Evaluating the Effectiveness of Model-Based Power Characterization

John C. McCullough, Yuvraj Agarwal, Jaideep Chandrashekar[†]
Sathyanarayan Kuppuswamy, Alex C. Snoeren, and Rajesh K. Gupta

UC San Diego and [†]*Intel Labs, Berkeley*

Abstract

Accurate power characterization is important in computing platforms for several reasons ranging from power-aware adaptation to power provisioning. Power characterization is typically obtained through either direct measurements enabled by physical instrumentation or modeling based on hardware performance counters. We show, however, that linear-regression based modeling techniques commonly used in the literature work well only in restricted settings. These techniques frequently exhibit high prediction error in modern computing platforms due to inherent complexities such as multiple cores, hidden device states, and large dynamic power components.

Using a comprehensive measurement framework and an extensive set of benchmarks, we consider several more advanced modeling techniques and observe limited improvement. Our quantitative demonstration of the limitations of a variety of modeling techniques highlights the challenges posed by rising hardware complexity and variability and, thus, motivates the need for increased direct measurement of power consumption.

1 Introduction

Electrical power is a precious resource and its consumption is important to all forms of computing platforms from handheld devices to data centers. Numerous research efforts seek to optimize and carefully manage energy consumption at multiple levels—starting from individual components and subsystems such as wireless radios [27], storage devices [23], and processors [15, 28], to entire platforms [16, 39].

An important goal of these optimizations has been to achieve power proportionality, that is, power consumption that is proportional to the computational work done. As a result, a modern system's instantaneous power draw can vary dramatically. Moreover, the exact relationship between power draw and activity level is becoming increasingly complex due to the advent of microprocessors with multiple cores and built-in fine-grained thermal control, as well as hidden device states that are not necessarily exposed to the operating system.

While increasingly energy efficient components are a key enabler, achieving the goal of platform-wide power proportionality requires an intelligent dynamic power management (DPM) scheme. A critical first step towards that goal is to characterize how much power is being consumed, both by the platform as a whole and also by individual subsystems. Armed with a reasonable characterization of power consumption, a DPM system then needs to accurately predict how changes in utilization will impact future power consumption. For example, a DPM system can guide resource allocation decisions between heterogeneous but functionally similar resources such as multiple radios on the same platform [27].

Current approaches to prediction rely on assumptions of power proportionality to make architectural or system-level tradeoffs. These tradeoffs are evaluated by building a model of the system that describes power consumption in terms of power states and attributing the model to one or more observable hardware and software counters, correlating these with changes in measured power draw. Previous work on power modeling has focused on modeling total system power consumption using several learning techniques such as linear regression [21, 31], recursive learning automata [25], and stochastic power models [29]. The effectiveness of each of these techniques has been evaluated independently across various benchmarks (see [19] for a review).

However, we are unaware of any definitive comparison of these models for a diverse set of benchmarks on a given platform. Further, the increasingly nuanced relationship between a component's activity level and power consumption limits the utility of published models in modern systems. In particular, as we shall show, these models are effective only in a restricted set of cases, e.g., when system utilization is constant and very high, for

relatively straightforward executions in single cores, or when the system's static power consumption is dominant and the dynamic component is within the margin of error. Consequently, even well-designed DPM algorithms employing these models will make suboptimal decisions (to shutdown and/or slowdown components) if the actual utilization dynamically changes the significance of various components to overall power consumption.

To overcome the limited ability of models to predict power consumption, manufacturers often build reference systems complete with a large number of sense resistors and use precision analog-to-digital converters (ADC) to create a measurement instrument that accurately captures power consumption at fine time granularities. Research efforts have mimicked this approach by developing custom designs that can breakdown power consumption within sensor platforms [35] or monitor whole-system power for general purpose computers at the power supply [10]. While such extensive direct instrumentation can provide accurate power measurements, it requires significant design effort and increases costs due to board space constraints and the need for additional components. Hence, we are not aware of any production systems so instrumented. Moreover, significant increases in cross-part variability [13, 38] limit the applicability of predictions based upon even the measured behavior of a single or small number of reference systems.

We evaluate the need for pervasive power instrumentation by exploring the effectiveness of power modeling on modern hardware. Mindful of the fact that the required level of accuracy varies based upon the specific DPM goal, we consider how well increasingly sophisticated models can predict the power consumption of realistic workloads. We make the following contributions:

- We show empirically that while total system power can be modeled with 1–3% mean relative error across workloads when restricted to a single core, it rises to 2-6% for multi-core benchmarks.

- We find that linear models have significantly higher mean relative error for individual subsystems such as the CPU: 10–14% error on average, but as high as 150% for some workloads. We employ more complex techniques to improve predictive performance, but only by a few percent.

- We present an in-depth analysis of why modeling fails for modern platforms, especially under multi-core workloads. We posit that this poor predictive performance is due to effects such as cache contention, processor performance optimizations, and hidden device states not exposed to the OS. In addition, we present quantitative evidence of significant variability between identical components, which fundamentally bounds the potential accuracy of any modeling based approach.

Taken together these results make a strong case for pervasive instrumentation and dynamic collection of power usage data. While traditionally eschewed due to significant increases in design and manufacturing costs, the advent of relatively inexpensive ADCs and associated circuits makes such an approach increasingly feasible.

2 Related work

Researchers have long been interested in optimizing the energy efficiency of computing devices. In the context of mobile platforms, researchers have considered optimizing individual subsystems such as the CPU [15, 28], disk [23] and wireless radios [27]. While much of the early work focused on battery powered devices for usability reasons [16, 27, 39], economic motivations have dramatically increased interest in general purpose computing such as PCs and servers [1, 3, 26].

There have been a number of efforts to predict energy consumption through in-situ power measurements by adding different levels of hardware instrumentation. The Openmoko Freerunner mobile phone platform was designed to support developer access. Carroll and Heiser leveraged its sense-resistor support to characterize power consumption at a component level, deriving simple, time-based linear models for each component [7]. The LEAP platform [35] proposes adding fine-grained instrumentation to all power rails in embedded sensor nodes, and develops the required software instrumentation within the OS kernel to attribute energy to running tasks. In contrast, Quanto [17] proposes a single point of measurement by observing the switching regulator on sensor platforms to increase the practicality of energy attribution. Quanto requires changes to TinyOS and individual applications to track individual state transitions and offline analysis for energy attribution.

These measurement efforts, however, are restricted to special-purpose platforms with limited availability. Moreover, while such detailed, instrumentation-based approaches provide accurate power characterization, they are frequently regarded as too costly to implement at scale. Hence, efforts focused on more general purpose platforms have typically relied upon modeling, mostly for predicting total system power.

Recent activity in server environments is driven by the observation that most computer systems are largely idle [1] or exhibit low utilization except during peak traffic [3]. Hence, Barroso and Hölze argue for more energy-proportional designs so that the energy used is propor-

tional to the utilization [3]. This quest for energy proportionality has led to a variety of low-power server designs [2]. Some applications, however, have been shown to perform poorly on these low-power platforms [30].

Predicting the energy use of a particular application on a general-purpose platform is challenging, however. Previous studies have shown that CPU utilization is not useful by itself and that performance counters used to build models only help if they are correlated with dynamic power [24, 31]. Economou et al. explored the use of component-level measurements including CPU, memory and hard disk but were unable to significantly decrease prediction error [14]. Conversely, Bircher and John explored using performance counters to model not only total system power, but component-level power (e.g., CPU, memory, disk, and I/O) as well in a Pentium IV-class system [5]. They find that linear models are a poor fit for all but the CPU, and instead resort to multiple-input quadratics to provide an average of 9% absolute error.

In the hosted setting, Koller et al. turn to application-level throughput—as opposed to hardware performance counters—to improve power prediction for virtual clusters using linear combinations [22]. The Joulemeter project [21] proposes combining total-system power measurements from server power supplies with power modeling to attribute power consumption at the coarse level of individual virtual machines (VM) running on a physical server. Their model is generated by exercising specific VMs by running particular applications and using the CPU utilization metric to attribute energy.

Despite prediction errors, a number of researchers have demonstrated the utility of power modeling. For example, Bircher and John combine CPU metrics with instantaneous voltage demand information to ensure that the processor uses the correct DVFS setting and achieve an average speedup of 7.3% over Windows Vista's default DVFS algorithm [6]. Wang and Wang utilize feedback control theory to jointly optimize application-level performance and power consumption [37], while Tolia et al. improve power proportionality by leveraging virtual machines, DVFS, and fan control [36].

3 Power characterization

Characterizing the power consumption of a computing platform need not be difficult in principle. Ideally, original equipment manufacturers (OEMs) are well positioned to add extensive power instrumentation to their platforms, which would enable accurate and fine grained power measurements. Combined with such instrumentation, OEMs could further expose interfaces to an operating system to query detailed power information in a low-overhead manner. This information can then be used by the OS as well as individual applications to manage their energy consumption dynamically. Unfortunately, this ideal scenario is not realized in practice due to manufacturing constraints such as increased board area, cost of components and design costs. Modern platforms are already extremely complex and OEMs are reluctant to add functionality without clear and quantifiable benefits. Hence, while OEMs may have extensive power instrumentation on their *development* platforms during design and testing, we are unaware of any commodity platform that provides fine-grained power measurement capabilities in hardware.

Instead, in the absence of direct power measurement, the commonly used alternative is to make power models. The basic idea behind power modeling is to take as input various software and hardware counters and use those to predict power consumption after suitable training on an appropriately instrumented platform. Regardless, any power characterization approach, whether based upon modeling or direct instrumentation, must trade off between several design alternatives as discussed below.

3.1 Measurement granularity

One of the dimensions that affects both forms of power characterization—power measurement and power modeling—is the granularity of measurement. Power can be characterized at the level of an entire system (a single power value) or can be done at a logical subsystem granularity, such as the display, CPU, memory and storage subsystems. The appropriate measurement granularity depends on the application. For example, in data centers an application for macro-scale workload consolidation on servers will likely only require total system power measurements at an individual server level. On the other hand, fine-grained scheduling decisions on individual heterogeneous processor cores requires power consumption data for individual cores.

While total system power can be measured at the wall socket directly using myriad commercial devices (e.g. `WattsUP` meters) the applicability is limited in the case of any fine grained adaptation. Furthermore, power measurements at the system level cannot distinguish between the actual power used and the power wasted due to the inefficiencies in the power supply. On the other hand, fine-grained subsystem level power characterization is more useful since the total system power can still be estimated accurately by adding the power consumption of individual components. Most of the research to date has focused on total system power modeling [14, 31]. In this paper we explore the design space of power characterization and especially investigate the feasibility of accurate power modeling at subsystem granularity.

3.2 Accuracy

In the case of power instrumentation, it is possible to get very high levels of accuracy with precision analog-to-digital converters (ADCs), albeit at higher costs. The accuracy of power modeling with respect to the ground-truth measurements is important since the accuracy depends on how well the model fits. Similar to the measurement granularity dimension, the accuracy requirements of power modeling are also somewhat dependent on the application. When consolidating workload to fewer servers in a data center, modeling total power within 5–10% error is sufficient to guide policy decisions since the base power of servers is high [14, 31]. On the other hand, for an application of fine grained scheduling on different heterogeneous processor cores, the accuracy of power characterization needs to be at least as good as the differences in power consumption between the cores, otherwise scheduling decisions may be incorrect.

Furthermore, the required accuracy is likely to vary with particular subsystems based on factors such as their dynamic range, and their contribution to the total system power. Subsystems that are dominant in particular platforms need to be measured more accurately since the penalty of mis-predicting the power is higher. Furthermore, subsystems that are more complex and dynamic, such as processors, need higher accuracy measurements. On our test systems, the CPU consumes between 0.5W and 27W (constituting up to 40% of the total system power), and prediction errors translate to high absolute error. In contrast, an SSD disk drive on the SATA interface, or the network interface, consumes lower power (fewer than 2–3W) and is less dynamic; therefore, higher modeling errors can be tolerated. However, in cases of platforms with a larger number of disks this modeling error will have a more significant impact.

3.3 Overhead and complexity

Both power modeling and power instrumentation have associated overheads and complexity. In the case of power instrumentation, OEMs have to integrate the power measurement functionality into their platforms, usually in the form of a shunt resistor connected to ADCs. The ADCs measure the voltage drop across the shunts, which is converted into current and power consumption. Since modern platforms have multiple voltage domains, and subsystems can be powered using +3.3V, +5V, and +12V power supplies, a large number of ADC inputs are required. Furthermore, in case the voltage to the subsystem is not constant, such as with processors that employ dynamic voltage scaling, we need additional ADC inputs to measure voltage as well. Some subsystems, such as processors, can also have multiple power lines powering different functional units within them which each need to be measured separately. The shunt resistors themselves need to be chosen while keeping in mind the dynamic range of the power consumed by individual subsystems, possibly from several milliwatts to tens of watts, to give high precision and low power loss in the sense resistor itself. All of these factors contribute to higher costs—of components, board area and design, test and validation time.

Complexity in power modeling arises in part from the need to capture all the relevant features expressed by software and hardware counters to serve as inputs to build the models. Often platforms and components, such as the CPU, either support tracking a limited set of platform counters simultaneously, or have-non trivial overhead in collecting a large set of counters at fine granularities. Therefore, it is important that the model be sparse and use as few counters and states are possible, while still providing reasonable modeling accuracy. Additionally, the models themselves can be arbitrarily complex and can require non-trivial amounts of computation. In this paper, we explore a series of increasingly complex models to understand the accuracy/complexity tradeoffs.

Finally, in the case of power modeling, transferability or robustness of the power modeling is key: The model should be generated once and should be applicable over time and to other instances of the same platform. While we believe that this is intuitively the case, recent work has highlighted a significant amount of platform heterogeneity, where process and manufacturing variations and aging effects lead to identical hardware exhibiting significant variation in power consumption [8, 13, 38]. We show some preliminary results relating to this aspect and highlight the associated challenges with power modeling in the face of increasing variability in hardware.

4 Power modeling

While there has been a considerable volume of work in the area of power modeling, notably for system power and CPU power, a common toeu hread joining most of the previous work is the assumption that system power can be well predicted by simple linear regression models [14, 21, 31]. Our goal in this paper is to understand whether (i) these simple models are compatible with more contemporary platforms (CPU and platform complexity has increased significantly since many of the previous modeling approaches were proposed), and (ii) whether these models can be applied to individual subsystems within platforms. The latter is important to understand because, with the increasing emphasis on power proportionality and energy awareness, there are several adaptations that can be done at the platform level as well

as the subsystem level, provided fine-grained power consumption information is available.

Our initial attempts to use simple linear regression models—including replicating specific ones previously proposed—were disappointing: The models perform poorly on non-trivial workloads. This result could be explained by one of the following reasons:

- The features being fed into the model contain a certain level of cross-dependency, whereas linear models assume feature independence.
- The features used by previously published models are no longer appropriate for contemporary platforms. There may, however, exist a different set of counters that can still lead to a good model.
- Modern hardware components, such as processors, abstract away hardware complexity and do not necessarily expose all the power states to the OS and are thus fundamentally hard to model since changes in power consumption are not necessarily associated with changes in exposed states.

In this section, we describe a number of increasingly complex regression models that we use to fit the power data. Unfortunately, we found that increasing the complexity of the model does not always improve the accuracy of power prediction across the different subsystems that we are trying to model. We begin by describing how linear regression models are constructed and enumerating the specific models we use.

4.1 Linear regression models

Let $\mathbf{y} = [y_1, y_2, \ldots, y_k]$ be the vector of power measurements and let $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ be the normalized vector of measurements on the n variables (hardware counters and OS variables) collected at time t_i. The linear regression model is expressed as:

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_n x_{i,n} + \epsilon_i,$$

where ϵ_i is a noise term that can account for measurement error. Thus, the linear regression models are solved by estimating the model parameters β, and this is typically done by finding the least squares solution to $\hat{\beta} = \mathbf{y}.\mathbf{X}^{-1}$, which can be computed as

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{k} \left(y_i - \beta_0 - \sum_{j=1}^{n} \beta_j x_{i,j} \right)^2,$$

or simply $\hat{\beta} = \operatorname{argmin}_\beta \left(||\mathbf{y} - \beta \mathbf{X}||_2^2 \right)$.

The challenge in building a good power model is to correctly identify the set of n most relevant features. On the platforms we considered, there are in excess of 800 different hardware counters that can be tracked (even though only a few can be tracked *simultaneously*). Previous work has overcome this problem by using domain knowledge about the platform architecture to hand pick counters that are believed to be relevant [14, 31]. We believe that increasing complexity in modern processors and platforms makes this task harder with each generation. To understand whether such domain knowledge is critical to power modeling, we also use modeling techniques that perform automatic feature selection in the process of constructing a model. We observe that the features selected by the more complex techniques correspond to the features with the highest mutual information [11] for a given power rail. This makes us confident that these state-of-the-art modeling techniques are leveraging all relevant features and are not missing anything that is relevant but not linearly correlated.

We now briefly describe the regression techniques that we explore, listed in order of increasing complexity.

MANTIS: Ecomomou et al. developed a server power model by fitting a linear regression of four distinct utilization counters obtained from different platform subsystems to the power measurements taken at a wall socket [14]. The input utilization metrics are obtained by running a number of systematic workloads that stress the platform subsystems in sequence.

In particular, they consider counters corresponding to CPU utilization, off-chip memory accesses, and hard drive and network I/O rates. We extend the basic MANTIS linear model to also consider instructions per cycle as a representative baseline obtained from a best-in-class full-system power model [31]. While there have been a large number of efforts focused on identifying a suitable set of performance counters, we choose the MANTIS model as a starting point because it has been shown to have very good predictive properties on previous-generation hardware [14]. Since a few of the counters used by the original MANTIS model are no longer available on modern platforms, we communicated with the authors themselves to find appropriate substitutes.

Lasso regression: There are two drawbacks to building a (linear) regression model. First, it requires domain knowledge to identify the correct set of features (possibly from a large space)—in this case, we seek features possibly related to power consumption. Second, when the features are correlated to each other, the regression simply distributes the coefficient weights across the correlated features, and all correlated features are included in the final model, rather than identifying a smaller subset of somewhat independent features. In current complex platforms, there is a very large space of features that can be measured and it is a non-trivial task—even

for an expert—to correctly identify the smallest possible subset of power relevant features [33]. Lasso regression, which is a specific instance of l_1-regularized regression, overcomes this challenge by penalizing the use of too many features. Thus, it tends to favor the construction of sparse models which incorporate just enough features as are necessary. This is done by incorporating a penalty factor into the least-squares solution for the regression, which is expressed as:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left(\sum_{i=1}^{k} \left(y_i - \beta_0 - \sum_{j=1}^{n} \beta_j x_{i,j} \right)^2 + \lambda \sum_{j=1}^{d} |\beta_j| \right)$$

or, simply $\hat{\beta} = \operatorname{argmin}_\beta \left(||\mathbf{y} - \beta \mathbf{X}||_2^2 + \lambda \, ||\beta||_2 \right)$.

Here, λ is a penalty function that encourages the solution to aggressively set β values to zero (and exclude the associated features from the model). Compared to regular methods, Lasso regression is advantageous since it relies less on strong domain knowledge to pick out the right features; in addition, it is computationally simple, and automatically picks models with a small number of features, which are critical requirements for a usable power model. The optimal value for the λ parameter is selected by cross validation on the training data. We used the `glmnet` package to perform the Lasso regression [18].

4.2 Non-linear regression models

Linear regression models work well when the features being modeled are independent of each other and tend to predict poorly when there are interdependencies between the modeled features; non-linear models can often capture these feature dependencies. (Indeed, previous work has shown that quadratic models can be more effective at modeling subsystem power [5].) The non-linear form of the model can be expressed as:

$$y_i = \beta_0 + \sum_{\ell=1}^{m} \beta_\ell \phi_\ell(x_i) + \epsilon_i,$$

where ϕ_ℓ are non-linear basis functions over the feature vectors $\mathbf{x}_i$. We use the Lasso least-squares formulation as before to solve the regression and construct a model.

In general, the set of possible ϕ_j is arbitrarily large and solutions exist for only a few families. We experiment with three well known functions:

Polynomial with Lasso: Here, the basis functions are defined as exponentiated forms of the original variables. So, $\phi = \{x_i^a : 1 \leq a \leq d\}$ where $d = 3$. Again, with Lasso, only the relevant features—now including the polynomial terms which may have cross dependencies—are inserted into the model.

Polynomial + exponential with Lasso: In this slight variation of the previous model, ϕ also includes the functions e^{x_i}. As before, we run the full set of terms through the Lasso (linear) regression package which picks out a sparse subset of the terms. In the previous case as well as this one, the optimal λ is selected by cross validation.

Support vector regression (SVR): We also experiment with support vector machine (SVM)-based regression. At a high level, SVMs operate by fitting a hyperplane decision boundary to a set of labeled data instances taken from different classes. The hyperplane boundary is constructed so as to maximize the separation between the two classes of data, and can be used to perform classification, regression, and function estimation tasks on the data. SVMs employ a trick to handle non-linearity, the data is run through a non-linear kernel that maps the data to a higher dimensional space where greater separation may be achieved. An important difference between SVR and Lasso-based methods is that SVR does not force the regression to be sparse. When features are correlated, weights are distributed across them. We use the `libsvm` [9] and, in particular, the radial basis kernel. The parameters required for the RBF kernel were optimally selected by cross validation on the training data.

5 Evaluation setup

We collect platform subsystem power using an instrumented Intel Calpella platform. The platform is a customer reference board that corresponds to the commercially available mobile Calpella platform. This particular board, which is based on the Nehalem processor architecture, was outfitted with an Intel quad-core i7-820QM processor, 2x2GB of DDR3-1033 memory and a SATA SSD drive, running the 2.6.37 Linux kernel. Importantly, we turned off HyperThreading and TurboBoost on the platform to avoid hidden states (these states change operating points but are controlled in hardware and the OS has little visibility into them). A salient feature of this particular board is that it has been extensively instrumented with a very large number of low-tolerance power sense resistors that support direct and accurate power measurements of various subsystems (by connecting to a data acquisition system). The platform contains over one hundred sense resistors and it is a non-trivial task to collect readings from all of them. Instead, we first identified the platform subsystems that were of interest to us and simply instrumented the resistors for those subsystems and connected them to two 32-input National Instrument USB6218 DAQs. Finally to measure the total power consumption at the wall we use a commercial WattsUp meter. The Calpella platform is powered soley by a 12-V input from the ATX power connector and we consider

Subsystem	# resistors	min–max
CPU core	3	0.5–27W
CPU uncore (L3, mem. controller)	1	1–9W
integrated graphics	2	n.a.
discrete graphics	2	≈15.3W
memory	2	1–5W
CPU fan	1	≈0.7W
SATA	3	1.3–3.6W
LAN	1	≈ 0.95W
Chipset + other	0	0.5–5W
12V ATX in	1	23–67W

Table 1: Power characterization for the calpella platform. The 500-W ATX PSU that we use dissipates 20–26W due to conversion inefficiency and is not shown.

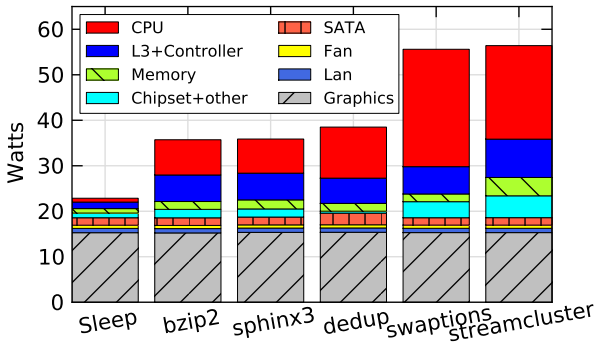

Figure 1: Power breakdown for sample workloads.

the 12-V rail to represent the total system power in order to eliminate the influence of variable power-supply inefficiencies. The power breakdown of the various subsystems is shown in Table 1.

The 16-bit NI-DAQs have a worst case error of 0.02%, but to scale the measured voltages to the range of the NI-DAQ we use precision voltage dividers, which in turn introduce a 0.035% measurement error, leading to an overall error of 0.04%. To minimize measurement overhead, the data from the NI-DAQs and the WattsUp meter is collected on a separate computer.

Along with the (externally) collected power readings, we collect OS-level statistics, hardware states and performance counters from the platform itself (for simplicity we will use the terms *counter* and *state* interchangeably). We extract OS-level statistics from `/proc` and `/sys` in Linux; these include processor utilization, disk usage, and processor C-state residency statistics. We collect hardware performance counters using the Linux `perf_event` framework.

By default the `perf_event` framework provides access to the four programmable and the six fixed hardware counters available per core. As a departure from previous processor models, Nehalem processors introduce "uncore" counters, which measure the performance of the L3 cache, QPI bus, and the memory controller. To replicate the MANTIS model, we need to measure last-level cache misses in the L3 cache. Fortunately, we have a kernel patch that provides access to the eight programmable per-socket performance counters. The measurement framework reports a total of 884 counters. While it would be ideal to measure them all concurrently and allow the models to pick out the most relevant features, the small number of programmable counters that can be read concurrently makes this task impossible.

Instead, we use a simple heuristic to reduce this number to a more manageable size: we sweep through the entire set of possible counters, making sure to get at least one run for each counter; then we compute the correlation of each counter with the total system power and discard all the counters that show no variation (with power), or that have very poor correlation. This brings down the set of potential counters to about 200, which is still large. To bridge the gap, we select all of the OS counters (these can be measured concurrently), and we greedily add as many hardware counters, in order of their correlation coefficients, as we can measure concurrently.

Note that due to issues with the aging OS required for NI-DAQ driver support, our test harness is initiated from an external machine. A high-level diagram of the measurement setup is shown in Figure 2. We have set up the NI-DAQ to sample each ADC channel at 10Khz and output average power consumption for each subsystem once per second for accurate power measurements. In our setup, we thus collect power readings as well as the on-platform measurements at a one-second granularity. Adjusting the collection granularity does not appreciably impact the prediction accuracy, and we feel that one second is a reasonable compromise: sampling the OS level counters at a faster rate would incur a higher overhead and introduce stronger measurement artifacts (where the act of measuring itself takes a non-trivial amount of power), while sampling it any slower might limit how quickly applications can react to changes in power consumption.

5.1 Benchmarks

To systematically exercise all possible states of the platform, particularly the subsystems that we are measuring, we selected an array of benchmarks from two well known benchmarking suites, as well as a few additional benchmarks to extend the subsystem coverage. We include the majority of the benchmarks in the SpecCPU benchmark suite [34]. We include 22 of the 32 benchmarks, excluding ten because they would either not com-

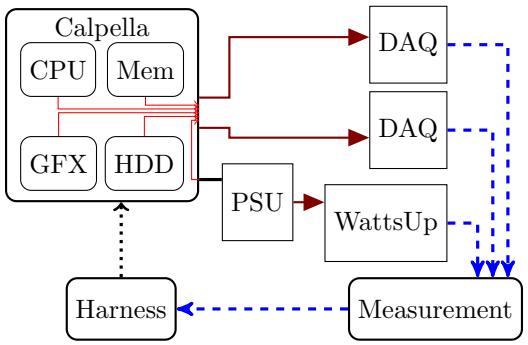

Figure 2: Test harness configuration.

pile with a modern compiler or tended to exhaust memory. While the SpecCPU suite is a well established benchmark, it only contains single-threaded benchmarks. To get a more representative set of benchmarks that would better exercise our multi-core system, we included the PARSEC [4] benchmark suite; this consists of a range of multi threaded "emerging workloads", including file de-duplication and x264 compression. We also include the `Bonnie` I/O benchmark, a parallel `LinuxBuild` kernel compile, `StressAppTest` [32], a `memcached` workload, as well as a synthetic `cpuload` benchmark. The StressAppTest program is a burn-in program designed to place a realistic high load on a system to test the hardware devices. We observe that while most of these benchmarks use the hardware as quickly as possible, evidence suggests that systems are not always fully loaded [3]. To capture this behavior, we supply `memcached` with a variety of different request rates to target different utilizations, and we duty cycle our synthetic floating point `cpuload` benchmark. Finally, we include `Sleep` to represent the system at idle.

5.2 Modeling evaluation

The effectiveness of a model is often decided by learning the model from a training set of data, and then assessing its predictive performance on a (different) testing set. 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 (a form of this was done in [31]). However, this is hard to achieve when systems are complex and have a large operating space. When we tried such an approach, the results were disappointing and led us to ask a more basic question: how well does the model work when the testing and training data are similar? This puts the focus on whether good models can be generated at all, rather than picking the smallest set of workloads needed to construct a good model. We employ a well known validation technique known as $k \times 2$ cross-validation. For this technique, we randomize the ordering of the data (collected from all the benchmarks), partition into two halves, use the first half as training data and learn the model, and then compute the prediction error on the latter half. The process is repeated multiple times (we repeat 10 times) and the errors from each run are aggregated.

We note that in our experimental evaluation, the error observed on the testing data set is approximately the same as that on the training data set, not only when the error is low, but also when the error is high. This raises our confidence that the models obtained each time are sufficiently general. In the next section, we present results from building models on specific platform subsystems.

6 Results

In this section we present results from evaluating the various models on several different operating configurations. The metric we use to test the efficacy of a model is *mean relative error*, which we shorten to *error* in the discussion, defined as follows:

$$error = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{p}_i - p_i}{p_i}\right|$$

where $\hat{p}_i$ is the model predicted power, and p_i is the actual power measurement. For the results shown in this section, we execute each benchmark and configuration five times ($n = 20$). We note that the metric used is consistent with previous work [31]. One point of departure with previous work is that we measure "system power" after the PSU, and hence we capture a more accurate reflection of the actual power being consumed.

After running a large number of experiments across a variety of configurations, we can reduce the findings into three takeaways, which we discuss next.

6.1 Single core

To reduce number of variables we first limit the system to use one processor core only and run all the benchmarks on that single core. HyperThreading and TurboBoost were also turned off, and the processor P-state (frequency) was fixed. This is a reasonable approximation to the systems that were used to develop the MANTIS model. For this configuration, we note the following: the mean prediction error (averaged over all the benchmarks to obtain a single number) for total system power is between 1–3%. The errors are also low for platform subsystems: for example the average error in CPU power

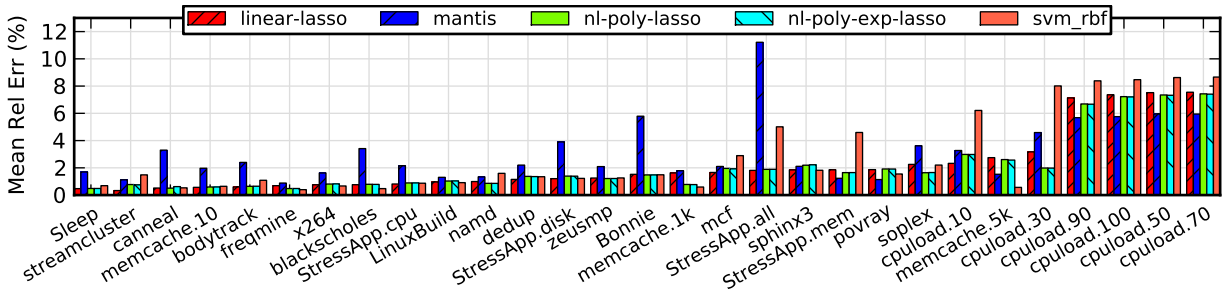

Figure 3: Modeling acuracy for total system power (Single-Core). Mean relative error is 1–3% across workloads.

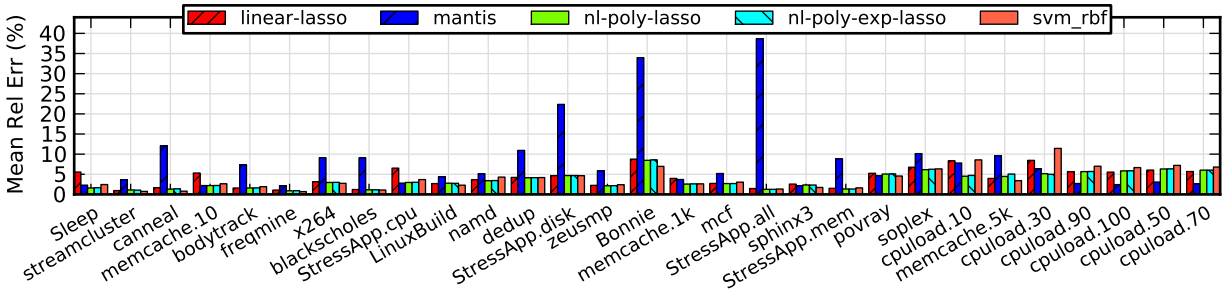

Figure 4: Modeling acuracy for CPU power (Single-Core). Mean relative error is 2-6% across workloads.

prediction is 2–6%, and the average error in predicting memory power is about 4–8% across the different models. Due to space constraints we do not show the results for the other subsystems, however the prediction errors is similar to that of the CPU.

In this context, mean prediction error can be misleading because the mean is computed over all time samples and is weighted towards longer-running benchmarks. Thus, the error will be worse if the system is executing a specific workload that is poorly predicted.

Figure 3 compares the prediction error in total system power over all the benchmarks used. The models do rather well, the majority of benchmarks show an error of less than 5%. The MANTIS model does well (which we expected), but the linear-Lasso model does slightly better than do the non-linear Lasso models. Upon closer inspection we observe that this ordering of results is tied to how well the models predict the CPU power, which along with a considerable base power factor, is a large contributor to the total system power.

Figure 4 compares the prediction errors across different models for the CPU subsystem power. Here, the differences between the models are more pronounced. We see that the MANTIS model is off by at most a few percent for most of the benchmarks, except for `canneal`, `StressApp`, and `bonnie`, which have high utilization, low IPC, and consequently lower power than the model attributes to utilization alone. It is important to note that the linear-Lasso model, which picks out the set of features automatically, consistently outperforms the MANTIS model, which uses domain knowledge to select the features. Not surprisingly, the set of counters picked out by linear-Lasso is a superset of the counters used by the MANTIS model; the C-state counters included in the linear-Lasso model, but not the MANTIS model seem to improve predictive power. Thus, this goes to establish that as systems become increasingly complex, the task of applying domain knowledge to pick out the most accurate set of counters becomes progressively harder and techniques that do automatic feature selection will be very useful in building effective models.

Finally, Figure 5 compares the error in prediction in the memory subsystem for different models and across various benchmarks. Similar to the CPU, all models save for SVM-rbf, do quite well and have comparable errors. Also, when compared to CPU power, the prediction error for different models are similar. This hints at the fact that all models use the same set of relevant features, which for the memory subsystem (which is simple) is quite predictable—L3-cache-misses being the most relevant and dominant feature.

Most of the results discussed so far were as expected—linear models have been shown to work well to predict full system power [14, 31]. Promisingly, even with an increase in subsystem power management complexity over prior work [5], the same linear models also do well in predicting platform subsystems.

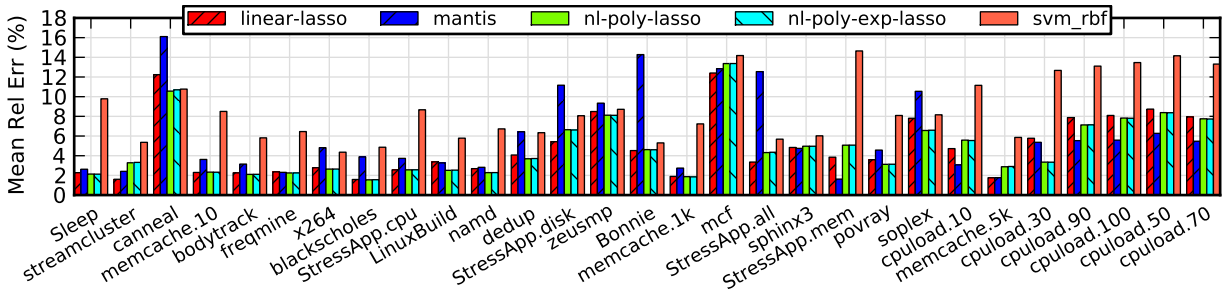

Figure 5: Modeling acuracy for memory power (Single-Core). Mean relative error is 4–8% across workloads.

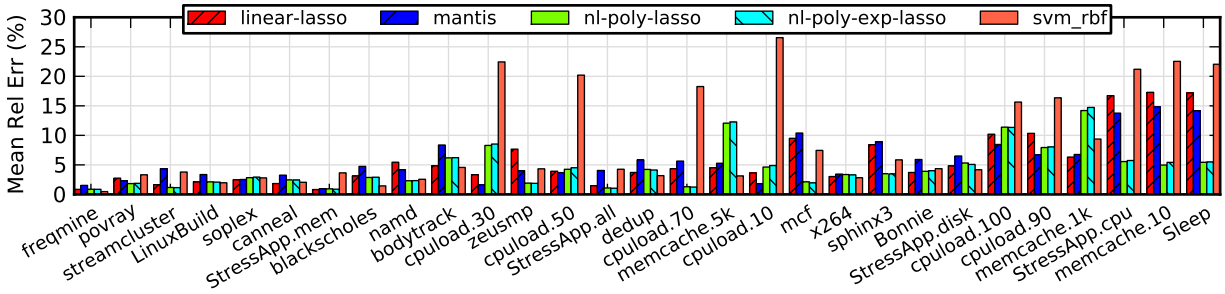

Figure 6: Modeling acuracy for total system power (Multi-core). Mean relative error is 2–6% across workloads.

6.2 Multicore

Next, we move to a more complex, yet more representative system configuration that utilizes all four available cores (HyperThreading and TurboBoost are still disabled and the P-state is still fixed). While more realistic, it still insulates the system from the extra hidden states of DVFS and functional contention, making power as easy to predict as possible. Figure 6 shows that across all the benchmarks, total system power is predicted to within 2-6%, which is respectable. This is well within the accuracy range required for tasks like data center server consolidation. However, as shown in Figure 7, the prediction error for CPU power is significantly higher, compared to that of the single core configuration, at 10-14%.

If we look more closely at the individual benchmarks for system power (Figure 6), we see that the error varies drastically by particular benchmark. Some benchmarks are predicted quite well (those near the left of the bar graph) and others do rather poorly (those to the right). Interestingly, the ordering of models changes with each benchmark, i.e., a particular model does not consistently do better than another over the entire set of benchmarks. In every benchmark there is at least one model that has an error less than 6%, but it is not always the same model. Our intuition for this behavior is that the model finds a roughly linear/polynomial/exponential space that fits some of the benchmarks, but then fails to capture the complex nature of contention on system resources to accurately model all workloads.

Figure 7 shows the prediction error across models for the CPU power. These results are even more striking. For workloads that lightly load the system (`Sleep`, low-rate `memcached`, etc.) and workloads that stress very specific components (`StressAppTest`, `cpuload`, etc.), the prediction is poor. This is particularly concerning because previous work shows that most production systems are run at low utilization levels [3]. Hence, one might hope that prediction is much better at idle or low load (which is a more realistic scenario in production systems). Unfortunately, the error climbs above 80% for most of the models. Thus, we find that all the models we evaluate are limited in their ability to predict the power consumption of workloads on multicore systems.

The metric of mean absolute error indicates instantaneous prediction error. An astute reader might observe that some prediction applications might be concerned with long term averages and that the instantaneous errors might balance out. While time averaging can reduce the overall impact, many of the benchmarks experience one-sided errors and the models systematically overpredict for some and underpredict for others. Furthermore, even though the percentage error is influenced by the actual power magnitude, we find that most benchmarks are systematically mispredicted by 1–6W on average.

We posit that one of the factors that contributes in a significant way to the poor prediction performance is the increasing presence of hidden power states. In the present case, there are several resources shared across

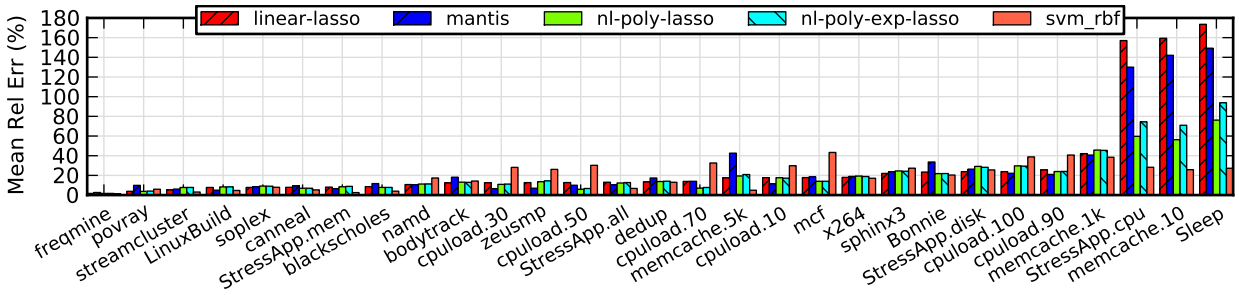

Figure 7: Modeling accuracy for CPU power (Multi-core). Mean relative error is 10–14% across workloads, but as high as 150% for some workloads on the right.

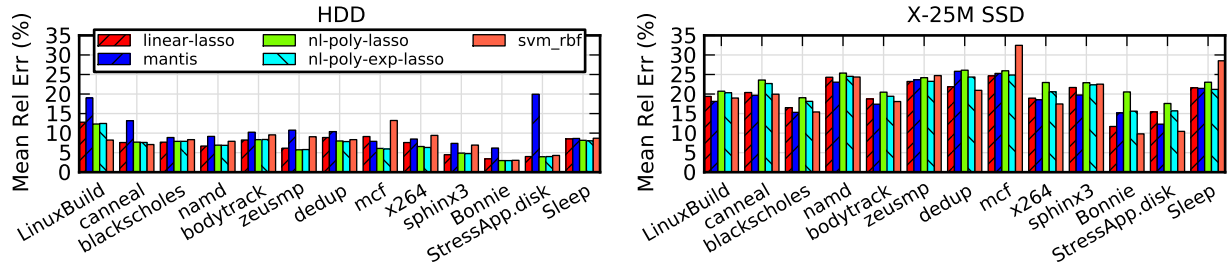

Figure 8: Prediction error with hard disk (left) and newer technology SSD (right).

the cores (L2 caches, for one), which lead to resource contention between the cores causing bottlenecks in processing. However, this very low level behavior is not captured in any of the exported features, and consequently does not make its way into the model. Since we cannot observe the unexposed CPU state to understand what is really happening, in the following section we use the increased internal complexity of SSDs vs. hard drives, instead, to demonstrate degradation in modeling due to hidden power states.

6.3 Hidden states

An important concern when modeling individual hardware components is whether the model, or the inputs to the model, capture all relevant aspects of the component's operation. This derives from the tension between the increasing complexity of the component, and hiding state to present a simple and consistent interface to the OS. If the component incorporates optimizations and circuitry that affect its power draw without varying any of the counters and states that it is exporting externally, then the model, which relies completely on the externally visible features, is likely to fail. As a case in point, there is anecdotal evidence to suggest that newer processors aggressively optimize for power by turning off functional blocks inside the CPU that are not being used, or doing clock gating at a finer granularity than what is exposed on the C-states (neither of which can be easily observed or inferred by software). Another example of this phenomenon can be seen in modern SSD drives: these include a number of performance and robustness optimizations (e.g. wear leveling, page re-writing, etc.). While these complexities are well known [8], they are not exposed via the SATA statistics, as evidenced by low mutual information with the power values. Thus, the power consumed by a given write may have more to do with the hidden state of the device than with the write itself.

To explore this systematically, we ran the same benchmarks on the same platform but with two different hard disks. The first was a conventional 2.5" WD Caviar Black 10K-RPM HDD, and the second was a newer Intel X-25M SSD. Power measurements on the drives show similar power ranges: 0.9–3.6W for the traditional platter based HDD, and 1.2–3.6W for the newer technology SSD. Note that for both disks the features that are recorded are identical, and attempt to capture the amount of work done by the OS in writing/reading from the disk.

Figure 8 shows the prediction errors for both the drives. The high level takeaway is that the error is consistently larger for the SSD than it is for the traditional drive. Specifically, we see that across the set of benchmarks, the model predictions are off by around 7% in the case of the conventional HDD, while they are off by approximately 15% for the SSD drive. Since the features collected and examined in each case are the same, the prediction errors are clearly caused by internal state

changes in the SSD that are reflected in the power draw, but not exposed in the features being tracked. This augurs poorly for power prediction models, given that hardware complexity continues to grow by leaps and bounds.

7 Discussion

While our results indicate that the modeling techniques we study suffer from significant prediction error, it is natural to ask 1) whether this error is in any way fundamental, or could be overcome with more sophisticated techniques, and 2) how useful the resulting predictions might be for particular dynamic power management (DPM) applications. We discuss both issues in this section. First, we present anecdotal evidence that the inherent variability between identical hardware components is likely to introduce a basic error term to any modeling based approach that cannot be solved by adding complexity into the model. Second, we discuss how useful currently achievable levels of accuracy can be to DPM systems.

7.1 Variability

Power modeling is based upon an underlying presumption that the error characteristics of the model do not change over time or over instances of the platform. That is, the model can be generated as a one-time operation by training on a specific platform instance, and the model can be used to predict the power consumption for any *other* instance of platform with the same specifications. While minor variations in manufacturing parts is a given, historically it has not significantly affected the operating characteristics of the platform and processors.

However, the increasing complexity of modern hardware, with staggering amounts of circuitry being stuffed into ever smaller packages exaggerates the variations significantly. These variations lead to variability in power consumption both in the active and standby modes. Furthermore, the variability is only exacerbated by aging and differences in operating environment. Recent work shows that in an 80-core general purpose chip designed by Intel, the frequency, which is directly co-related with the power consumption, of individual cores varied by as much as 25–50% for different operating voltages [13]. Research in embedded systems has shown that multiple instances of a Cortex M3 micro-controller can vary in sleep power consumption by as much as 5× [38]. Recent work has also demonstrated that the performance and power consumption of flash memory chips varies widely based on age, wear and write location [8].

This level of variability raises questions about the ability of power models to generalize over identical systems because they do not actually perform identically. Errors in the power model are amplified by variations across

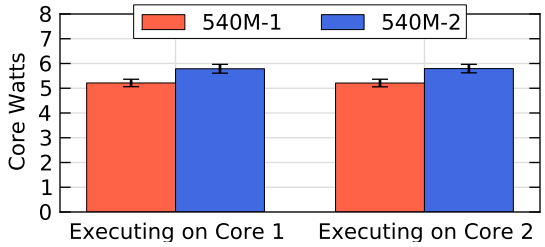

Figure 9: Variability in power consumption measured across two CPUs of the exact same type: Intel Core i5-540M. Core 1 shows a 11% variability between the two processors, while Core 2 shows 11.2% variability. Measurements are averaged across ten runs for each case, with standard deviation marked.

different instances. Using our measurement platform we see significant differences between two identical Core i5-540M processors. Figure 9 illustrates the measured power consumption of the two processors (540M-1 and 540M-2) running our `cpuload` benchmark pinned to either Core 1 or Core 2 using Linux `cpu_sets`. We report CPU power on a single platform so that everything—the mainboard, memory, benchmark, measurement infrastructure, and the operating environment(temperature)—but the processor is constant.

As can be observed from Figure 9, the CPU power consumption when executing on Core 1 for 540M-2 is 11% higher than when using 540M-1, and is similarly 11.2% higher when executing on Core 2. Note that the power consumption is averaged across ten runs on the individual cores (Core 1 or Core 2) for the two processors (540M-1 or 540M-2). We also report the standard deviation in Figure 9 which is measured to be less than 3.1% in all cases. Thus, if one of the models from Section 5 were trained on 540M-1 and applied to workloads executing on 540M-2, then a mean prediction error of 10% could translate into a 23% error and 20% prediction error translates into a 34% prediction error. Given the undeniable trend toward more complex—and therefore inherently variable—components such as processors, this fundamental accuracy gap seems likely to continue to grow. Hence, power instrumentation may be the only choice for accurate power characterization.

7.2 Implications for DPM

As discussed previously, the level of accuracy and the granularity (i.e., which subsystems are characterized) required for dynamic power management is strongly tied to the particular application domain. In some cases, reducing energy might be possible with only a coarse grained and approximate power consumption estimate. In other cases, the application is likely to need a higher degree of accuracy than modeling can currently provide.

A promising way to save energy on a computing platform is by scheduling computation more optimally. This could be done by migrating threads to different cores on the same socket or on different sockets (power gating is often done at the socket level, so using additional cores on the same socket has a very small cost). The processing cores available on a platform may be homogeneous (all derived from identical parts) or heterogeneous (from disparate parts and even architectures).

When there are a multiplicity of processing cores available, we expect the power cost to be quite different, and characterizing the power for each of these cores is critical when deciding to migrate computation. As seen in Section 5, the power models for subsystems like the CPU can have errors of up to 40%. When the errors dwarf the actual power variations across the cores (and this is a likely scenario when the cores are not architecturally different) it is likely that the mispredictions have an adverse effect. However, when the choice is between heterogeneous components such as between a CPU or a GPU, with significantly different power characteristics, i.e., where the variation might be larger than the model errors, it might still be acceptable to rely on modeling.

In another domain, prior work on mobile devices has shown that dynamically switching between multiple heterogeneous radios, such as WiFi and Bluetooth, can in some cases double battery lifetime [27]. Choosing between different radio alternatives like these with vastly different power characteristics seems straightforward even with very poor accuracy. However, recent work has shown that modern WiFi radios, such as those based on the 802.11n MIMO standard, have many more complex states, each with different power consumption tradeoffs [20]. Accurate component-level power characterization will therefore be essential to make optimal decisions on which radio interface or computational unit—and in which mode—to use.

Finally, we note that power-aware resource scheduling is not limited to resources within the same platform. In fact the advent of abundant cloud computing resources has accelerated research into systems, such as MAUI [12], that can use both local computation (on mobile devices) and also execute code remotely in the cloud whenever needed. Currently these systems operate under the assumption that servers in the cloud are always-powered and, hence, their energy costs are not as important as those of battery-powered mobile devices. These systems would benefit significantly from detailed power characterization on the local mobile device as well as the servers in the cloud. Using this information, the policy decisions on when to execute code locally or remotely can be more informed and therefore more optimal. The absolute amounts of energy being considered (i.e., the execution of a single function call) in code offload scenarios, however, are fairly small, so high degrees of accuracy seem essential.

8 Conclusion

The models we consider are able to predict total system power reasonably well for both single core (1–3% mean relative error) and multi-core scenarios (2–6% mean relative error), particularly when the base power of the system is high. However for predicting subsystem power, we show that linear regression based models often perform poorly (10–14% mean relative error, 150% worse case error for the CPU) and more complex non-linear models and SVMs do only marginally better. The poor subsystem power modeling is due to increased system and device complexity and hidden power states that are not exposed to the OS. Furthermore, our measurements show surprisingly high variability in processor power consumption, for the same configuration across multiple identical dies, highlighting the fundamental challenges with subsystem power modeling. Looking forward, while modeling techniques may suffice for some DPM applications, our results motivate the need for pervasive, low-cost ways of measuring instantaneous subsystem power in commodity hardware.

Acknowledgments

We wish to thank Bharathan Balaji for his help with the power measurement setup and Lawrence Saul for his advice regarding the applicability of various machine learning techniques. We also thank our shepherd, Kai Shen, and the anonymous reviewers for their comments and feedback that improved our paper. This work is supported by a grant from Intel and NSF grants CCF-1029783 and CCF/SHF-1018632.

References

[1] Y. Agarwal, S. Savage, and R. Gupta. SleepServer: A Software-Only Approach for Reducing the Energy Consumption of PCs within Enterprise Environments. In *USENIX ATC '10*, 2010.

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *SOSP*, 2009.

[3] L. A. Barroso and U. Hölze. The Case for Energy-Proportional Computing. *IEEE Computer*, 2007.

[4] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[5] W. L. Bircher and L. K. John. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. *2007 IEEE International Symposium on Performance Analysis of Systems and Software*, 2007.

[6] W. L. Bircher and L. K. John. Predictive Power Management for Multi-Core Processors. In *WEED*, 2010.

[7] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *USENIX ATC*, 2010.

[8] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: Using Flash Memory to Build Fast, Power-Efficient Clusters for Data-intensive Applications. *ACM SIGPLAN Notices*, 44(3):217–228, 2009.

[9] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] H. Chen, S. Wang, and W. Shi. Where Does the Power Go in a Computer System: Experimental Analysis and Implications, 2010. Technical Report MIST-TR-2010-004.

[11] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscienc, NY, USA, 1991.

[12] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making Smartphones Last Longer with Code Offload. In *MobiSys '10*. ACM, 2010.

[13] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, et al. Within-die Variation-Aware Dynamic-Voltage-Frequency Scaling Core Mapping and Thread Hopping for an 80-core Processor. In *Proceedings of ISSCC*. IEEE, 2010.

[14] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-System Power Analysis and Modeling for Server Environments. *MOBS*, 2006.

[15] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic Performance Setting for Dynamic Voltage Scaling. In *Proceedings of MobiCom '01*, 2001.

[16] J. Flinn and M. Satyanarayanan. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Trans. Comput. Syst.*, 22(2):137–179, 2004.

[17] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking Energy in Networked Embedded Systems. In *OSDI*, 2008.

[18] Glmnet for matlab. `http://www-stat.stanford.edu/~tibs/glmnet-matlab/`.

[19] R. K. Gupta, S. Irani, and S. K. Shukla. Formal methods for Dynamic Power Management. In *Proceedings of ICCAD '03*, 2003.

[20] D. Halperin, B. Greenstein, A. Sheth, and D. Wetherall. Demystifying 802.11n Power Consumption. In *HotPower*, 2010.

[21] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. Bhattacharya. Virtual Machine Power Metering and Provisioning. In *Proceedings of the 1st ACM Symposium on Cloud computing (SOCC '10)*. ACM, 2010.

[22] R. Koller, A. Verma, and A. Neogi. WattApp: An Application Aware Power Meter for Shared Data Centers. In *ICAC*, 2010.

[23] K. Li, R. Kumf, P. Horton, and T. Anderson. A Quantitative Analysis of Disk Drive Power Management in Portable Computers. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 22–22, 1994.

[24] T. Li and L. K. John. Run-time Modeling and Estimation of Operating System Power Consumption. In *SIGMETRICS*, San Diego, June 2003.

[25] Y. Lu, E. Chung, T. Simunic, G. De Micheli, and L. Benini. Quantitative Comparison of Power Management Algorithms. In *Proceedings of DATE '00*, 2000.

[26] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *SOSP*. ACM, 2007.

[27] T. Pering, Y. Agarwal, R. Gupta, and R. Want. CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. In *MobiSys*, 2006.

[28] T. Pering, T. Burd, and R. Brodersen. Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System. In *Power Driven Microarchitecture Workshop, attached to ISCA98*, 1998.

[29] Q. Qiu, Q. Qu, and M. Pedram. Stochastic Modeling of a Power-Managed System-construction and Optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2002.

[30] V. J. Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency. In *ISCA*, 2010.

[31] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A Comparison of High-Level Full-System Power Models. In *HotPower*, San Diego, December 2008.

[32] N. J. Sanders. Stressapptest. `http://code.google.com/p/stressapptest/`.

[33] K. Shen, M. Zhong, S. Dwarkadas, and C. Li. Hardware counter driven on-the-fly request signatures. *ASPLOS*, 2008.

[34] SpecCPU2006. `http://www.spec.org/cpu2006`.

[35] T. Stathopoulos, D. McIntire, and W. J. Kaiser. The Energy Endoscope: Real-time Detailed Energy Accounting for Wireless Sensor Nodes. In *IPSN*, 2007.

[36] N. Tolia, Z. Wang, M. Marwah, C. Bash, and P. Ranganathan. Delivering Energy Proportionality with Non Energy-Proportional Systems–Optimizing the Ensemble. In *HotPower*, San Diego, December 2008.

[37] X. Wang and Y. Wang. Coordinating Power Control and Performance Management for Virtualized Server Clusters. *TPDS*, 2010.

[38] L. F. Wanner, R. Balani, S. Zahedi, C. Apte, P. Gupta, and M. B. Srivastava. Variability-Aware Duty Cycle Scheduling in Long Running Embedded Sensing Systems . In *Proceedings of Design, Automation and Test in Europe 2011 (DATE)*, 2011.

[39] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *ASPLOS*, 2002.