides a comprehensive state-of-the-art on power modeling. Section 3 proposes basic preprocessing for learning data. Section 4 introduces our approach with artificial neural networks for power modeling. Section 5 details the experimentation. Section 6 validates and discusses the results. Section 7 concludes the article and give perspectives.

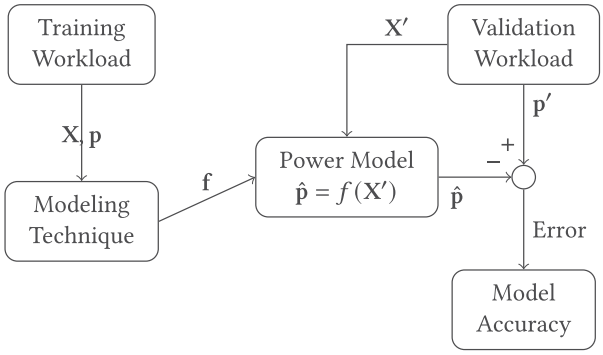


Fig. 1. Power model creation workflow.

2 RELATED WORKS IN POWER MODELING

Power models allow a better understanding of the energy consumption on computing systems. The creation of such models requires the execution and monitoring of different workloads. Figure 1 presents the workflow for creating a model. First, a training workload provides observations of the inputs $\mathbf{X}$ and power targets $\mathbf{p}$ to the modeling methodology to generate a power estimator $f(\mathbf{X})$. Then, the obtained model needs to be validated against a new workload that has not been used during the model creation. The validation workload provides new inputs ($\mathbf{X}'$) and targets ($\mathbf{p}'$) that are used to estimate the power for each input and to compute the error of the estimation, providing the accuracy of the model.

According to a Reference [43], this field of research on models has been particularly active until the end of 2014. Most recent works are focusing on new hardware architecture such as GPU instead of focusing on new approaches. It also shows that recent advances are focused on tools and usage of power models.

The models differ according to numerous aspects. In this section a thorough review on the state-of-the-art on power consumption estimation and modeling is done. The method for the evaluation of the proposed models and their accuracies are given when available. The literature review is organized in two parts: (iii) models that are predefined, in the sense that the variables being in the power model are known (the analytical power model formula is known, but possibly not the respective weights of the variables); (ii) models that do not *a priori* choose neither the variables, nor their weights (the power model formula is unknown). Finally, a discussion on the existing models is done by summarizing and comparing them based on characteristics such as granularity, accuracy, portability, simplicity, type of power meter used during the creation/evaluation of the model, and others. Each of these characteristics will be detailed in Section 2.4.

Several works [60, 66] use the source code of applications to evaluate their power consumption. As the focus of this article is on unknown applications, these approaches will not be evaluated in the following.

Similarly, the present work focuses on system-wide power models and will exclude hardware-specific models. Several works propose models related to a particular processor, as this is considered as the element consuming the largest amount of dynamic power. Gschwandtner et al. [26] propose a Power7 model leveraging the detailed knowledge of the processor architecture for the choice of the performance counters. The results show that under the limit of using a particular compiler (GCC), the power model error for the sole processor is between 1% and 5% for several HPC benchmarks.

Finally, the model described in the following are application-independent. To improve power models, one method consists in creating application- and hardware-specific models.

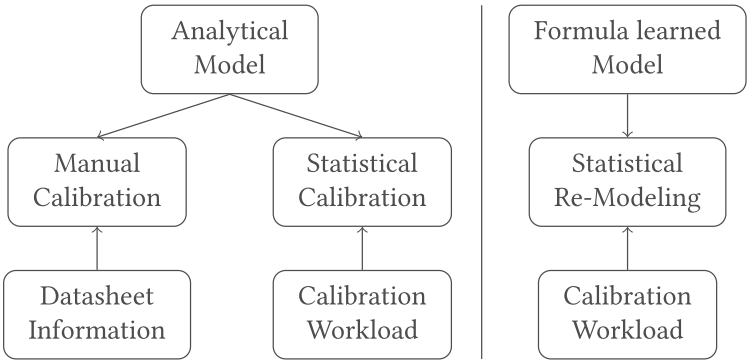


Fig. 2. Portability of power models.

Tiwari et al. [57] propose a methodology to create dedicated power models for several classical HPC kernels (such as matrix multiplication or LU) and obtain an error of 5.5%.

2.1 Analytical Models vs. Formula-learned Models

Modeling techniques can be grouped into detailed analytic models and high-level machine learning models. Analytic models are highly accurate, requiring *a priori* information from experts' knowledge. Thus, their portability is limited, although they can still use training data to tune themselves. However, machine learning models are statistical models that extract knowledge directly from the dataset, creating an entire model from scratch automatically.

Figure 2 compares both approaches with respect to their portability. Analytical models can either require information from an expert to provide inputs from hardware data-sheets or execute a calibration workload to automatically parameterize the model—usually a linear regression is used. Formula-learned models are created automatically; the simple execution of a workload can create a model for a new target machine. This makes them more portable than analytical ones.

2.2 Analytical Models

Predefined models are based on *a priori* knowledge of the most important and impacting characteristics of a computer to its power consumption. In other words, the Graal formula linking the usage of computer's resources to its power consumption is considered known.

2.2.1 Hardware-dependent Models. Intel first introduced in 2012 a power management architecture embedded on their Sandy Bridge microprocessors to enable the Turbo Boost technology to change cores' frequencies, respecting processor's thermal design power constraints [47]. The power management architecture predicts processor's power usage based on an analytical model. This model collects a set of architectural events from each core, the processor graphics, and I/O, and combines them together with energy to predict package's power consumption. Leakage power is estimated based on the voltage and temperature. The authors claim that the actual and reported power correlate accurately, but no in-depth information regarding neither the model, nor the accuracy is given. Intel also made available an interface—namely, Running Average Power Limit (RAPL), which allows the end-user to access power and energy measurements on different granularity. Energy measurements of processor's package, core, and DRAM sockets are available through Model Specific Registers (MSR). RAPL is also available in Linux's kernel mainline through the `perf` tool since 2013 [20]. Similar approaches can be found in recent AMD processors, which can report "Current Power in Watts" [1] while Nvidia GPUs can report power usage of the entire board via the Nvidia Management Library (NVML) [42]. Some libraries and software work as wrappers, simplifying the access for such embedded meters. The Performance API (PAPI) traditionally provides

low-level cross-platform access to Performance Monitoring Counters (PMC) available on modern processors. In Reference [62], the authors extended PAPI to collect power consumption of Intel processors and NVidia's GPUs via RAPL and NVML, respectively. In both cases, power metrics can only be retrieved at system-level and are based on the unknown analytical models given by the vendors. Note, however, that since the Haswell processor family, Intel RAPL is reporting actual power measurements and not any more estimates based on power models. Finally, note that these models only provide information related to the power consumption of processor, memory, and GPU, which is insufficient to obtain the power consumption of a whole computer.

2.2.2 Architectural-level Simulators. Low-level architectural events in a simulation system provide precise measurements with drawbacks on speed and portability. Such simulators can estimate the power consumption in different levels of granularity from transistors, through circuits, to instructions. Although important in early stages of the design cycle, the high execution time turns them into an offline method, being unfeasible to provide runtime estimations. In addition, low-level simulators are hardware-specific, which reduces their portability.

Gate-level simulators emulate components such as logic gates, multiplexers, or ALUs. CACOPS [6] is a cycle-accurate simulator that estimates the power consumed by an embedded system. The methodology used to implement the power models is based on the number of gates switching in the component when it is activated in a given cycle. The capacitance of each gate needs to be provided by the user, requiring in-depth knowledge of the hardware and circuitry.

In circuit-level simulators, the emulation occurs at a coarser grain. Wattch [12] is a processor power model that tries to accurately model the energy consumed by array structures, wires, and clocking. Each structure is modeled based on the dynamic power consumption (P_d) in CMOS microprocessors, defined as follows:

$$P_d = \alpha C V_{dd}^2 f,$$

where C is the load capacitance, V_{dd} is the supply voltage, and f is the clock frequency. The activity factor, α, is a fraction between 0 and 1 indicating how often clock ticks lead to switching activity on average. Similar approaches have been applied to memory [45], disk [65], and networking [61]. SimplePower [59] is a full-system cycle-level simulator that captures the energy consumed by a five-stage pipeline instruction set architecture, the memory system, and the buses.

Other simulators work at the instruction-level, such as Mambo [48], which is an IBM proprietary full-system simulation tool-set for the PowerPC architecture that includes a power estimator. The simulator uses the average power consumption of each instruction to provide the overall consumption of an application. The authors forced an instruction to run and measured its power consumption using a mainboard instrumented with a very accurate power meter. Although dealing with a single-core, architecture-specific estimator, the reported errors varied from −11.3 to 6.6%, while the average error was 4.1%. Similar approaches can be found for other architectures as well [3, 35].

Recently, generic low-level simulators such as McPAT [36] made possible the creation of power-aware simulators for GPU systems. In Reference [37], authors propose a power simulator for GPU that obtains an error of 8%–13% on numerous HPC workloads.

2.2.3 A Priori Analytical Models. The most basic approaches consider that the weights appearing in the power model are also known.

The first attempt to model the power consumption of applications at the process-level was done in 1999 by Flinn and Satyanarayanan with PowerScope [21]. The authors proposed a methodology to map the energy consumption to program structure by profiling the power and application's performance indicators. Basically, it decouples the total energy based on the running time of each

application. The profiled computer needs kernel modifications and can only handle single processor. The energy of each process is estimated first by synchronizing the power with the PID of the running process and then by integrating each PID's power over time as follows:

$$E \approx V_{meas} \sum_{t=0}^{n} I_t \Delta t,$$

where I_t is the current measured at regular intervals of time Δt, while the voltage V_{meas} is considered to be constant. `PowerScope` considers only single-core processors, making it inapplicable to recent processors.

Do et al. [17] proposed pTop, a top-like tool to monitor the power consumption of each process using only OS information. The underlying power model aggregates the consumption of three devices (processor, disk, and network) to measure the full-system power consumption of a process. The application's energy consumption is estimated indirectly through its resource utilization (u). The energy consumed by an application i is the sum of the energy of each resource j plus the energy of interaction with the system. For a time interval t, application and resource energy are computed as follows:

$$E_{appi} = \sum_{j} u_{ij} E_{resourcej} + E_{interaction},$$

with, for each device, a dedicated model:

$$E_{CPU} = \sum_{f \in F} P_f t_f + \sum_{k} n_k E_k,$$
$$E_{Neti} = t_{sendi} P_{send} + t_{recvi} P_{recv},$$
$$E_{Diski} = t_{readi} P_{read} + t_{writei} P_{write},$$

where E_{CPU} is the energy spent by the processor, P_f and t_f are the power and time spent by the processor in a given frequency (F available frequencies), n_k is the number of times a transition k occurs, and E_k is the energy spent on such transition. The energy for the network and disk are measured for each process i based on each device state's (send/receive packages, read/write from/to disk) power P and elapsed time t. The model was validated using three benchmarks: a sorting algorithm, a downloader, and an image viewer. The reported model accuracy is of 2 W maximum in a 3 to 15 W range (13% to 66%). Later, the authors extended the implementation to Windows OS, including a memory power model [14]. The reported accuracy for Media Player and Internet Explorer benchmarks shows a good approximation for processor and disk power, although pTopW presents bad estimations for the networking, showing 4 W error in a 7 W range (57%).

PowerAPI [41] is a process-level power estimator library that uses analytical models for processor and network card. Machine power consumption is estimated by summing the devices' power, as follows:

$$P_{sys} = \sum_{pid \in PID} P_{cpu}^{pid} + P_{nic}^{pid},$$
$$P_{cpu}^{pid} = \frac{\sum_{f \in F} P_{cpu}^{pid,f} \times t_{cpu}^{f}}{t_{cpu}},$$
$$P_{cpu}^{pid,f} = cfV^2 \times \frac{t_{cpu}^{pid}}{t_{cpu}}, \qquad c = \frac{0.7 \times TPD}{f_{TPD} V_{TPD}^2},$$
$$P_{nic}^{pid} = \frac{\sum_{i \in S} t_i \times P_i \times d}{t_{total}},$$

where c, f, and V are, respectively, the processor's capacitance, frequency, and voltage; F is the set of available processor's frequencies; f_{TPD} and V_{TPD} are the frequency and voltage at the thermal design power; PID is the set of all running processors in the system; S is a set of network states; P_i is the power consumed by the card in state i (provided by manufacturers). PIDs' usage always sums up to 1. The authors do not expose the number of states used to model their NIC. The evaluation workload is composed of `stress` and MPlayer benchmarks with no frequency changes. Linux's `stress` command is set to run concurrently in 1, 2, 3, and 4 cores, while MPlayer video execution is single threaded. Although the proposed model is a power model, the accuracy is measured based on the total energy, which masks accumulated power errors. The reported error is below 0.5% for the evaluation workload and 3% (1 W) for more complex software such as Jetty and Apache's Tomcat web server.

2.2.4 Statistically Calibrated Analytical Models. More advanced works use machine learning (linear regression in particular) for assessing the weights of the different variables involved in the power model formula. They differ by the number of variables considered, their target architectures, and their experimental evaluation methodology.

These models relate events fetched from both performance counters and operating system (OS). Some performance monitoring counters (PMC) are generic, being available in most of the recent architectures, while OS information is hardly deprecated over OS versions, making this approach quite stable and allowing a higher portability than the previous reported methods. Event-driven models have become popular during the past years due to their low overhead, allowing runtime estimations and management. In this section, different approaches to create event-driven models are described.

In 2000, Joule Watcher [8] was one of the first models to use PMCs as inputs. They use a linear model composed of four PMCs that strongly correlate to specific energy consumption profiles generated through the execution of micro-benchmarks to stress integer operations, floating-point operations, cache misses, and memory I/O. The selection of the PMCs was done manually and the selected variables were: retired micro-instructions (MUOP/s), floating-point operations (MFLOP/s), second-level cache address strobes (L2_ADS/s), and main memory transactions (BUS_TRAN_MEM/s). The model needs to be calibrated for each system; for this reason, synthetic micro-benchmarks are executed, while an external power meter measures the electrical power of the whole system. The authors do not comment about the accuracy of the model, only pointing out the importance of an embedded counter exclusively devoted to energy accounting.

Economou et al. [19] proposed Mantis, a full-system linear model composed of both OS utilization metrics and performance counters. They argue that the cost of PMC time multiplexing could be reduced by the use of similar OS metrics. The model requires an offline calibration phase through an external power meter for each new architecture. They selected four variables and derived a linear model using u_{cpu} (processor's utilization), u_{mem} (off-chip memory access count), u_{disk} (hard-disk I/O rate), and u_{net} (network I/O rate). Only the memory usage is measured via performance counters; all the others are fetched from the OS. For the calibration, the authors used a synthetic benchmark to vary the utilization levels of processor, memory, hard disk, and network and applied linear regression. The model was evaluated on two different hardware while running the SPECcpu2000 integer and floating-point, SPECjbb2000, SPECweb2005, streams, and matrix multiplication benchmarks. The reported average error for each evaluation benchmark reaches up to 15%, while the average error for all benchmarks is 10%.

In Reference [34], a system-wide energy consumption model was proposed using performance counters incorporating the model of electromechanical such devices as fans. The model computes the aggregated value of each device (processor, memory, electromechanical, and board component)

as follows:

$$E_{system} = \alpha_0(E_{proc} + E_{mem}) + \alpha_1 E_{em} + \alpha_2 E_{board} + \alpha_3 E_{hdd}.$$

The processor model is a linear regression model, while other devices are described using predefined analytical equations. The power consumed by the processor is expressed as a linear combination of the following variables: ambient and die temperatures for processors 0 and 1, HyperTransport transactions (HT1 and HT2), and L2 cache misses per core. The memory model is computed by $E_{mem} = N_{LLCM} \times P_{read-write}$, where N_{LLCM} is the number of last-level cache misses, and $P_{read-write}$ is the average power spent to read or write in memory. The hard drive model is $E_{hdd} = P_{spin-up} \times t_{su} + P_{read} \sum N_r \times t_r + P_{write} \sum N_w \times t_w + \sum P_{idle} \times t_{idle}$, where $P_{spin-up}$ is the power to spin-up the disk from 0 to full rotation, t_{su} is the time to achieve spin-up, P_{read} and P_{write} is the power consumed by kilobyte of data read and wrote from/to the disk, respectively, N_r and N_w are the number of kilobytes read/written from/to the disk in time-slices t_r and t_w, respectively. The electromechanical contribution is given by $E_{em} = \sum P_{fan} \times t_{ipmi-slice} + \sum P_{optical} \times t$ where $P_{fan} = P_{base} \cdot (\frac{RPM_{fan}}{RPM_{base}})^3$, where $t_{ipmi-slice}$ is the time-slice to update fan's rotation values, $P_{optical}$ is the power consumed by optical drives, P_{base} defines the base of the unloaded system, i.e., the power consumption of the system when running only the base operating system and no other jobs, and RPM measures fan's rotational speed in revolutions per minute. Finally, the main board model is $E_{board} = \left(\sum V_{pow-line} \times I_{pow-line}\right) \times t_{timeslice}$, where $V_{pow-line}$ and $I_{pow-line}$ are the voltage and current drained by the main-board, i.e., processor, disk, fan, and optical-drive power lines are excluded. The total power consumed by the system is then computed by calibrating the model to define the α values. The error on some benchmarks reaches more than 10%.

Joulemeter [31] is a freeware from Microsoft that provides power footprints for Windows end-users at process and device-specific level. Using a combination of PMC and OS information, it computes the power dissipation of the full-system as a base power (γ) added to three device models for processor, memory, and disk. Each device energy is measured linearly as follows:

$$\begin{aligned} E_{cpu}(T) &= \alpha_{cpu} u_{cpu}(T) + \gamma_{cpu}, \\ E_{mem}(T) &= \alpha_{mem} N_{LLCM}(T) + \gamma_{mem}, \\ E_{disk}(T) &= \alpha_{rb} b_R(T) + \alpha_{wb} w_R(T) + \gamma_{disk}, \end{aligned}$$

where α and γ are constants, $u_{cpu}(T)$, $N_{LLCM}(T)$, $b_R(T)$, $w_R(T)$ are, respectively, the processor utilization, the number of LLC misses, and the number of bytes read and written over a time duration T. As the static energy E_{static} cannot be decoupled from each device constant γ, in practice, all constants are coupled together in a base energy, i.e., the energy used to load the system. The tool has a calibration procedure that allows determining constants' values through a WattsUp power meter. The model was validated with five benchmarks from the SPEC_CPU 2006. The accuracy of a calibrated model can reach up to 5% (10 W) error, while the average is around 3% (6 W).

In Reference [5], Basmadjian and De Meer proposed a multi-core processor model that differs from the literature by considering that the power consumption of a processor is not only an aggregation of the consumption of its cores. Differently from the Wattch approach [12], Basmadjian proposed a methodology to estimate the capacitance of each circuit based on system's observation during the execution of micro-benchmarks. The authors model the power consumption of a multi-core CPU based on its resource-sharing and power-saving techniques. The training workload used to estimate the capacitance stresses processor's components at different levels through the execution of micro-benchmarks. Power measurements are done using an internal power

meter to directly monitor processor's 12 V channel. For the resource sharing, the authors analytically modeled processor's chip, die, and core components:

$$P_{proc} = P_{mc} + P_{dies} + P_{int\_die},$$

where P_{mc}, P_{dies}, and P_{int_die} are the power of chip-level mandatory components, die-level, and inter-die communication, respectively. Mandatory chip-level components and inter-die communication models follow the capacitive model based on the effective capacitance c_{eff}, operating voltage v and frequency f, as follows:

$$P_{mc} = c_{eff} v^2 f,$$
$$P_{int\_die} = \sum_{k=1|d_k \in D}^{n-1} c_{eff} v_k^2 f,$$

where d_k and D indicate, respectively, the set of active cores of a die k and the set of active dies involved in the communication. Similarly, the power of each die is divided into die-level mandatory components, cores, and off-chip cache memory. And recursively, the power of cores is divided into the power of cores themselves and inter-core communications. The power models are validated using two synthetic benchmarks: while-loop and look-busy. The reported errors are usually less than 5% and always under 9% (3 W).

2.3 Formula-learned Models Using Machine Learning

The following methods for proposing a power model do not assume anything on the target formula. Machine learning is used to select from a set of variables the most appropriate ones together with their relative importance to the power model (their weights). The use of machine learning methodologies to propose new power models does not depend on in-depth knowledge of the target hardware. This strength allows models to easily adapt to new architectures.

The available variables are numerous—several hundred performance counters and system values are available on a classical Linux system, for example. Not selecting a subset is viable only for specific sub-systems such as the GPU. Song et al. [51] use neural networks to create the power model of GPU using the small number of available monitoring elements. The resulting error of the model for particular workloads is 2%.

A first well-known technique is to determine statistically the most promising variables, and then using those, to learn a model. Within iMeter [64], a initial set of 91 PMC is selected from a comprehensive understanding of the internal of the target architecture, and Principal Component Analysis is reducing to seven principal components, leading to about 20 PMC. Support Vector regression is used allowing non-linear models to be created from the subset of preselected PMC. Their work is done in the context of virtualized environments and achieved 5% error on average. Bertran et al. [11] similarly proposed a PMC linear model for multicore processors for high-performance computing and extended it with others to virtualized environment [10]. They select a number of performance counters beforehand, having a close look on the micro-architecture of the Core2 Duo, exploit DVFS, and apply PCA, as in Reference [64].

A second approach is to create the model without preselecting the variables. Da Costa and Hlavacs [16] proposed a methodology to automatically generate linear power models based on the correlation of the input variables and power. As explanatory variables, it explores a set of 330 performance indicators containing PMCs, process and system information collected from `perf`, `pidstat`, and `collectd`, respectively. The model is created based on synthetic workloads implemented to stress memory, processor, network, and disk, along with a mixed setup. The authors compare two approaches: one to find the best combination of variables and another that

sequentially adds the best variables until the inclusion of new variables does not enhance the quality of the model. To determine the best combination of variables, first a linear regression with all the available variables is done. From this regression, the variables that show no impact on the power consumption are removed. Thus, for each workload type, a model is created through exhaustive search to find the best combination of the remaining variables. The model using 10 variables reports an average percentage error of less than 0.7% for the training data. Authors also provide the results based on the correlation between measured and estimated values and report a correlation always greater than 0.89. The main limit of this work is that the learning and evaluation are done on a few simple applications.

McCullough et al. [40] proposed variations of linear regression with LASSO (Least Absolute Shrinkage and Selection Operator) regularization [56] and Support Vector Machine (SVM) for regression problems [49]. The Lasso is a shrinkage and selection method for linear regression that penalizes the number of variables, encouraging the creation of models with low number of inputs. The `linear-lasso` model creates a linear regression model of the inputs. In `nl-poly-lasso` a polynomial function of the variables is used, i.e., use the same variables as before but add their squared and cubic values (x_i, x_i^2, x_i^3) letting lasso guarantee a few variables. The `nl-poly-exp-lasso` extends `nl-poly-lasso` by including exponential values of each variable (e^{x_i}). The last model is an SVM with radial basis function. Variable selection of performance counters was done by removing those that showed small correlation with the power, reducing from 884 to 200 counters. The top most correlated variables were then used as explanatory variables, allowing concurrent measurements. After the variable reduction, the methodology explored 3 OS-level variables (processor utilization, disk usage, and processor C-state residency statistics), 10 PMC variables (4 programmable and 6 fixed hardware counters), and 3 uncore counters (L3 cache performance, QPI bus, and memory controller). The benchmarks used were the SpecCPU, PARSEC, Bonnie I/O, LinuxBuild, StressAppTest, memcached, a synthetic CPU load, and sleep. All benchmarks were used during the training and evaluation of the models through a cross-evaluation technique. The results show that for single core systems, Mantis [19] and SVM have worse performance than all Lasso implementations (which have a similar performance). On multi-core systems only non-linear Lasso models outperform linear Lasso, Mantis, and SVM. The reported average energy errors for single and multi-core ranged from 1% to 3% and from 2% to 6%, respectively, with the non-linear models being slightly better than linear ones.

Witkowski et al. [63] extended Da Costa's work [16] by proposing a linear model using PMCs and processors' temperature—along with their respective square root and logarithm—for real HPC workloads. The variables are selected according to their correlation with the power—a given variable is added to the model if its inclusion on the linear model, after calibration, has a better correlation than before. Different models were created for each evaluated hardware. The reported errors vary from 1% to 1.5% with the workloads used to calibrate and from 3% to 7% to new ones, presenting an average error of 4% (15 W).

Yeseong et al. [32] propose a similar approach with a complete split between the resource consumption models of applications and the hardware power consumption models. At the cost of precision that reaches 7%, the proposed models are able to be run on different hardware with a minimal relearning cost.

Jarus et al. [29] proposed the use of decision tree to select a workload-specific model at runtime, providing more accurate results. Each workload power estimator is a linear model that uses PMCs and processor's temperature as inputs. As in Reference [63], the variables are added to the model one-by-one according to their correlation with power until the correlation between model's output and measured values decrease. However, estimation is adjusted to classes of programs. Several

models were calibrated using different classes of problems, and then at runtime the choice of which model to use is done based on a decision tree. The reported error reaches 5%.

In a different context, other works propose to use machine learning methods to model power. In Reference [2], authors use neural networks to model power of mobile systems. In this context they assume the dynamic part of the power consumption is mainly linked to the networking hardware, and they focus on selected variables linked to this subsystem.

Including the work on different fields such as mobile, most existing work propose models with carefully selected inputs or are based on existing *a priori* knowledge on the applications (such as availability of source code).

2.4 Summary of Existing Approaches

This chapter presents the state-of-the-art on computing systems' power modeling. As described earlier, the models differ in several characteristics. To compare the reported models, the following metrics were used:

Degree of autonomy. The autonomy indicated the degree of dependence of external equipment. Based on the model's requirements, they may be classified into hardware-dependent (HW) or software-only (SW) approaches.

Level of granularity. The granularity states at which level the model can estimate. It can be divided into logical and physical levels. At the logical level, the model can estimate process-, system-, thread-, application-, or VM-level measurements. While at the physical level, it can decouple the power in device-only, device-aggregation, or estimate the entire system's power. The evaluated devices were: processor (P), memory (M), network (N), hard disk (D), mainboard (MB), and fans (F).

Methodology. The methodology used to create the models is divided into simulators (S), analytical (A), or machine learning (ML) techniques.

Simplicity. The simplicity of a model can be measured by the number of variables used as inputs. Complexity in power modeling arises in part from the need to capture all the relevant features expressed by performance indicators to serve as inputs to build the models. Other aspects such as the complexity of the functions and libraries used by the model should also be taken into account, but they require an in-depth knowledge of the model, making the comparison hard to be made. Therefore, the number of input variables can be used to compare the system's overhead while computing the estimation.

Portability. The portability refers to the capacity of adaptation of the model. Three classes of portability are used: none, partial, and full. Partially portable models exploit statistical calibration procedure to adapt themselves to similar architectures, while fully portable ones generate the model for any architecture without inputs of an expert.

Accuracy. The accuracy of the model defines how precise the estimated values are relative to the measured ones. Accuracy can be measured either in percentage (%) or absolute (W) errors. Percentage error may not be a good metric to compare computing systems, since low-power systems have higher relative percentage errors than more power-hungry systems for exact same models. Also, most studies limit themselves to evaluate error on total energy while not providing error on power.

Power meter. The power meter used during the creation and evaluation impacts on the overall accuracy of the model. Some models use intra-node devices, while others prefer external devices.

Table 1 presents a comparison between the above-mentioned models and techniques. One can see a large variety of techniques and limitations. A really low number of publications explore

Table 1. Comparison of Different Power Estimators

Ref.	Year	Alias	Autn.	Granularity		Methodology	Accuracy		Portable	Simplicity	Power
				Logical	Physical		W	%		(# vars)	Meter
[21]	1999	PowerScope	HW	Process	Node	Analytical	-	-	No	2	Ext.
[8]	2000	Joule Watcher	SW	Process	Node	Analytical	-	-	Partial	4	Ext.
[12]	2000	Wattch	SW	System	P	Simulator	-	10	No	≈20	N/A
[48]	2003	Mambo	SW	System	P	Simulator	<0.2	4.1	No	24	Int.
[19]	2006	Mantis	SW	System	P+M+N+D	Analytical	-	5	Partial	4	Ext.
[34]	2008	Lewis	SW	System	P+M+D+MB+F	Analytical	2.3	-	Partial	≈18	Ext.
[17]	2009	pTop	SW	Process	P+N+D	Analytical	2	20	No	6	Ext.
[31]	2010	Joulemeter	SW	Process	P+M+N	Analytical	10	5	Partial.	5	Ext.
[16]	2010	Da Costa	SW	System	Node	ML-LR	-	<0.7	Full	10	Ext.
[7]	2010	PowerMon2	HW	System	PSU rails	Measure	-	-	No	-	Int.
[24]	2010	PowerPack	HW	System	P+M+D+MB	Measure	-	-	No	-	Int.
[40]	2011	McCullough	SW	System	P+M+N+D	ML-LR/SVM	-	6	Full	5	Ext.
[5]	2012	Basmadjian	SW	System	P	Analytical	0	5	Partial	≈16	Int.
[41]	2012	PowerAPI	SW	Process	P+N	Analytical	1	3	Partial	≥4	Ext.
[10]	2012	Bertran	SW	VM	Node	ML-LR	-	5	Full	-	Ext.
[63]	2013	Witkowski	SW	System	Node	ML-LR	15	4	No	10	Ext.
[51]	2013	Song	SW	System	GPU	ML-LR	-	2	Specific	12	Ext.
[29]	2014	Jarus	SW	System	Node	ML-LR/DT	-	4	Full	5	Ext.
[64]	2014	Yang	SW	VM	Node	ML-LR/SVR	-	5	Full	≈20	Ext.
[37]	2014	Lim	SW	System	GPU	Simulator	-	8–13	Partial	-	N/A
[13]	2014	Castaño	HW	VM	Node	ML-LR	1.38	4.6	No	4	Int.
[32]	2017	Yeseong	HW	Process & Node	Node	Analytical	-	7	Full	14	Ext.

process-level granularity; in this review, only five references tackle this level of granularity, with an accuracy varying from 3% to 20%. Most approaches intend to model each component with main efforts to processor, due to its high power consumption. Most models are linear regression models or analytical models that use linear regression to calibrate themselves. The number of variables in a model can vary from 2 to 24, although no direct influence on the accuracy can be noticed. Nine among 15 use external meters to estimate their power without considering PSU's losses, modeling noise, or providing imprecise estimations. The use of machine learning technique has increased during the past five years; most of them are based on linear regression models.

Although extremely important, the accuracy of the models is usually evaluated under specific workloads and, in most of the cases, cannot be used as a comparison. The majority of the models use synthetic workloads to evaluate the model, providing high accuracy although not using real-world applications. In addition, some workloads used to validate the models are run under controlled environment, with or without Simultaneous Multithreading (SMT) and frequency scaling, which makes it difficult to compare the reported results.

3 LEARNING METHOD FOR POWER MODELS

Several steps are followed in machine learning methods. Between data acquisition and learning, data must first be preprocessed to increase the learning quality. Machine learning methods' precision heavily depends on the quality of data acquisition and of preprocessing.

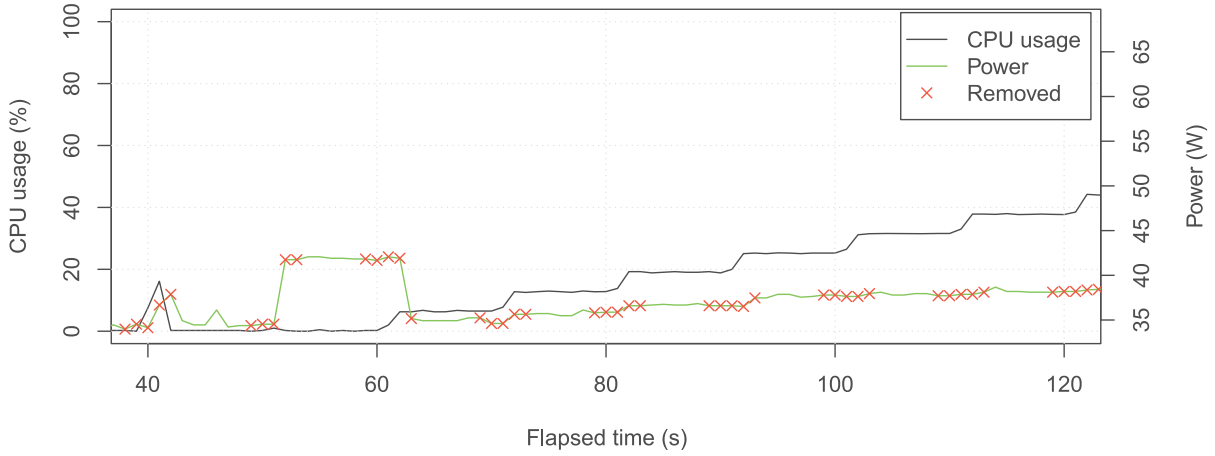


Fig. 3. Steady state method requires the removal of transient data.

3.1 Data Preprocessing

Before creating or calibrating a model, it is recommended to preprocess the collected data to achieve better and more reliable results. This section proposes a set of preprocessing methods to resolve power-metering issues. In total, four preprocessing methods were conducted, as follows:

`psu_model` This method uses the PSU conversion losses model proposed in Reference [15] to filter the acquired data.

`timesync` This method removes the timing jitters by the insertion of a synchronization pattern before the execution of each workload as described in Reference [15].

`unique` This method removes sequentially repeated values, keeping the first value and removing subsequent repeated values, as they come from the fact that watt-meters usually give outdated values if they are requested too frequently [15].

`steady` The impact of the previously identified issues can be circumvented by the detection of steady states. This can be done by an analysis of the variance of each variable or, as in our case, the record of the transients in a controlled scenario. In this case, during data acquisition, the transients are recorded and then, during data preprocessing, data values related to two seconds before and after the transient are removed from the dataset. Figure 3 shows an example of this filter, where the crosses represent the removed data.

This combination of methods was selected to preprocess all data used for creating and calibrating the models further presented in this article.

Methodology Step I: Preprocessing data taking into account the distributed and complex monitoring infrastructure

3.2 Learning Datasets

As quoted in Reference [40]:

> "Selecting the training set is often a non-trivial task and must ensure that the training set includes enough samples from various operating points. When this is not done judiciously the testing error can be large, even though the training error is small. The ideal scenario is to identify a set of basis benchmarks that are known to provide the sufficient coverage and to generate the training data from these

benchmarks. However, this is hard to achieve when the systems are complex and have a large operating space."

Therefore, we propose the use of three different cases for the learning set, which differ according to their workload configuration. These cases allow us to evaluate the quality and usability of each model, as well as the impact on the accuracy of the final model of using only a generic workload to create the learning set. The learning workload varies according to the *a priori* knowledge of the workload type to be executed on a target machine. The three cases were defined as follows:

Case 1 (ideal). Considers that the user has no knowledge of the workload executed on the host. A generic workload [23] with an exhaustive coverage is used to learn the model and all other workloads are used to validate it.

Case 2 (one-of-a-kind). Considers that the types of the workloads executed on the host are known. In this case a medium-size problem of each workload is included to the learning workload additionally to the generic one, while smaller and larger problem sizes are used for evaluation.

Case 3 (all). Used to define if a unique model can be achieved for all executed workloads. This case includes a single execution of each available workload in the learning set.

Methodology Step II: Learning dataset with a wide coverage

3.3 Additive Models

As discussed in Section 2.2.4, model calibration is one of the most used techniques for system power estimations. Additive models sum subsystem models assuming a complete knowledge of the hardware infrastructure. Linear regression can be used to calibrate these predefined models (and compute w_* in the following equations). The basic principle is to transform the observed data into the polynomial's variables.

$$P_{sys} = P_{static} + P_{proc} + P_{leak} + P_{mem} + P_{net}, \tag{1}$$

$$P_{static} = w_0, \tag{2}$$

$$P_{proc} = w_1 * u_p * \sum_{c \in Cores} f_c, \tag{3}$$

$$P_{leak} = w_2 * \sum_{c \in C} t/^oC_c, \tag{4}$$

$$P_{mem} = w_3 * LLCAM + w_4 * LLCWM, \tag{5}$$

$$P_{net} = w_5 * snd + w_6 * rcv. \tag{6}$$

An additive model is the result of adding several models, improving the precision with each one. The simplest model, a dummy constant model, consisting of the average of the learning workload power, is proposed in Equation (2); this model considers a static power consumption. Then, a processor proportional model is defined in Equation (3) and uses for each core c its frequency f_c and the processor load u_p. This is a very usual approach [5, 12, 41]. Equation (4), models the leakage power due to the processor temperature as proposed in Reference [47] and uses core temperatures $t/^oC_c$. Equation (1) is the analytical model using all the available devices including memory and network information. Memory power is computed based on the processor's last level cache

Table 2. Summary of ANN's Properties

Parameter	Setup
Weights initialization	Nguyen-Widrow
Learning algorithm	Levenberg-Marquardt
Error metric	Mean Squared Error
Early stopping condition	$E_{valid}(t) < E_{valid}(t+i), \forall i \in [1,6]$
Stopping condition	1,000 epochs
Training set size	70%
Validation set size	15%
Test set size	15%

miss when accessing (LLC-load-misses) or writing (LLC-store-misses) data (P_{mem}). The linear network model uses the number of sent and received bytes during the last time duration (P_{net}). By increasing the number of sub models, the precision increases, but also the monitoring cost (see Section 6.2).

Static power model (avg)
P_{static}
Capacitive power model (cap)
$P_{static} + P_{proc}$
Capacitive + Leakage power model (capleak)
$P_{static} + P_{proc} + P_{leak}$
Aggregated power model (aggr)
$P_{sys} = P_{static} + P_{proc} + P_{leak} + P_{mem} + P_{net}$

A linear regression using monitored variables is used to compute the model constants w_*.

In the following, aggr model will be used as a comparison to evaluate our neural network proposal, as it is close to state-of-the-art proposed in References [16, 29, 63].

4 NEURAL NETWORKS FOR POWER MODEL LEARNING

The use of Artificial Neural Networks (ANN) in regression problems has been done is several areas of expertise. In such cases the use of a Multi-Layer Perceptron (MLP) network topology with one or two hidden layers is suggested. A single hidden layer can approximate any continuous function, while with two layers it can also map discontinuous functions. In the methodology proposed here, we explore both approaches, keeping the best suitable one.

The general network setup used during the experiments is summarized in Table 2. The Nguyen-Widrow algorithm allows a uniform distribution of the initial weights and biases. We chose Levenberg-Marquardt for the learning algorithm, which uses the Mean Squared Error as error metric. Using this learning algorithm is convenient due to its fast convergence and, as it uses the Jacobian matrix instead of the Hessian matrix through the standard back-propagation algorithm, it presents a fast computing time. An early stop condition is used to avoid over-fitting; this condition requires that the validation error in a given epoch t is less than the error for the six next epochs. If the early stop condition is not reached, then a maximal number of 1K epochs is set as the limit to avoid infinite loops. The dataset is then randomly divided into training, validation and test sets in a 70%, 15%, and 15% ratio, respectively. The training set is used to learn the model, the validation set defines whether the training should stop, and the test set is used to evaluate the final resulting model.

The learning phase of an ANN is the most time-consuming one, even when using a fast convergence algorithm such as the Levenberg-Marquardt. During our experiments, for a single model creation, it reached up to 2 minutes on a classical laptop (Intel Ivy-Bridge I5-3210M@2.5 GHz). However, to estimate a value it does not take longer than 50 ms. At first, one can say that this time is not a problem, but as we are dealing with a stochastic algorithm, it needs to be executed several times when defining the neural network topology and reducing its number of input variables. For instance, the proposed variable reduction procedure for neural networks described in Section 4.3 takes almost one day to be completed. More details regarding the impact of the number of inputs and hidden neurons will be discussed later in this section.

4.1 Topology Configuration

Multilayer Perceptron (MLP) networks are known as good function approximation topologies. The number of hidden neurons of such networks impacts the number of variables that the minimization algorithm needs to parameterize to minimize the error metric. Although proven that a two-layer MLP is able to approximate any non-linear function, the number of neurons may lead to a time-consuming methodology, even unfeasible in some cases. The total number of weights and biases to optimize is computed as follows:

$$wb = (i + 1) * l_1 + (l_1 + 1) * l_2 + (l_2 + 1) * o, \tag{7}$$

where i is the number of inputs of the network (dimension of the problem), o is its number of outputs, l_1 and l_2 are the number of neurons in the first and second hidden layers, respectively. Figure 4 shows the impact of the number of variables for a problem having 20 and 200 dimensions. In this figure, HL1 and HL2 represent hidden layer 1 and 2, respectively. As stated in Equation (7), the number of weights and biases is linear to the number of inputs. For a 200-inputs problem, this number can reach around 4,500 variables, representing a very hard minimization problem to solve. This enhances the necessity to reduce the number of input variables of the model.

The stochastic learning requires several executions of the learning procedure to evaluate a topology. This issue is tackled by evaluating ANN's training through the median of 10 runs. Another issue is the time duration of the learning algorithms, which is held by setting a maximum topology size that can be learned in a feasible time execution. In addition, the accuracy of a topology may vary according to the number of input variables; this is taken into account by comparing the network performance by varying its structure from 1 to 20 neurons at the first hidden layer and from none to 15 at the second, increasing at 5 neuron steps guaranteeing that the number of neurons on the first layer is always greater than the second one. Furthermore, an ANN with only 1 neuron is equivalent to a linear regression; hence, this methodology explores linear regression, single and a double layer MLP network. To illustrate the proposed methodology, consider a set composed by (x, y), x being the number of neurons in the first and y in the second layer; the structure is tested interactively as follows:

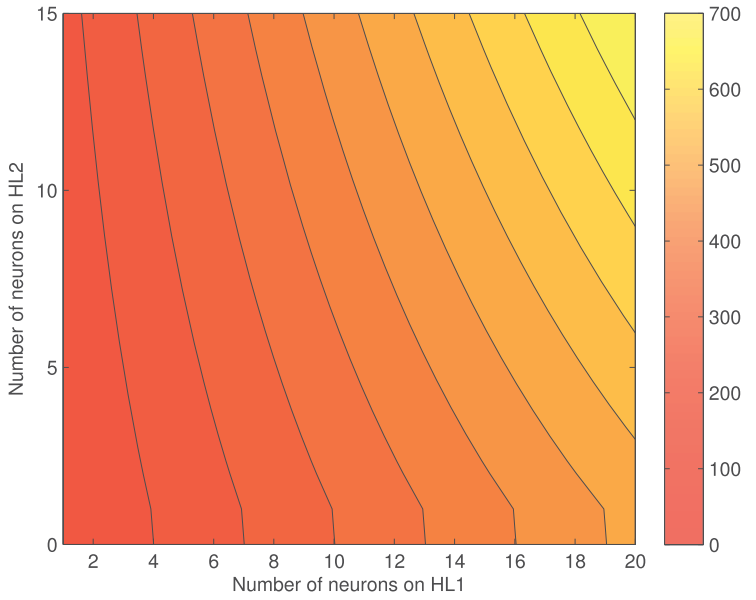
$$\{1, 0\}, \{5, 0\}, \{10, 0\}, \{10, 5\}, \{15, 0\}, \{15, 5\}, \{15, 10\}, \ldots, \{20, 15\}. \tag{8}$$

The procedure used to identify the best network layout for a given number of variables is listed in Algorithm 1.

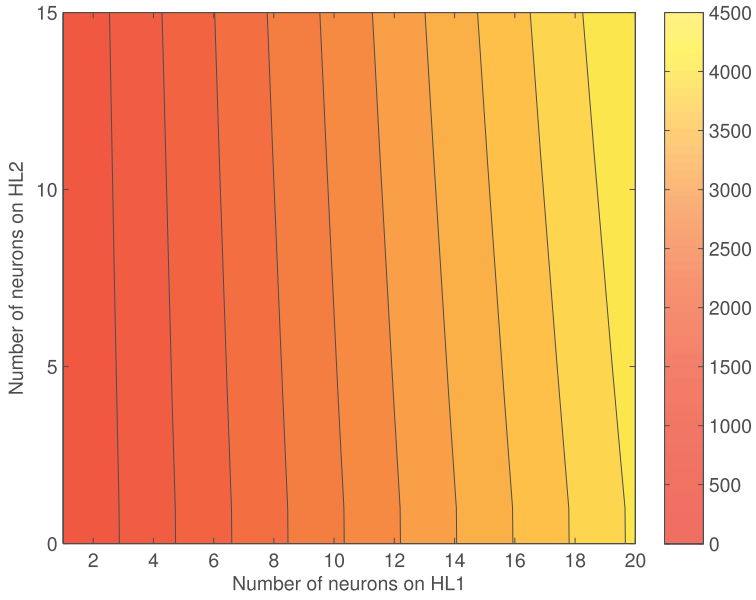
Methodology Step III: Topology optimization

4.2 The Need for Variable Reduction

There is an extensive variety of performance indicators that can be used as explanatory variables to achieve power models. For instance, the Linux library `libpfm` contains a list of 4,196



(a) 20 dimensions



(b) 200 dimensions

Fig. 4. Number of variables (weight and biases) to be optimized in an ANN according to the dimensionality of the problem.

performance counters from which 162 are supported in a classical Intel I7 processor. Some of these counters can be configured per core, increasing even more the number of variables in our models. Reference [15] shows that using numerous performance counters reduces the precision of these measures but also has an impact on performance and power consumption. High dimensional problems are usually complex to understand and hard to solve. There are several techniques [22, 53] to reduce the dimension of a problem either changing the hyperspace or reducing the number of variables. Dimension reduction modifies the hyperspace by searching for a different dimension space where two or more variables can be put together in a single dimension; the most used dimension

ALGORITHM 1: FindBestTopology

Input: A training set matrix of Key Performance Indicators $\mathbf{X}$, where each row is an observation and each column a variable of the problem; and an array $\mathbf{y}$ of targets

Output: The best topology $best_{topo}$ and its mean squared error $best_{mse}$

$l_1^{min} \leftarrow 0$; $l_1^{max} \leftarrow 20$
$l_2^{min} \leftarrow 0$; $l_2^{max} \leftarrow 15$
$\delta \leftarrow 5$
$best_{mse} \leftarrow \infty$
for $l_1 \in \{l_1^{min}, l_1^{min} + \delta, l_1^{min} + 2\delta, \ldots, l_1^{max}\}$ **do**
    for $l_2 \in \{l_2^{min}, l_2^{min} + \delta, l_2^{min} + 2\delta, \ldots, \min(l_1 - 1, l_2^{max})\}$ **do**
        $topology \leftarrow [\max(1, l_1), l_2]$
        for $i \in \{1, 2, \ldots, 10\}$ **do**
            $mse_i \leftarrow \text{AnnTrain}(\mathbf{X}, \mathbf{y}, topology)$
        if $best_{mse} \geq \overline{mse}$ **then**
            $best_{topo} \leftarrow topology$
            $best_{mse} \leftarrow \overline{mse}$
return $best_{topo}, best_{mse}$

reduction technique is the PCA (Principal Component Analysis). In the modeling perspective, it enables the creation of simpler models. Variable reduction plays an even more important role reducing, not only the complexity of the model, but also its overhead during data acquisition and online estimation. Thus, variable reduction has two main benefits: First, it generates simpler power models, which are easier to understand. Second, it reduces power estimation's impact on the target platform.

From a methodology point of view, in the following, several heuristics will be used to create a similar simple base of the data-space that is specific to our problem, i.e., with evaluating, in the context of power models, what are the improvement of the ANN quality. These heuristics are well suited for our case, but there will be no proof of their quality in more general cases, contrary to classical methods such as the ones presented in Reference [22].

4.3 Variable Reduction

Variable reduction on artificial neural networks is complex: First, because it is a non-deterministic learning algorithm; second, due to the impact of the number of variables in the network structure. This section proposes a variable selection methodology that can be used to reduce variables not only for ANNs but also for any regression problem.

A forward selection is used to achieve a model with a few variables while keeping a good accuracy. Forward selection is a search strategy that selects individual candidate variables one at a time. This method is wrapped inside the network training algorithm. Usually, forward selection methods iteratively start by training d single-variable ANN models and by selecting the input variable that maximizes an optimal criterion. Then it iteratively trains a $d - 1$ bivariate ANN adding the previously selected input variable, where d is the number of input variable candidates. A stop criterion is reached when the addition of another input variable does not improve the model's performance. To reduce the time duration of variable selection, a common approach is to select the variables most correlated with the targets (power) and insert them into the model until the model's performance stops increasing.

ALGORITHM 2: ForwardSelection

Input: A n-by-m training set matrix of KPIs $\mathbf{X}$ and an array $\mathbf{y}$ of n targets
Output: An array $\mathbf{v}$ with the most significant variables from $\mathbf{X}$

```
1  ŷ ← 0
2  best_mse ← ∞
3  ntrials ← 2
4  for i ∈ {1, 2, ..., m} do
5      r ← y − ŷ                                              // residuals
6      idx ← argmax(corr (X, r))                  // single most relevant candidate
7      v_i ← idx                                        // list of variable indexes
8      X′ ← X(:, v)
9      y′ ← y(v)
10     topology, mse ← FindBestTopology(X′, y′)
11     if best_mse ≥ mse then
12         best_mse ← mse
13         trial ← 0
14     else
15         trial ← trial + 1
16         if trial ≥ ntrials then
17             return v
18 return v
```

A modified version of the forward selection method is proposed to fit the ANN time constraint and to include the most important variables into the model. Sometimes, the variables most correlated with the target are correlated among themselves. Thus, we propose a modification of such procedure, by not training all single variables ANNs, but selecting the variables most correlated with the residuals and inserting them into the power model if the model's performance enhances, as shown in Algorithm 2. The residuals of a given model are calculated as the difference between expected and actual values (see line 5 of Algorithm 2). In addition, we allow that the insertion of a variable decreases the performance two times before stopping the algorithm, enabling the creation of slightly more complex models. The proposed methodology is generic and can be used for reducing the number of variables in other machine learning algorithms; it is just a matter of changing line 10 of Algorithm 2 to receive the learned model from another regression technique instead of the best ANN topology.

Methodology Step IV: Variable reduction

4.4 Theoretical Limitations

The use of artificial neural networks is subject to a fundamental constraint: The input variables to estimate a new value need to be in the same range as the training set. For instance, if during the training of an ANN a variable x varies from 5 to 15, one cannot use it to estimate a value for neither x greater than 15, nor smaller than 5. Otherwise, the activation can be saturated and the predictions might not be realistic. Overall, the quality of the resulting ANN will be directly correlated with the quality of coverage of the benchmarks used during the learning phase.

Table 3. Characteristics of I7 Nodes Used in Experiments

Processor	Intel Ivy Bridge I7-3615QE
Number of cores	4 (8 logical)
Cache L1	32 kB (per core)
Cache L2	256 kB (per core)
Cache L3	6 MB (shared)
RAM	16 GB
Op. Frequencies	1.2–2.3 + Boost
HardDisk	None
Network	1 GB Ethernet
Max. power	46.1 W
Idle power	10.9 W

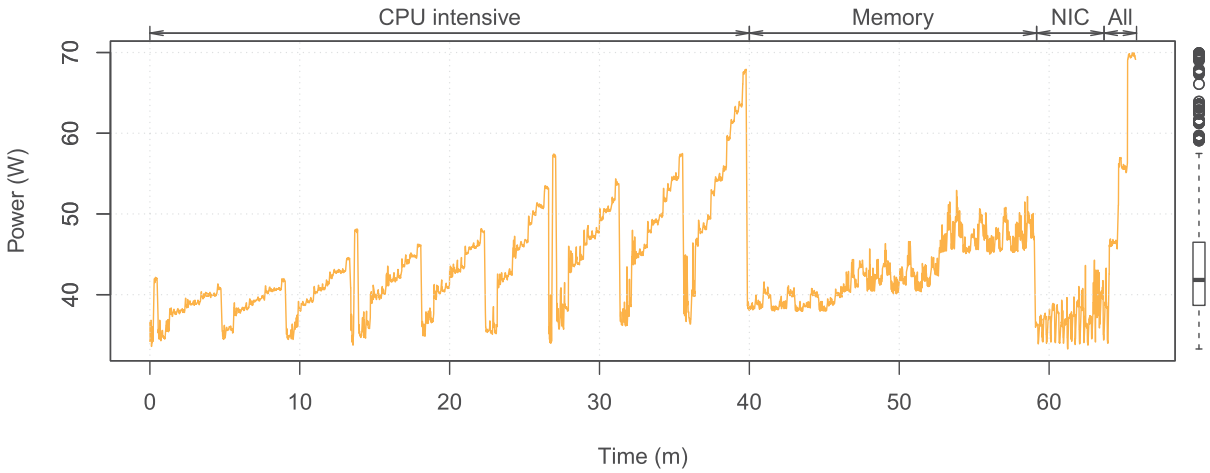


Fig. 5. Generic workload proposal based on μ-benchs to stress the processor, memory, and network, along with a mixed setup to stress all devices concurrently.

5 METHOD OF EVALUATION OF THE METHODOLOGY

Methodology of ANN Workflow Power Estimation Model

5.1 Hardware and Software Platform

Experiments are done on a four-node Ivy Bridge I7-3615QE composed of four cores, 16 GB of memory, 1 GB Ethernet, and without a local disk. A complete description of the motherboard characteristics is given in Table 3. Similar experiments were conducted on a low-power processor (Intel ATOM N2600), and as they reached the same conclusion, they will not be detailed in the following.

We aim at being able to model different types of workloads, from Cloud to high-performance computing fields. Thus, we selected the following ones to obtain a large evaluation set:

Generic (Figure 5) is a combination of micro benchmarks targeting several subsystems: processor, memory, network, or entire system stress. It runs through several phases, first CPU-intensive stressing cores, then a memory-intensive phase followed by a network-intensive phase. A last phase stresses all the above sub-systems.

Cloud Apache web Server [54], Pybench [33], OpenSSL [55],
HPC Stress (Linux benchmarking command), C-Ray [58], GROMACS [9], HPCC [18], NPB [4].

Several benchmarks encompass numerous programs. For example, GROMACS, a molecular dynamics benchmark, uses several input files such as *lzm* (named as **gmx_lzm** in the following). HPCC and NPB propose multiple benchmarks with several sizes. In the end, there are 1 *generic* benchmark, 4 *cloud* benchmarks, and 18 *HPC* benchmarks. The three learning workloads described in Section 3.2 will be in the following experiments:

Case 1 ideal workload: **generic** benchmark only;
Case 2 one-of-a-kind workloads: **generic**, **apache**, **c-ray**, **gmx_lzm**, **hpcc_B_dist**, **npb_B8_dist**, **pybench**, and **openssl** benchmarks;
Case 3 all workloads: all available benchmarks.

During all the experiments, the monitoring and model are done on a single node (called *main* node), but the benchmark is running on the adapted number of nodes. For single code such as **openssl** it will run on the main node; for the distributed HPC benchmarks such as **hpcc** it will run on the four nodes, and for the asymmetric one such as **apache** the main node will run the benchmark while the other nodes will send it requests to provide its load.

5.2 Metrics

The analysis of each workload can be done at a high granularity, as illustrated on one example in Figure 6. Each model and workload can be evaluated given their estimate for the power consumption, the residuals of each estimate, the residuals histogram, and the regression of the estimations and targets. The estimates show how close the model is from the target; this result gives an empirical feeling of how good is the model. The residuals' histogram shows the dispersion of the error, while the residue graph allows identifying where the error is larger. Finally, the regression between estimations and targets provides the correlation between them, showing how close is the model from the actual case. The example of Figure 6 shows the results from a synthetic generic workload when using an artificial neural network as predictor.

However, as the number of evaluated workloads and models increases, the above-mentioned analysis gets too difficult to manage. This analysis can only be used during design of the methodology but not for actual comparison between numerous models due to this complexity. Thus, the Mean Average Error (MAE) was used to summarize the performance of a model into a single number. The smaller the MAE, the better the model is. Small errors may incur in better correlation between targets and estimations and provide the order of magnitude of the error in Watts. In the case of Figure 6, the MAE is 1.175 W.

The error metrics used to evaluate the models consider both average power and total energy consumed by each workload. MAE and MAPE metrics are derived to provide estimators' errors in Watts and Joules, respectively. The error metrics are described in terms of estimated ($\hat{p}$) and measured (p) power as follows:

$$MAE_W = \frac{1}{N}\sum_{i=1}^{N} |p_i - \hat{p}_i|\,, \tag{9}$$

$$MAPE_J = 100 * \frac{|\sum_{i=1}^{N}\hat{p} - \sum_{i=1}^{N} p|}{\sum_{i=1}^{N} p}, \tag{10}$$

where N is the number of samples, MAE_W is the absolute error (in Watts) of power estimations, while $MAPE_J$ is the percentage error of the energy estimation for an entire workload execution.

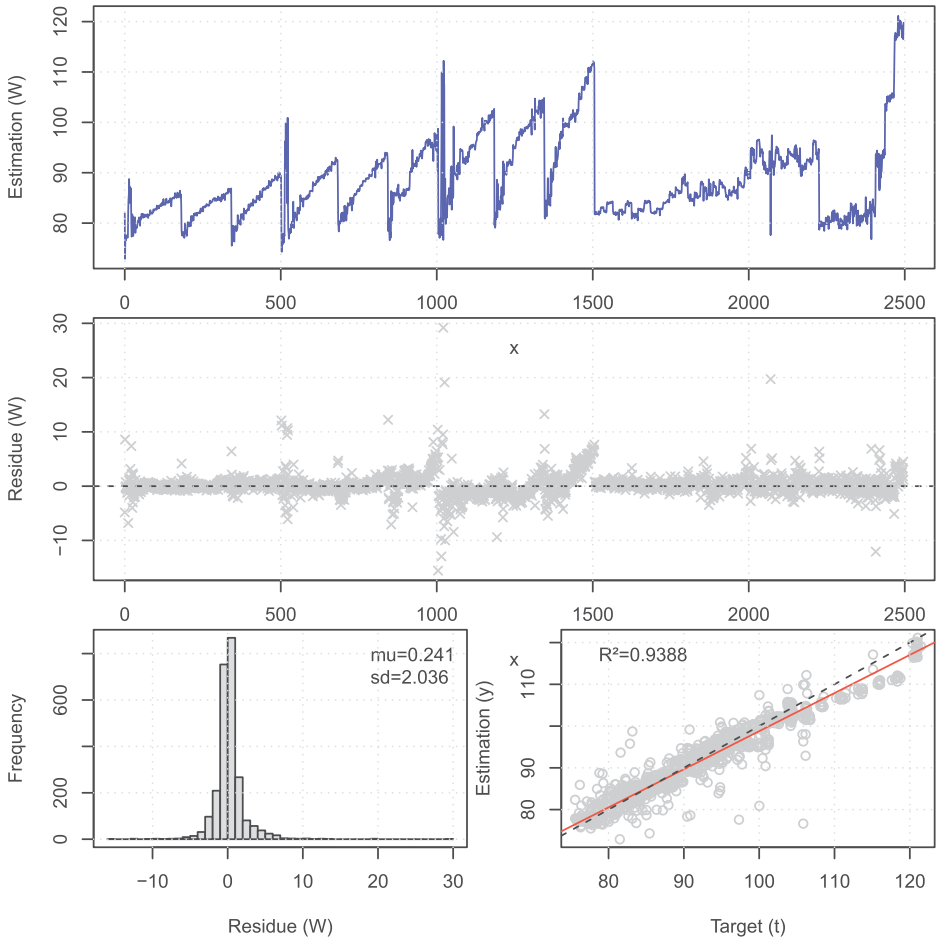


Fig. 6. Generic workload results for an artificial neural network using all available variables as inputs. This model has a MAE of 1.175 W for this workload.

$MAPE_J$ is similar to the value used in most of the studies described in the state-of-the-art. MAE_W represents the actual error of the models for each estimation.

Each workload defined previously was executed five times. The first execution is used as the learning set according to its case, and the four other runs are used to evaluate the models. The evaluation is done based on the average and standard deviation of the error metric (MAE_W) for these runs.

First, a comparison between the *additive* models is done, followed by a comparison between the learned models. For the reduced ANN model, once the variables are reduced, all the workloads are re-run collecting only the required performance indicators. This decreases the impact of time multiplexing for the PMCs variables (as shown in Reference [15]) and reduces the system overhead of the data acquisition tool. This re-execution profiling works in the same way as before, but the learning will only search for the best topology. Thus, for the learned models the comparison is done between the linear regression and ANN using all variables and only reduced variables.

After identifying the best of each class of models, i.e., additive and learned, both models are compared.

6 EVALUATION AND DISCUSSION

Figure 7 is a high-level reminder of the global methodology proposed in our approach.

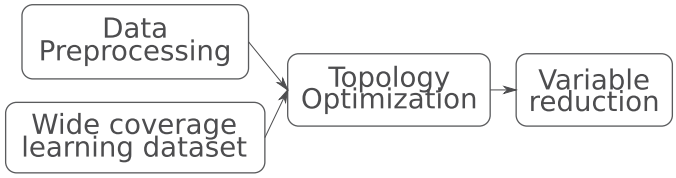


Fig. 7. Workflow of the different elements of the proposed methodology leading to a high-quality neural network.

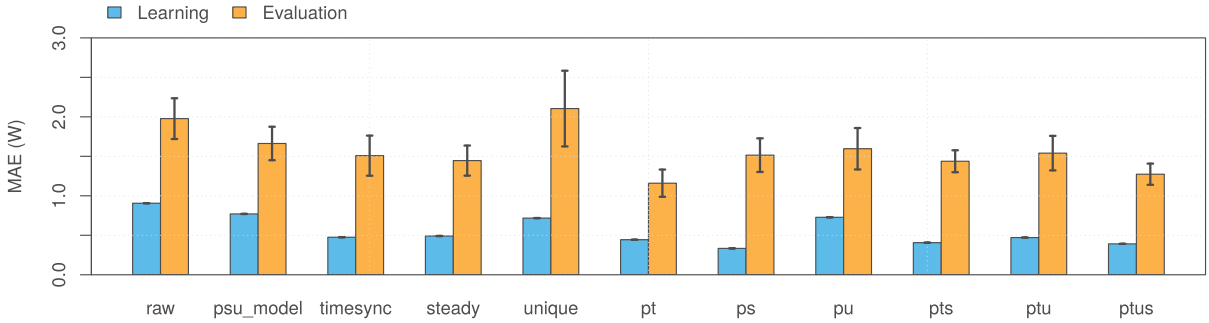


Fig. 8. Impact of each preprocessing method on the final model for the training and evaluation runs.

6.1 Data Preprocessing

The impact of data preprocessing on system-level models' accuracy is evaluated by training an ANN for some combinations of the preprocessing methods as described in Section 3.1. Accuracy of the methods is defined using the MAE metric to measure their learning and validation's error. The results for each preprocessing case exploit the same data acquired during several executions of the `generic` workload. Figure 8 shows the results of this experiment, where the bars represent the average of the error metric, while the whiskers represent the standard deviation. For the learning phase, a single run is considered, so no standard deviation is seen. However, four datasets from four different executions were used to evaluate the methodology. The number of test executions was kept small due to the `generic` workload's long execution time duration, which takes more than one hour to be executed and due to the small standard deviation observed from the data from the test executions.

In Figure 8, the `raw` bars represent the ANN's performance when using the acquired data without applying any preprocessing technique, while the `psu_model`, `timesync`, `steady`, and `unique` bars correspond each to a single method applied to the raw data. One can see from these five grouped bars that any preprocessing method alone is better than using the raw data for learning. Although the `unique` method has the worst evaluation with a high variance of the error, it is still under the same range of error than the `raw` data. The best preprocessing techniques, when applied independently, are the `timesync` and `steady`, which present similar performance for both learning and evaluation datasets. However, due to the non-linearity that the `psu_model` adds to the data (as shown in Reference [15]), we decided to keep it for further evaluations, coupling it with other methods.

The proposed preprocessing methods were then coupled as follows: Bars named `pt`, `ps`, and `pu` combine the `psu_model` with `timesync`, `steady`, and `unique` models, respectively. The results of these combinations show that they decrease the evaluation error of the `timesync` and `unique` models, keeping the error of the `steady` model identical. This means that PSU modeling will either enhance or not influence the results of data preprocessing. To continue the experiment, it was decided to keep the method that enhanced the accuracy the most, i.e., the `timesync`. Then, the `psu_model` and `timesync` were combined with `steady` and `unique` methods (`pts` and `ptu` bars),

and the evaluation error slightly increased, suggesting the worst combinations. Finally, all methods were combined (ptus bars). The combination of all methods presents similar evaluation accuracy with the pt combination, given that their evaluation variance superpose, while the learning error of ptus slightly decreases.

The comparison between the ptus combination and the raw data shows that the former enhances the accuracy of both learning and evaluation results. Learning error decreases from 0.90 to 0.39 W, i.e., an improvement of 55%; while the evaluation mean error goes from 1.97 to 1.27 W, i.e., a gain of 35%. These results are quite impressive, considering that the only change here was the data preprocessing. The rest of the experiments conducted in this work use the ptus combination to preprocess the data, since it covers all the identified issues and provides large improvement of models' accuracy.

6.2 Additive Models

The calibration of *a priori* models is the most common approach for power modeling, providing some flexibility to the models to adapt themselves to new hardware architectures. This section compares the models described in Section 3.3 for the three learning dataset cases described in Section 3.2. The results presented in Figures 9 to 11 show each model's average performance and standard deviation for the learning dataset and for every workload used for evaluation (see Section 5.1). As a general aspect, one can notice that the learning error always decreases as the model increases in complexity. However, the error on evaluation does not necessarily decrease when the complexity of the model increases.

Figure 9 shows the results for the ideal case, i.e., when only the generic workload is used for training. One can see that the average model, which considers the power consumption constant, presents very poor results reaching up to 25 W of MAE. These results evidence that this assumption cannot be used in most of the cases. It can also be noticed that the errors for the workloads that are not too close to the generic workload, such as stress, Gromacs (gmx), HPCC, and NPB, have a poor performance when using the capacitive model. However, more complex models provide better results. The capacitive with leakage power and the aggregated models have similar performance. Most of the workloads present evaluation errors near 2 W; however, for the stress, HPCC_A, and NPB_A workloads, some high errors are noticed. In addition, a large difference can be noticed for the distributed version of the NPB size C (npb_C8_dist).

Figure 10 shows the results for Case 2, where one workload of each kind is used in the learning set used to calibrate the model. In this case, the overall error of all models decreases. Once again the average model has the worst accuracy. Even though the capacitive model presents large improvements compared to the previous case, its performance is still below more complex models. The results of the capacitive with leakage and aggregated models surpass the others for nine of the workloads (generic, apache_perf, openssl, pybench, hpcc_B, hpcc_C, npb_B4, npb_B8_dist, and npb_C4), have similar performance in eight (apache, c-ray, gmx_dppc, hpcc_A_dist, hpcc_B_dist, hpcc_C_dist, npb_A4, and npb_A8_dist), and a worse performance for the five remaining ones. One can notice that for the npb_C8_dist workload, the capacitive with leakage and the aggregated models have a large difference, similarly to the previous learning case.

The results when learning from one execution of each workload (Case 3) is shown in Figure 11. This case allows the evaluation of each model's best performance, since it considers all possible configurations of the workloads. One can notice that for all models, except the Static power model (avg), the results are quite close to Case 2. This proximity convinces that Case 2 is a good approach, i.e., the concept of using one workload configuration of each kind to create the model is suitable. It is important to notice that even for this case, the error of the stress workload is still high. This is due to the dynamic behavior of the workload, which varies from high to low power usage in a

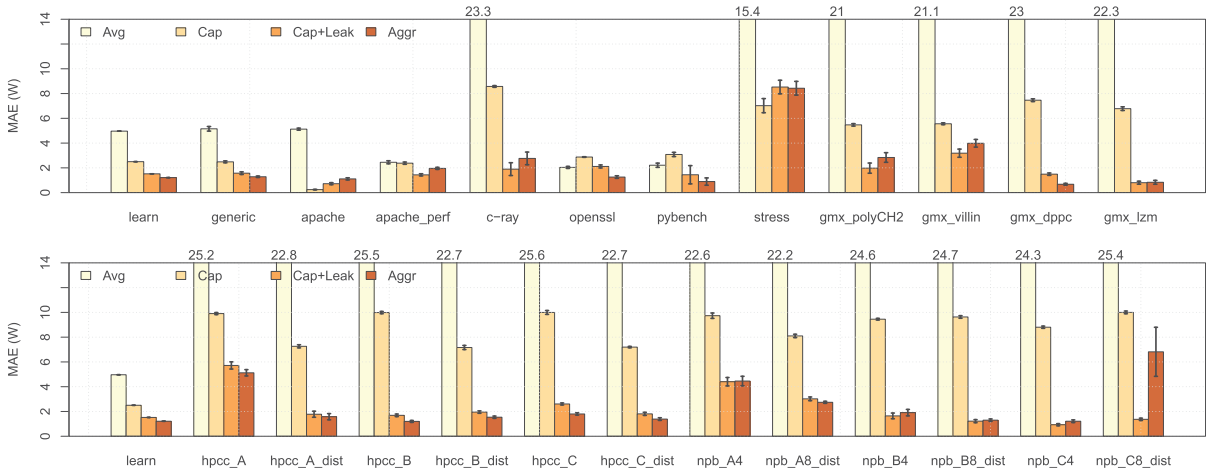


Fig. 9. Additive models' performance after calibration using the learning workload for the ideal case (Case 1).

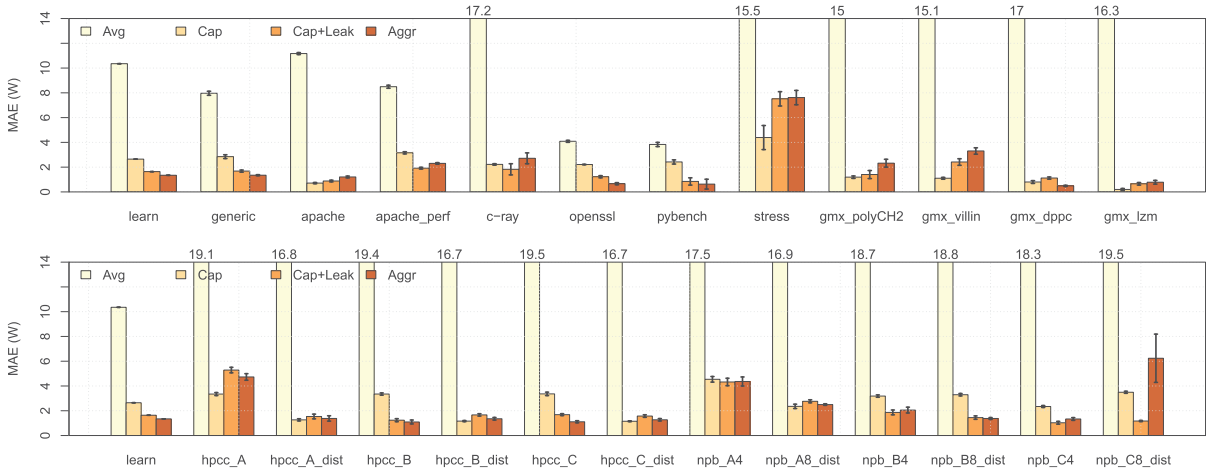


Fig. 10. Additive models' performance after calibration using the learning one workload of each kind (Case 2).

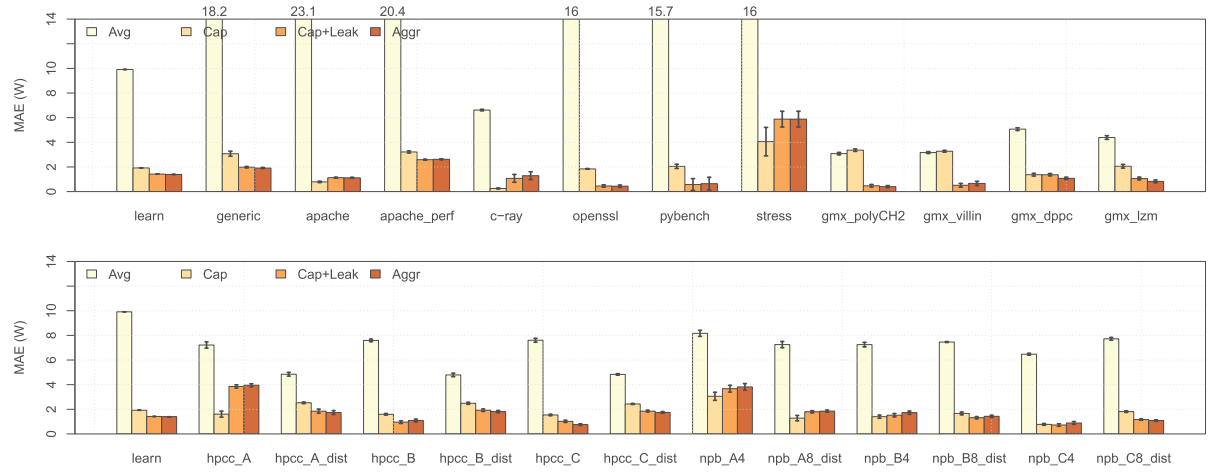


Fig. 11. Additive models' performance after calibration using the learning all workloads (Case 3).

Fig. 12. Comparison between learning workloads for each case (L1, L2, and L3) and the evaluation runs for the `npb_C8_dist` workload (V1, V2, V3 and V4).

short period. Due to this high dynamic heat production, the models that use the temperature as an input will not provide good results, because temperature presents inertia and its changes are not instantaneous.

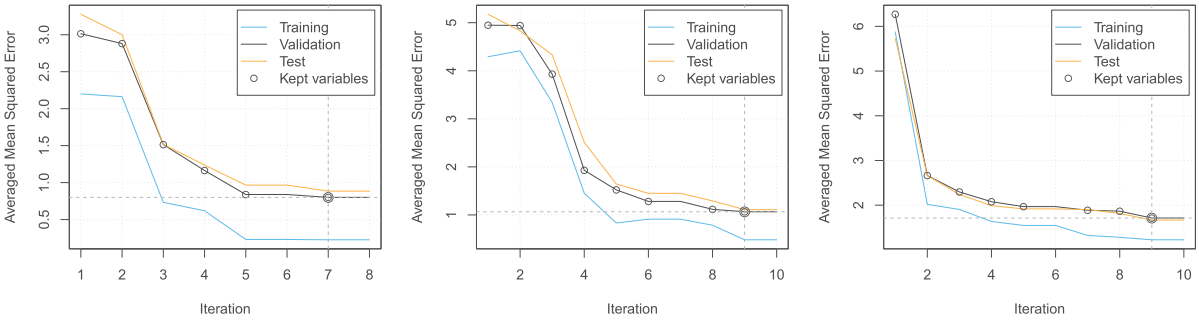
The aggregated model has three extra variables compared to the capacitive with leakage model: total number of cache misses; and network sent and received bytes. An analysis of the `npb_C8_dist` workload was done to understand why the results of learning cases 1 and 2 present such difference between these models. Figure 12 shows the means and standard deviations of each extra variable. One can notice that the network communications of the evaluation datasets are higher than any learning dataset. Since they are not in the same range as the learning data, any variation above the training range may incur unexpected behavior (see Section 4.4). This does not happen for the learning Case 3, since the workload is included as part of the learning dataset.

6.3 Machine Learning Models

The process of learning a model from scratch without an expert's input enables its usage to model new hardware architectures without any extra implementation cost. This section compares two machine learning methodologies: Linear Regression (LR) and Artificial Neural Networks (ANN). The linear regression is used as a reference learning algorithm to evaluate the performance enhancement for the ANN approach. As described in Reference [15], performance counter measurements vary according to the number of counters measured concurrently, suggesting that models with fewer counters could be more reliable. Furthermore, the number of inputs of an ANN will have a large impact on the total number of parameters it needs to optimize during the learning phase and then on its accuracy. These issues are tackled through the variable reduction of ANNs and the re-execution of workloads. Finally, a comparison between LR and ANN using all input variables and an ANN with the reduced variables is done.

6.3.1 Variable Reduction. Variable selection was executed for each of the three learning workloads defined in Section 3.2. Variable reduction is based on a bottom-up approach, as described in Section 4.3. The two approaches described earlier were executed: one that includes variables based on their correlation with the power consumption of the machine (reference technique from literature), the other with the residuals of the previous selected model (our proposed approach). After the variable selection, the number of explanatory variables of the model decreased from 60 to a maximum of 8 variables, depending on the reduction methodology.

Figure 13 shows the evolution of the average error for our residuals correlated variable insertion methodology. For all the learning dataset cases, the error evolves according to the number of input variables included in the ANN. The average error of 10 ANNs is used, since the learning algorithm used to train an ANN is nondeterministic. The x-axis represents the number of variables selected for inclusion during an iteration of the algorithm, while the circles indicate the variables that were actually kept in the model. The last circle represents the best combination of variables. One can



(a) Case 1: ideal (b) Case 2: one-of-a-kind (c) Case 3: all

Fig. 13. Evolution of the averaged error during the variable selection procedure. At each iteration, the variable most correlated with the residuals of the previous model is included in the model only if the error on the evaluation set is reduced.

Table 4. Comparison of Models' Explanatory Variables Selected through Each Variable Reduction Method (Power and Residual Correlated Selection) Executed for Each Learning Case (C1:ideal, C2:one-of-a-kind, and C3:all Workloads)

Variable reduction method	Power correlated			Residuals correlated		
Performance indicator	C1	C2	C3	C1	C2	C3
msrtemp	x	x	x	x	x	x
msrpstate		x		x	x	x
msrcstate		x	x	x		
pmccachemisses				x	x	x
syscstate				x	x	x
pmcdTLBstoremisses				x	x	
pmciTLBloads					x	x
pmcLLCprefetches					x	x
syscpuusage		x	x			
pmcbranchmisses						x
pmcdTLBstores	x					
pmcinstructions	x					
pmcL1dcachestoremisses						x
pmcmajorfaults					x	
Number of variables	3	4	3	6	8	8

see that the error rapidly decreases for the first included variables, then a slight decrease is still seen, until it reaches a stagnation point. By allowing a variable to not be sequentially included in a model if the ANN evaluation error is greater than the iteration before, we had the inclusion of 6, 8, and 8 variables for Cases 1, 2, and 3, which fail for iterations 8, 9, and 9, respectively. We can also see that when the learning dataset grows, the final error gets larger; this happens because of the difficulty of finding a single model able to model all situations presented in the whole dataset. In addition, the variation of the evaluation curve is close to the training set, showing that there were no data over fitting.

Table 4 lists the variable selected by both variable reduction approaches: power and residuals correlated variable insertion. Each reduction approach was executed for all learning cases. From

Table 5. Best Network Configuration, Selected from Different Variable Reduction Methodologies Using Distinct Learning Datasets

Reduction methodology	Learning Case	Inputs	Hidden Layer size First	Second	Weights+Biases
Power correlated	1	3	15	10	366
Power correlated	2	4	20	0	281
Power correlated	3	3	15	5	236
Residual correlated	1	6	15	10	546
Residual correlated	2	8	15	5	566
Residual correlated	3	8	15	10	666

the selected variables, one can notice that the temperature (`msrtemp`) is always present, as it is the variable most correlated with the power. As we are dealing with auto-generated models, there is no constraint regarding the observed variables. For the power-correlated approach, there is no consensus on any other variable, although the final models present simpler models having at most four input variables. However, for the residuals-correlated approach, in addition to the temperature, three other variables are present in all learning cases: P-States (`msrpstate`), C-States (`syscstate`), and cache misses (`pmccachemisses`). These variables represent processor's performance (like frequency), power savings techniques, and the RAM accesses, all of which have significant impact on the power consumption of a machine. It is important to notice that frequently used variables in the literature such as CPU usage variables (e.g., `syscpuusage` and `pmcinstructions`) were not selected in the residuals-correlated models for any of the learning cases. This could happen due to the high correlation between CPU usage times frequency and the temperature; as it is quite used and available in all hardware and usually updated more frequently, the inclusion of such combination could be of great value to replace the temperature.

Table 5 shows the ANN's topology configuration selected from different variable reduction methodologies using distinct learning datasets. Most of the networks present a similar topology consisting of two hidden layers with 15 neurons on the first layer and 5 or 10 neurons on the second one. The only exception is for Case 3 of the power-correlated method, which has only one hidden layer. This shows that the range of search for the topology definition (from 1 to 20 neurons on the first layer and from none to 15 on the second one) is enough, since the selected topologies are not located on a range border. Another aspect to be noticed is that the number of weights and biases for all the cases are smaller than the number of samples used in the training set, signifying that the number of samples used is adequate. The next section analyzes the accuracy of each learned model, including the reduction techniques. The remainder of this section will use this model to compare with additive techniques.

6.3.2 Models Comparison. The number and types of available variables depend on the hardware architecture. Thus, self-adaptive models need to be able to generate the model without external influence. This section compares four self-adaptive models; two of these use all available variables as input while using linear regression and artificial neural network learning algorithms, `lr-all` and `ann-all`, respectively; while the other two reduce the number of variables based on the power and residuals correlation, `ann-red-pow` and `ann-red-res`, respectively. As in the previous section, Figures 14 to 16 show the results from each of the three learning cases described earlier.

It is clear, from Figure 14, that the use of the `generic` workload to train a highly accurate model is not enough for any of the tested learning methodologies. Here, the importance of properly selecting the dataset to learn from is evident. For all methodologies, Case 1 presents a bad accuracy

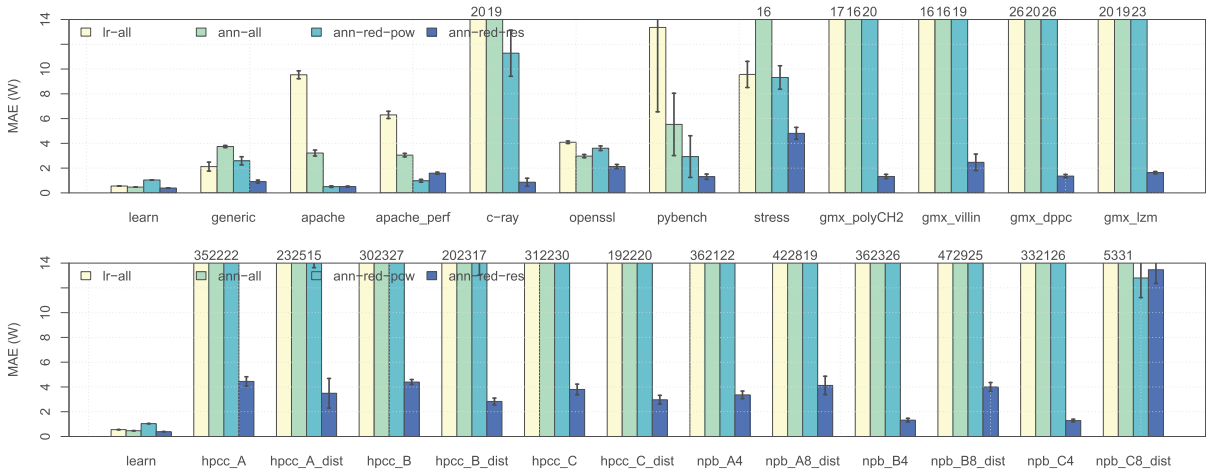


Fig. 14. Machine learning models' performance after calibration using the learning workload for the ideal case (Case 1).

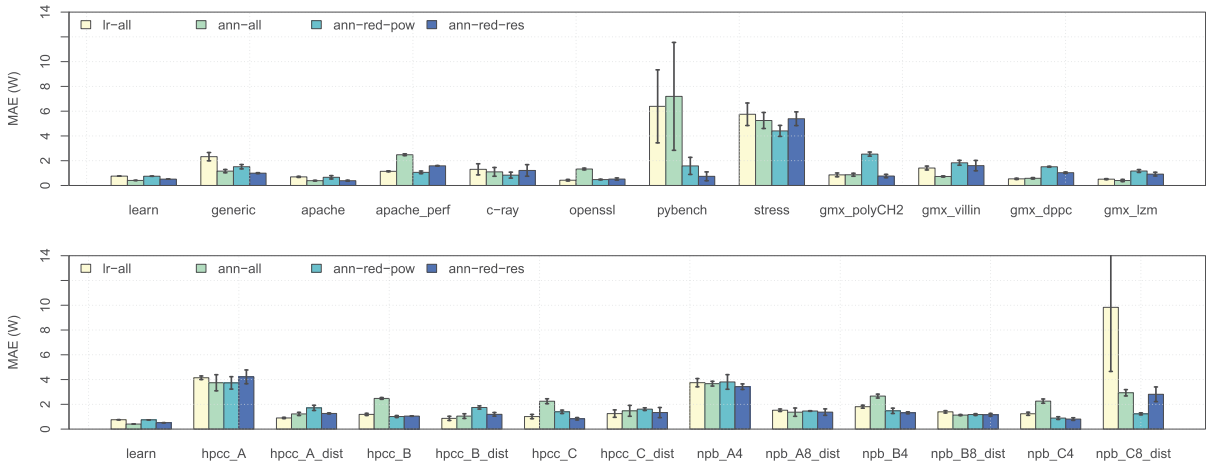


Fig. 15. Machine learning models' performance after calibration using the learning one workload of each kind (Case 2).

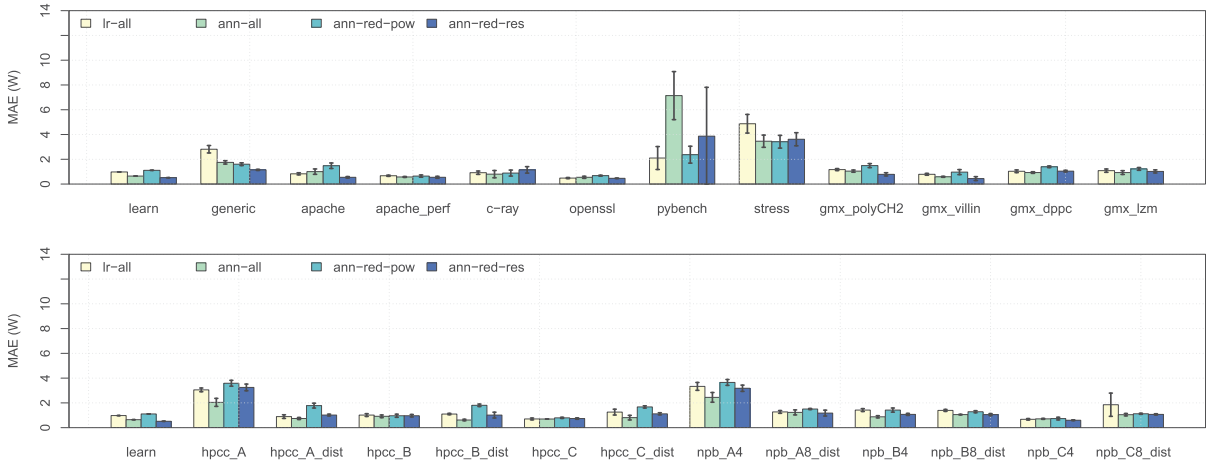


Fig. 16. Machine learning models' performance after calibration using the learning all workloads (Case 3).

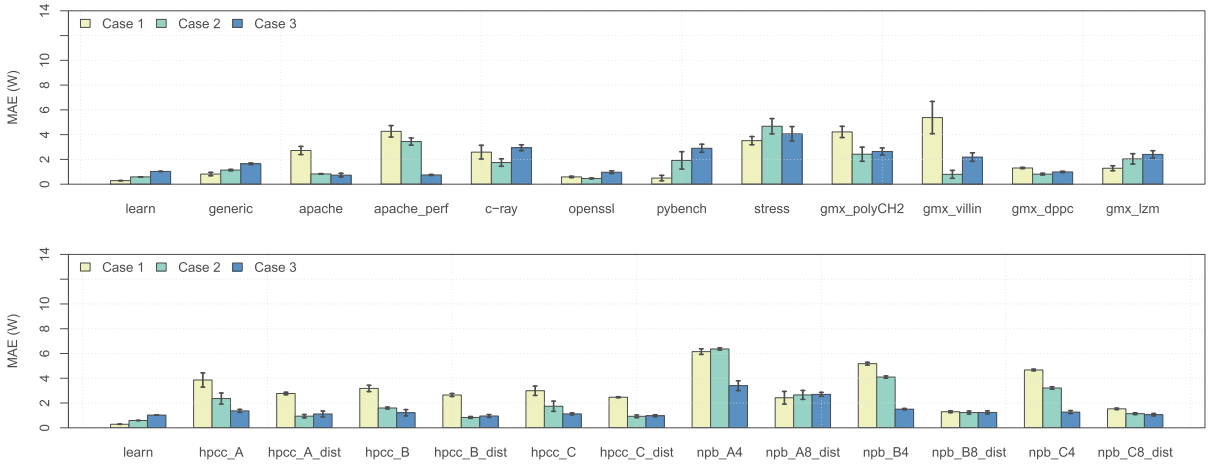


Fig. 17. Residual's based reduced ANN power models' comparison for each learning case after workload re-execution.

when compared with Cases 2 or 3. The only methodology that presents suitable results is the reduced ANN based on residuals (`ann-red-res`), except for the `npb_C8_dist` workload.

The overall comparison between Case 2 and 3, Figures 15 and 16, present similar results. This means that the use of Case 2, where one workload of each kind is used to learn the model is a good solution, not only when calibrating additive models but also when a model is created from scratch. Once the learning dataset is well-chosen (Figures 15 and 16), the learning methodologies present similar accuracies. One can notice that there is a small difference in the learning aspect, where `ann-all` and `ann-red-res` present better results. For the evaluation, the only workloads presenting a significant difference are `pybench` and `npb_C8_dist`, where the `lr-all` and `ann-all` have a high variation and mean error. For the other workloads, the models have similar results. Due to the capability of providing a useful model even when using only the generic workload (Case 1) to learn the model, the `ann-red-res` methodology was chosen as the best suitable to model the power consumption from scratch.

6.3.3 Workload Re-execution. The number of concurrently monitored performance counters influences their measurements (as shown in Reference [15]). This section analyzes the impact of such measurements on the accuracy of a reduced ANN. Thus, after the variable reduction procedure, all workloads must be executed once again to avoid noise and provide better precision measurements.

Figure 17 shows the error performance for each workload after five reruns where only the selected variables were monitored. One can notice an overall improvement when compared with the `ann-red-res` reduced model of Figures 14 to 16. The results of learning Case 1 present the most visible accuracy enhancement, mainly for the `npc_C9_dist` where the MAE decreases from 13 to 3 W. Learning Case 3 has similar results—except for the `hpcc_A`, the error decreasing from 4 to 2 W. For the learning Case 2, the results are close to previous runs.

It is important to notice that the decrease on the number of monitored variables also decreases the power consumed to observe the system without compromising model's accuracy. Thus, the re-execution of the workloads for learning the model once again provides better accuracy and more energy-efficient monitoring.

6.4 Additive vs. Machine Learning Models

In this section, we compare the calibration of additive models (`aggr`) with the creation of models from scratch using machine learning models (`ann`). In here, we are not only interested in the MAE

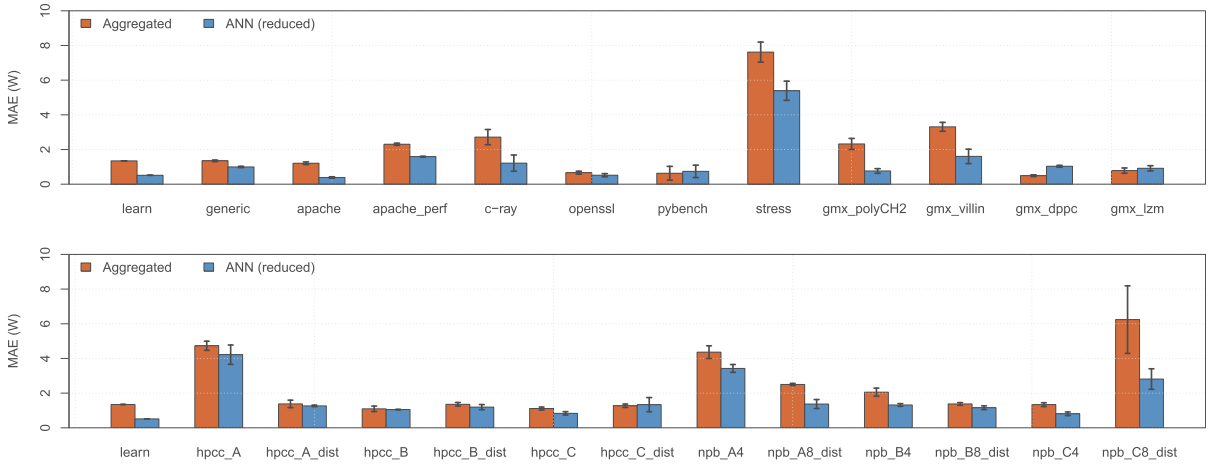


Fig. 18. *A priori* model vs. reduced neural network power models' comparison using the learning one workload of each kind (Case 2), absolute error.

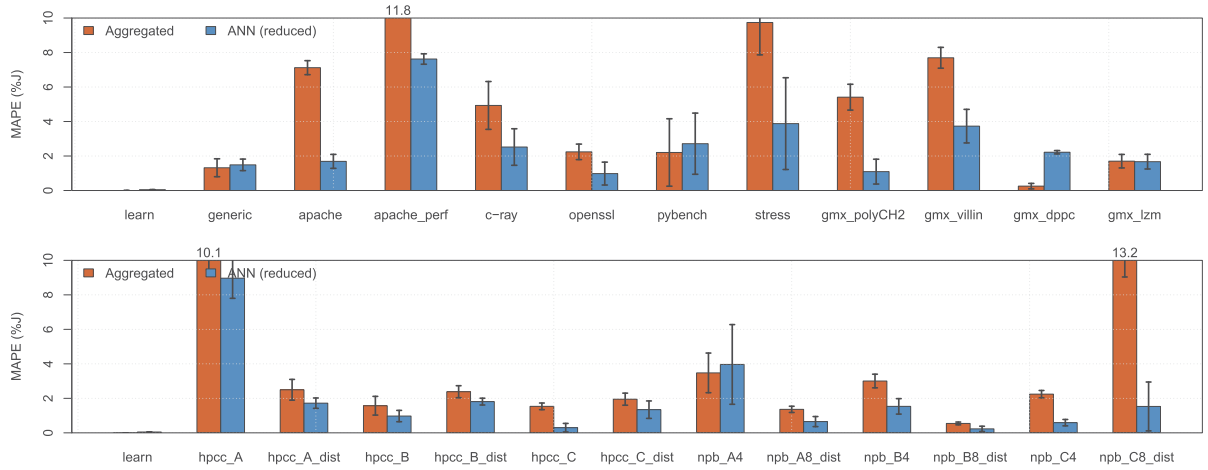
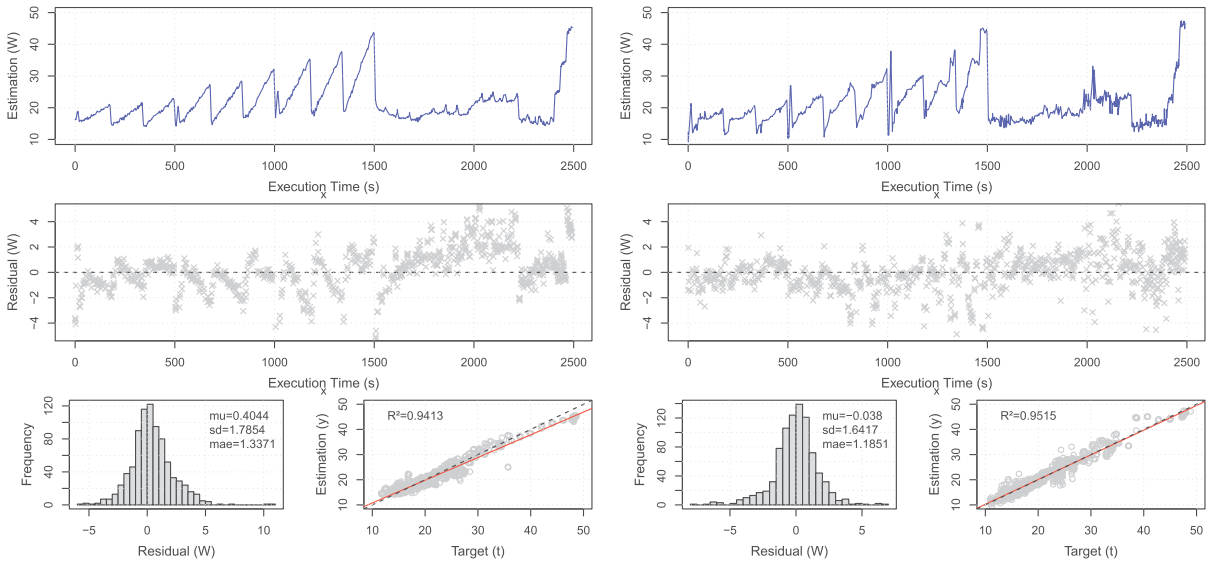


Fig. 19. *A priori* model vs. reduced neural network power models' comparison using the learning one workload of each kind (Case 2), accumulated error.

(Equation (9)), but also on the overall energy consumption error measured through its percentage error metric (Equation (10)). This comparison in done based on the most suitable learning workload, i.e., the use of one workload of each kind to be learned (Case 2). Figure 18 compares the best model from each technique. One can notice that, for all workloads, the learned model has a similar or better performance than the calibrated one. The superiority of the learned model is more evident in Figure 19, where the percentage error of the energy consumed during the entire workload execution is shown. With the additive models, eight workloads have large energy estimations error (5%); while for the learned model, only two workloads surpass this threshold.

An in-depth view of the `generic` workload's evaluation can be seen in Figure 20. This figure shows the aggregated and ANN models' estimation and residual analysis. It can be noticed that the ANN has a better performance than the aggregated model, presenting lower error (1.18W), higher correlation ($R^2 = 0.95$), and smaller standard error (residuals have a mean of -0.04 W and standard deviation of 1.67 W). From this figure, one can see that the aggregated model has the best performance when the temperature and the dissipated power are highly correlated, i.e., before 1500 s. After this point the variance of the residuals is higher. The same does not happen with the ANN, where the variance of the residual is more uniform across the entire execution and the



(a) *A priori* model

(b) Artificial neural network

Fig. 20. Performance of system-level power estimators for a random execution of the `generic` workload.

dependency to the temperature is not so high. This allows the ANN to model high variances in short time as seen in 1500 s, where the power drops from over 40 W to only 15 W in one second. Thus, even if the power errors are equivalent in the case of the `genetic` workload, the use of the ANN as estimator allows a more realistic modeling.

7 CONCLUSION

Several methods can be used to create power models with or without *a priori* on the resulting model. This article shows the methodology to obtain these two classes of models based on measurements. It also evaluates the precision of the state-of-the-art of predefined additive and of neural network models that can reach a few percent on a large variety of workloads, from Cloud to HPC.

This article provides an in-depth methodology needed to create an efficient neural network.

The methodology proposes a structure in four detailed steps, *Preprocessing data taking into account the distributed and complex monitoring infrastructure* in Section 3.1, *Learning dataset with a wide coverage* (Section 3.2), *Topology optimization* (Section 4.1), and *Variable reduction* (Section 4.3). The possibility to change each block of the approach with a better one while keeping the overall workflow is one of the good properties of this approach. One of the future works will be to evaluate different generic improvement and to compare them with the proposed dedicated one. As an example, we will evaluate Spearmint [50] for the network configuration to compare this generic method to the specific one described in this article.

The proposed methodology is particularly flexible and can be used on different types of architecture without user intervention. A future step will be to evaluate this methodology on heterogeneous architecture (multiple CPU and GPU) in which the classical aggregated model needs to be tweaked.

This article also proves that research based on learning methods must take into account measurement bias. The evaluated bias takes into account the power-supply losses that are dependent on the power load, the timing jitters between internal system measures and external power meter, outdated values from power meter and the transient state. By taking into account all these biases, the error of learned models is reduced by 1/3.

An open problem consists on the evaluation of the resulting models, which should be evaluated on current available processors, but also on machine-learning accelerators available in data-centers or in the one available as co-processor in recent generations of smartphones. A future work will be to evaluate the impact of network pruning [52] on the network resulting from our approach.

The proposed approach could also be used in other cases, such as performance modeling [28]. The main differences come from the fact that monitoring can be done in this case in a single node, compared to power modeling where an external node will monitor the consumed electricity.

REFERENCES

[1] Advanced Micro Devices Inc. 2013. *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors.* Advanced Micro Devices Inc. Rev 3.14.

[2] Sameer Alawnah and Assim Sagahyroon. 2017. Modeling of smartphones' power using neural networks. *EURASIP J. Embed. Syst.* 2017, 1 (2017), 22.

[3] Embedded Microprocessor, Ankush Varma, Eric Debes, Igor Kozintsev, and Bruce Jacob. 2005. Instruction-level power dissipation in the Intel XScale. In *Proceedings of the SPIE's 17th Symposium on Electronic Imaging Science & Technology.*

[4] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. 1991. The NAS parallel benchmarks—Summary and preliminary results. In *Proceedings of the ACM/IEEE Conference on Supercomputing (Supercomputing'91).* ACM, New York, NY, 158–165. DOI: https://doi.org/10.1145/125826.125925

[5] R. Basmadjian and H. De Meer. 2012. Evaluating and modeling power consumption of multi-core processors. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy'12).* 1–10.

[6] A. C. S. Beck, J. C. B. Mattos, F. R. Wagner, and L. Carro. 2003. CACO-PS: A general purpose cycle-accurate configurable power simulator. In *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI'03).* 349–354. DOI: https://doi.org/10.1109/SBCCI.2003.1232852

[7] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. 2010. PowerMon: Fine-grained and integrated power monitoring for commodity computer systems. In *Proceedings of the IEEE SoutheastCon.* IEEE, 479–484. DOI: https://doi.org/10.1109/SECON.2010.5453824

[8] Frank Bellosa. 2000. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop (EW'00).* ACM, New York, NY, 37–42. DOI: https://doi.org/10.1145/566726.566736

[9] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen. 1995. GROMACS: A message-passing parallel molecular dynamics implementation. *Comput. Phys. Commun.* 91, 1–3 (1995), 43–56. DOI: https://doi.org/10.1016/0010-4655(95)00042-E

[10] Ramon Bertran, Yolanda Becerra, David Carrera, Vicenç Beltran, Marc Gonzílez, Xavier Martorell, Nacho Navarro, Jordi Torres, and Eduard Ayguadé. 2012. Energy accounting for shared virtualized environments under DVFS using PMC-based power models. *Fut. Gen. Comput. Syst.* 28, 2 (Feb. 2012), 457–468. DOI: https://doi.org/10.1016/j.future.2011.03.007

[11] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. 2010. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing (ICS'10).* ACM, New York, NY, 147–158. DOI: https://doi.org/10.1145/1810085.1810108

[12] D. Brooks, V. Tiwari, and M. Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture.* 83–94.

[13] M. A. Castaño, S. Catalán, R. Mayo, and E. S. Quintana-Ortí. 2014. Reducing the cost of power monitoring with DC wattmeters. *Comput. Sci. Res. Dev.* 30, 2 (2014), 1–8. DOI: https://doi.org/10.1007/s00450-014-0262-z

[14] H. Chen, Y. Li, and W. Shi. 2012. Fine-grained power management using process-level profiling. *Sustain. Comput. Inf. Syst.* 2, 1 (2012), 33–42. DOI: https://doi.org/10.1016/j.suscom.2012.01.002

[15] Georges Da Costa, Jean-Marc Pierson, and Leandro Fontoura-Cupertino. 2017. Mastering system and power measures for servers in datacenter. *Sustain. Comput. Inf. Syst.* 15, Supplement C (2017), 28–38. DOI: https://doi.org/10.1016/j.suscom.2017.05.003

[16] G. Da Costa and H. Hlavacs. 2010. Methodology of measurement for energy consumption of applications. In *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID'10).* 290–297. DOI: https://doi.org/10.1109/GRID.2010.5697987

[17] T. Do, S. Rawshdeh, and W. Shi. 2009. pTop: A process-level power profiling tool. In *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09).*

[18] J. Dongarra and P. Luszczek. 2013. HPC challenge: Design, history, and implementation highlights. In *Contemporary High Performance Computing: From Petascale Toward Exascale*, Jeffrey Vetter (Ed.). CRC Press, Boca Raton, FL, 13–30.
[19] D. Economou, S. Rivoire, and C. Kozyrakis. 2006. Full-system power analysis and modeling for server environments. In *Proceedings of the Workshop on Modeling Benchmarking and Simulation (MOBS'06)*.
[20] S. Eranian. 2013. perf/x86: add Intel RAPL PMU support. *LWN*. Retrieved from http://lwn.net/Articles/573602/.
[21] J. Flinn and M. Satyanarayanan. 1999. PowerScope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*. 2–10. DOI: https://doi.org/10.1109/MCSA.1999.749272
[22] Imola K. Fodor. 2002. *A Survey of Dimension Reduction Techniques*. Technical report: UCRL-ID-148494. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory.
[23] Leandro Fontoura Cupertino. 2015. *Modeling the Power Consumption of Computing Systems and Applications Through Machine Learning Techniques*. PhD Thesis. Université Paul Sabatier, Toulouse, France.
[24] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K. W. Cameron. 2010. PowerPack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.* 21, 5 (May 2010), 658–671. DOI: https://doi.org/10.1109/TPDS.2009.76
[25] René Griessl, Meysam Peykanu, Jens Hagemeyer, Mario Porrmann, Stefan Krupop, Micha vor dem Berge, Thomas Kiesel, and Wolfgang Christmann. 2014. A scalable server architecture for next-generation heterogeneous compute clusters. In *Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC'14)*. IEEE, 146–153.
[26] Philipp Gschwandtner, Michael Knobloch, Bernd Mohr, Dirk Pleiter, and Thomas Fahringer. 2014. Modeling CPU energy consumption of HPC applications on the IBM Power7. In *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'14)*. IEEE, 536–543.
[27] Tom Guerout, Yacine Gaoua, Christian Artigues, Georges Da Costa, P. Lopez, and Thierry Monteil. 2017. Mixed integer linear programming for quality of service optimization in clouds. *Fut. Gen. Comput. Syst.* 71 (June 2017), 1–17. Retrieved from http://dx.doi.org/10.1016/j.future.2016.12.034.
[28] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.* 206 (2014), 79–111.
[29] M. Jarus, A. Oleksiak, T. Piontek, and J. Węglarz. 2014. Runtime power usage estimation of HPC servers for various classes of real-life applications. *Fut. Gen. Comput. Syst.* 36 (2014), 299–310. DOI: https://doi.org/10.1016/j.future.2013.07.012
[30] Mateusz Jarus, Sébastien Varrette, Ariel Oleksiak, and Pascal Bouvry. 2013. Performance evaluation and energy efficiency of high-density HPC platforms based on Intel, AMD, and ARM processors. In *Proceedings of the European Conference on Energy Efficiency in Large Scale Distributed Systems*. Springer, 182–200.
[31] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. 2010. Virtual machine power metering and provisioning. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'10)*. ACM, New York, NY, 39–50. DOI: https://doi.org/10.1145/1807128.1807136
[32] Yeseong Kim, Pietro Mercati, Ankit More, Emily Shriver, and Tajana Rosing. 2017. P 4: Phase-based power/performance prediction of heterogeneous systems via neural networks. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. IEEE, 683–690.
[33] M.-A. Lemburg. 2006. PYBENCH—A Python Benchmark Suite. Retrieved from http://svn.python.org/projects/python/trunk/Tools/pybench/.
[34] A. Lewis, S. Ghosh, and N.-F. Tzeng. 2008. Run-time energy consumption estimation based on workload in server systems. In *Proceedings of the Conference on Power Aware Computing and Systems (HotPower'08)*. USENIX Association, Berkeley, CA, 4–4. Retrieved from http://dl.acm.org/citation.cfm?id=1855610.1855614.
[35] Jiantang Li, Yongbin Zhou, Jiye Liu, and Hailong Zhang. 2011. An instruction-level software simulation approach to resistance evaluation of cryptographic implementations against power analysis attacks. In *Proceedings of the IEEE International Conference on Computer Science and Automation Engineering (CSAE'11)*, Vol. 2. 680–686. DOI: https://doi.org/10.1109/CSAE.2011.5952597
[36] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture*. ACM, 469–480.
[37] Jieun Lim, Nagesh B. Lakshminarayana, Hyesoon Kim, William Song, Sudhakar Yalamanchili, and Wonyong Sung. 2014. Power modeling for GPU architectures using McPAT. *ACM Trans. Des. Autom. Electron. Syst.* 19, 3 (2014), 26.
[38] Weiwei Lin, Wentai Wu, Haoyu Wang, James Z. Wang, and Ching-Hsien Hsu. 2016. Experimental and quantitative analysis of server power model for cloud data centers. *Fut. Gen. Comput. Syst.* 86 (2016). DOI: https://doi.org/10.1016/j.future.2016.11.034
[39] Somnath Mazumdar and Marco Pranzo. 2017. Power efficient server consolidation for cloud data center. *Fut. Gen. Comput. Syst.* 70 (2017), 4–16. DOI: https://doi.org/10.1016/j.future.2016.12.022

[40] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta. 2011. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the USENIX Technical Conference (USENIXATC'11)*. USENIX Association, Berkeley, CA, 12–12. Retrieved from http://dl.acm.org/citation.cfm?id=2002181.2002193.

[41] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier. 2012. A preliminary study of the impact of software engineering on GreenIT. In *Proceedings of the 1st International Workshop on Green and Sustainable Software (GREENS'12)*. 21–27. DOI: https://doi.org/10.1109/GREENS.2012.6224251

[42] NVidia Corp. 2012. *NVML API Reference Manual*. Version 3.295.45.

[43] Kenneth O'Brien, Ilia Pietri, Ravi Reddy, Alexey Lastovetsky, and Rizos Sakellariou. 2017. A survey of power and energy predictive models in HPC systems and applications. *ACM Comput. Surv.* 50, 3, Article 37 (June 2017), 38 pages. DOI: https://doi.org/10.1145/3078811

[44] Nikola Rajovic, Alejandro Rico, James Vipond, Isaac Gelado, Nikola Puzovic, and Alex Ramirez. 2013. Experiences with mobile processors for energy efficient HPC. In *Proceedings of the Conference on Design, Automation, and Test in Europe*. EDA Consortium, 464–468.

[45] F. Rawson. 2004. *MEMPOWER: A Simple Memory Power Analysis Tool Set*. Research Report. IBM.

[46] S. Rivoire, P. Ranganathan, and C. Kozyrakis. 2008. A comparison of high-level full-system power models. In *Proceedings of the Conference on Power Aware Computing and Systems (HotPower'08)*. USENIX Association, Berkeley, CA, 3–3. Retrieved from http://dl.acm.org/citation.cfm?id=1855610.1855613.

[47] E. Rotem, A. Naveh, A. Ananthakrishnan, D. Rajwan, and E. Weissmann. 2012. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *IEEE Micro* 32, 2 (2012), 20–27. DOI: https://doi.org/10.1109/MM.2012.12

[48] H. Shafi, P. J. Bohrer, J. Phelan, A. A. Rusu, and J. L. Peterson. 2003. Design and validation of a performance and power simulator for PowerPC systems. *IBM J. Res. Dev.* 47, 5.6 (Sept. 2003), 641–651. DOI: https://doi.org/10.1147/rd.475.0641

[49] A. J. Smola and B. Schölkopf. 2004. A tutorial on support vector regression. *Stat. Comput.* 14, 3 (2004), 199–222. DOI: https://doi.org/10.1023/B:STCO.0000035301.49549.88

[50] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the International Conference on Advances in Neural Information Processing Systems*. 2951–2959.

[51] Shuaiwen Song, Chunyi Su, Barry Rountree, and Kirk W. Cameron. 2013. A simplified and accurate model of power-performance efficiency on emergent GPU architectures. In *Proceedings of the IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS'13)*. IEEE, 673–686.

[52] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[53] Shufeng Tan and Michael L. Mayrovouniotis. 1995. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal* 41, 6 (1995), 1471–1480.

[54] The Apache Software Foundation. 1997–2014. Apache HTTP Server Project. Retrieved from http://httpd.apache.org/.

[55] The OpenSSL Project. 1999–2014. OpenSSL: Cryptography and SSL/TLS Toolkit. Retrieved from http://www.openssl.org/.

[56] Robert Tibshirani. 1996. Regression shrinkage and selection via the Lasso. *J. Roy. Stat. Soc. Series B (Methodol)* 58, 1 (1996), pp. 267–288. Retrieved from http://www.jstor.org/stable/2346178.

[57] Ananta Tiwari, Michael A. Laurenzano, Laura Carrington, and Allan Snavely. 2012. Modeling power and energy usage of HPC kernels. In *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW'12)*. IEEE, 990–998.

[58] John Tsiombikas. 2016. C-Ray Simple Raytracing Tests. Retrieved from http://www.futuretech.blinkenlights.nl/c-ray.html.

[59] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. 2000. Energy-driven integrated hardware-software optimizations using SimplePower. *SIGARCH Comput. Archit. News* 28, 2 (May 2000), 95–106. DOI: https://doi.org/10.1145/342001.339659

[60] Haifeng Wang and Yunpeng Cao. 2015. Predicting power consumption of GPUs with fuzzy wavelet neural networks. *Parallel Comput.* 44 (2015), 18–36.

[61] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and S. Malik. 2002. Orion: A power-performance simulator for interconnection networks. In *Proceedings of the 35th IEEE/ACM International Symposium on Microarchitecture (MICRO'02)*. 294–305. DOI: https://doi.org/10.1109/MICRO.2002.1176258

[62] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. 2012. Measuring energy and power with PAPI. In *Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW'12)*. 262–268. DOI: https://doi.org/10.1109/ICPPW.2012.39

[63] M. Witkowski, A. Oleksiak, T. Piontek, and J. Weglarz. 2013. Practical power consumption estimation for real life HPC applications. *Fut. Gen. Comput. Syst.* 29, 1 (2013), 208–217. DOI: https://doi.org/10.1016/j.future.2012.06.003 Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[64] Hailong Yang, Qi Zhao, Zhongzhi Luan, and Depei Qian. 2014. iMeter: An integrated VM power model based on performance profiling. *Fut. Gen. Comput. Syst.* 36 (2014), 267–286. DOI: https://doi.org/10.1016/j.future.2013.07.008

[65] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. 2003. Modeling hard-disk power consumption. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*. USENIX Association, Berkeley, CA, 217–230. Retrieved from http://dl.acm.org/citation.cfm?id=1090694.1090722.

[66] Xinnian Zheng, Lizy K. John, and Andreas Gerstlauer. 2016. Accurate phase-level cross-platform power and performance estimation. In *Proceedings of the 53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. IEEE, 1–6.