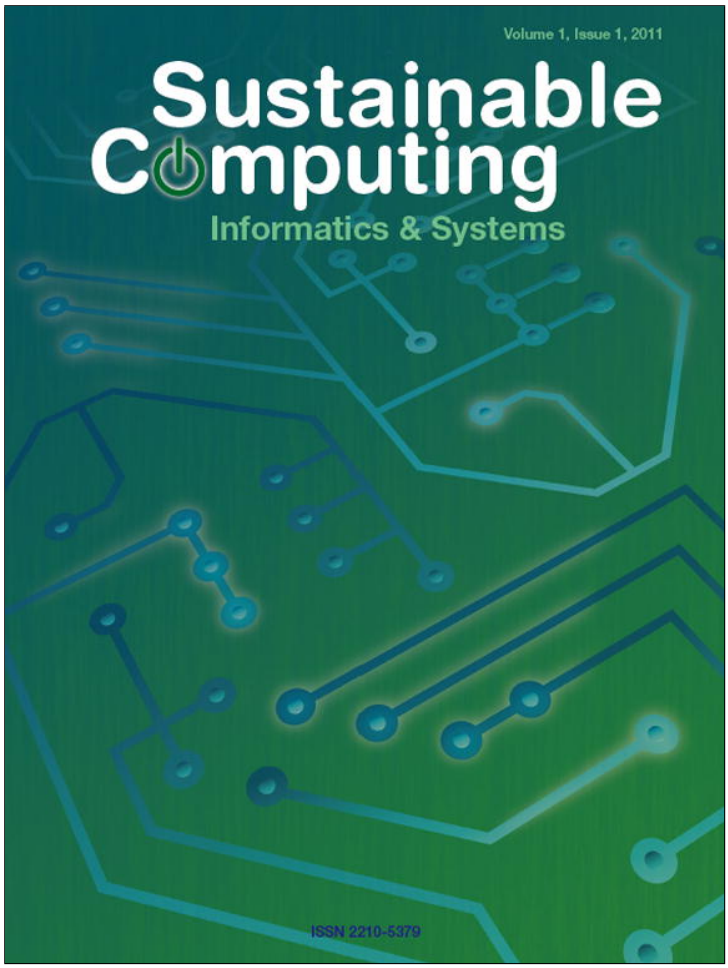

Sustainable Computing: Informatics and Systems 1 (2011) 23–34

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

SPAN: A software power analyzer for multicore computer systems

Shinan Wang, Hui Chen, Weisong Shi*

Department of Computer Science, Wayne State University, Detroit, MI, USA

ARTICLE INFO

Keywords:
Power model
Software power profiling
Multicore

ABSTRACT

Understanding the power dissipation behavior of an application/workload is the key to writing power-efficient software and designing energy-efficient computer systems. Power modeling based on performance monitoring counters (PMCs) is an effective approach to analyze and quantify power dissipation behaviors on a real computer system. One of the potential benefits is that software developers are able to optimize the power behavior of an application by adjusting its source code implementations. However, it is challenging to relate power dissipation to the execution of specific segments of source code directly. In addition, existing power models need to be further investigated by reconsidering multicore architecture processors with on-chip shared resources. Therefore, we need to adjust PMC-based power models from the developers' perspective, and reevaluate them on multicore computer systems.

In this paper, followed by a detailed classification of previous efforts on power profiling, we propose a two-level power model that estimates per-core power dissipation on chip multiprocessor (CMP) on-the-fly by using only one PMC and frequency information from CPUs. The model attempts to satisfy the basic requirements from developer point of view: simplicity and applicability. Based on this model, we design and implement SPAN, a software power analyzer, to identify power behavior associated with source code. Given an application, SPAN is able to determine its power dissipation rate at the function-block level. We evaluate both the power model and SPAN on two general purpose multicore computer systems. The experimental results based on `SPEC2008Cjvm` benchmark suite show the average error rate of 5.40% across one core to six core validation. We also verify SPAN using the FT benchmark from NAS parallel benchmark suite and a synthetic workload. The overall estimated error of SPAN is under 3.00%.

1. Introduction

Many previous efforts have focused on computer system power measurements and profiling [7,9,14,18,28,33,34]. Actually, power dissipation of a single computer system can be broken down into several pieces with each piece representing a component, such as CPU or memory. Furthermore, power dissipation of each component consists of two parts: static power and dynamic power. The former could be described as the basic power supplied to maintain this component in its operational state. The latter is the additional power dissipation for running a specific task. For years, it has been well-acknowledged that dynamic power is roughly determined by utilization rates, especially for CPUs. However, the experimental results show that, for the CPU dynamic power, the estimation error rate of using this method can be as high as 33.33% [12]. On the other hand, understanding the power dissipation behavior of a specific software/application is the key to writing power-efficient software and design energy-efficient computer systems. Therefore, we need a more accurate model to capture the power dissipation of computer systems.

Usually, there are four ways to estimate power dissipation: *cycle-level system simulators*, *instruction-level modeling*, *software-function-level macro-modeling*, and *PMCs-based modeling*. Cycle-level system simulators are time costly while providing more detailed information [10,56]. Instruction level modeling achieves simplicity and accuracy on embedded systems, but it is not realistic if we apply it to superscalar processors with a large number of instructions. For example, IA-32 ISA contains 331 different instructions, with 109,561 (331^2) instruction combinations if we consider the inter-instruction effects. Software-function-level macro-modeling techniques associate power dissipation with application function sub-routines and establish power models on top of application characteristics, such as algorithm complexity [49]. However, such information sometimes is inherently unavailable for end users. Moreover, the static feature of this method prevents its utility when we consider advanced run-time power management. Analytical power modeling based on performance monitoring counters (PMCs) enables run-time software power estimation [14,28,33,43]. Nevertheless, for a given processor, the power model based on PMCs is limited by the types of available event counters and the maximum number of counters that can

* Corresponding author. Tel.: +1 3135773186.
E-mail addresses: shinan@wayne.edu (S. Wang), huichen@wayne.edu (H. Chen), weisong@wayne.edu (W. Shi).

2210-5379/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.suscom.2010.10.002

be read simultaneously. For instance, most Intel processors only support sampling two counters per core concurrently. Regarding the power estimation utilizing PMCs, however, we also need to notice that accuracy highly depends on two sets of PMCs: those PMCs appearing in power models and those PMCs available on targeting platforms. Insufficient information representing the power characteristics of the microarchitecture will yield low accuracy.

Software contributes considerably to the total power of a computer system [6,12,45]. Hence, it is very important to find out how much power has been dissipated by a specific software component in order to design sustainable computer systems. Power dissipation, arguably speaking, is a fundamental aspect of software nowadays. On one hand, the total energy consumption of completing a task is power accumulation over time. Thus, power dissipation is a direct contributor to producing an energy profile. On the other hand, in some particular circumstances, controlling power dissipation provides more flexibility for systems. For example, temperature can be altered by restricting power dissipation. Besides, some infrastructures add "power envelop" as one of the constraints. For instance, it is crucial for data centers serving millions of people to maintain the whole power budget under a certain limit for power supply protection (huge current draw may damage transistors). As a result, it is worth to investigating the run time power dissipation of an application and the associated source code for sustainable computing point of view.

In this paper, we focus our discussion on identifying run-time factors that determine the power dissipation of processors for computation intensive workloads on power-aware multicore computer systems. Concretely, we model power dissipation in a two-level manner to reserve simplicity and accuracy. More importantly, we map power dissipation to software blocks at runtime by building SPAN libraries and interfaces. Specifically, the work presented in this paper includes the following contributions.

- First, we examine previous power measurement and profiling techniques and survey possible solutions for power estimation on modern multicore systems in a comprehensive way by classifying them into three main categories: *hardware-based*, *software-based*, and *hybrid method*.
- Second, we propose a two-level power model for power-aware multicore computer systems. The novelty of the proposed model is two-fold. First, we minimize the number of performance counters and training benchmarks utilized in the model to achieve simplicity and applicability. Second, we incorporate frequency in the power model to meet the requirements of modern DVFS techniques.
- Third, we design and implement SPAN to relate power dissipation to the different portions of an application source code using the proposed power model. By using SPAN, developers can easily identify the sections of code consuming the most power in the program.
- Finally, we perform comprehensive experiments on two recent multicore platforms, Asus Essentio CM5570 and HP Pavilion Elite HPE-000, under various DVFS configurations to evaluate and validate both the power model and SPAN via real hardware measurements.

The rest of the paper is organized as follows. We start the paper by presenting a detailed survey of the previous efforts on power measurement and profiling in Section 2. In Section 3, the details of the proposed model are described, followed by the SPAN design and implementation in Section 4. We illustrate the results of the proposed power model and SPAN on two recent multicore computer systems in Section 5. Moreover, to distinguish our work from the previous research, we discuss the most relevant techniques in Section 6. At last, we summarize our work and reach the conclusion in Section 7.

Table 1
Classification of power measurements and profiling.

Hardware-based	Software-based	Hybrid
Itsy00 [52]	Wattch [10]	PowerScope [17]
Jpseph01 [29]	SimplePower [56]	Isci03 [25]
Kamil08 [31]	SoftWatt [21]	PowerPack [18]
PowerExecutive [1]	Orion [53]	Chang03 [11]
IMPI [24]	SimWattch [13]	Lorch97 [36]
	Dempsey [57]	
	Bellosa00 [7]	
	vEC [30]	
	Powell:2009 [42]	
	Bertran10 [8]	

2. Background: power measurements and profiling

As energy consumption becomes one of the foremost considerations in designing new computer systems, power-aware system design raises a key issue in the community of computer systems. Power measurement and profiling, which are the basis of power-aware systems, not only can be used to evaluate power optimization techniques and to make power-performance tradeoff, but also can be used to generate critical power information for operating systems and power-aware software. Based on hardware and software techniques used, power measurement and profiling could be classified into three categories: *hardware-based method*, *software-based method*, and *hybrid method*. Table 1 summarizes the classification of these previous efforts.

2.1. Hardware-based method

The hardware-based methods mainly use two strategies: using meters to build a power measurement and profiling platform or integrating power sensors into hardware architectures.

2.1.1. Direct power measurement and profiling

The first strategy uses meters to measure the currents or voltages of wires that supply power for hardware; then compute power dissipation with these results. This strategy is usually used to evaluate the accuracy of software methods. In this paper, we also use this classic power measurement method to evaluate the accuracy of our models. One of the earliest studies of power measurement and profiling is done by Viredaz et al. on handheld computing devices [52]. Joseph et al. use a similar method to measure the power dissipation on a high performance processor [28,29]. They use a group of microbenchmarks with particular cache, bit activity, and branch prediction behaviors to evaluate performance and power trade-off. In [31], the authors adopt the direct power measurement method; then, they measure the power on a Cray XT4 supercomputer under several HPC workloads. Their results show that computation-intensive benchmarks generate the highest power dissipation. Nevertheless, memory-intensive benchmarks yield the lowest power usage. Physical measurement is fast and objective, but this method lacks a semantic connection between measurement results and evaluated programs [23].

2.1.2. Integrate power sensors into the system

The second strategy is usually used by high-performance servers [1,19]. For example, Intel uses service processor-based power-monitoring sensors to provide power information for systems through the API called Intelligent Platform Management Interface (IPMI) [19,24]. IBM BladeCenter and System x_TM servers supply PowerExecutive solutions that enable customers to monitor actual power draw and temperature loading information [1]. Though online power-aware applications can use this method, it is difficult to yield low-level power information because hardware circuits are

too complicated to distinguish the originality of power dissipation. In addition, power monitoring circuits also dissipate a large amount of power as well.

2.2. Software-based method

Even though hardware-based methods are more accurate than software-based methods, hardware cost and scalability requirements restrict their application range. In addition, during an architecture design cycle we cannot use hardware-based methods to balance power and performance. Software-based methods use power models to estimate power dissipation. Power models are created at different levels: circuits level, instruction level, component level, node level, and so forth. Based on different usage stages, we summarize software-based methods into two types: architecture-level power models, which are used to estimate power dissipation during the architecture design stage, and system-level power models, which supply live power information to operating systems and power-aware applications.

2.2.1. Architecture-level power model

Software-based methods spring up in the area of architecture-level power estimation. Most of the earliest work [10,35,39,41,56] in this category are based on the classic energy equation [27]. Liu and Svensson estimate the power on VLSI CMOS chips [35]. Register transfer level power model is analyzed by Marculescu et al. in [39]. Brooks et al. proposed Wattch, a framework for analyzing and optimizing microprocessor power dissipation at the architecture-level [10]. The power model of Wattch relies on per-cycle resource usage counts. In [56], Ye et al. present a comprehensive framework called SimplePower, which is based on the transition sensitive energy models. It not only can be used to evaluate the effect of high-level algorithmic, architectural, and compilation trade-off on energy, but also provides the energy consumption in the memory system and the on-chip buses using the analytical energy models. SoftWatt, which models the CPU, memory hierarchy, and the low-power disk subsystem, is described in [21]. This tool is able to identify the power hot-spots in system components as well as the power-hungry operating system services.

The power constraints in interconnection network design were noticed by [41]. Also, the authors propose the power model of routers and links and analyzes the performance of direct interconnection network topologies under a fixed power constraint. Wang et al. present a power-performance interconnection network simulator called Orion, which is capable of providing detailed power and performance characteristics to enable rapid power-performance trade-off at the architecture-level [53]. Eisley et al. estimate and analyze the power of CMPs by synergistically considering both the processor cores and the communication fabric in a multi-core chip [16]. Chen et al. propose SimWattch to integrate the system-level and the user-level simulators [13].

Besides those efforts that model the power dissipation of processors, several publications [10,22,40,55,57] propose methods to estimate the power of other devices, such as hard disks, memories, and network devices. Zedlewski et al. present Dempsy, a disk simulation environment that includes the accurate modeling of the disk power dissipation [57]. Dempsey attempts to estimate the power of a specific disk stage, which includes seeking, rotation, reading, writing, and idle-periods, with a fine-grained model. Molaro et al. also analyze the possibility to create a disk driver power model based on disk status stages [40]. In [22], the authors build the power model for hard disk based on the observation that a slight change on the rotation speed of a disk has a quadratic effect on its power dissipation. In [55], Ye et al. introduce a framework to estimate the power dissipation on the switch fabrics on network routers and propose different modeling methodologies for the node switches, internal buffers, and interconnect wires inside switch fabric architectures.

2.2.2. System-level power model

Specialized circuit techniques are important strategies for low-power designs, but these techniques alone are not sufficient. Higher-level strategies for reducing power dissipation and improving energy efficiency are increasingly crucial [10]. The live power information of systems is highly needed for designing high-level energy efficiency strategies. For example, Ecosystem [58] and [38], which propose the concept managing system energy as a type of resource, require the support from real-time power information on different levels. Furthermore, in [51], the authors argue that the traditional operating system design should be revisited for energy efficient usage. As part of the energy-centric operating system, energy profiles are also needed by new power-aware scheduling algorithms [5,32]. Ahmad et al. propose a new power-aware scheduling algorithm based on game theory [5]. In [32], Khan and Ahmad present a method, which is also based on game theory, to minimize the energy consumption on computational grids. System-level power models are built on the statistics of systems, which reflects activities of the hardware devices.

One of the earliest research in this category is [50]. Tiwari et al. propose an instruction-level power model for embedded processors and memories. Russell et al. present an energy model using a constant parameter for power dissipation of a 32-bit embedded processor [44]. Li and John [34] exploit a high correlation between the instruction per cycle (IPC) and the power dissipation, and they predict the run-time power dissipation on the OS routines based on regression model between power and IPC. Contreras and Martonosi [14] also discover the power-IPC correlation and use five PMCs to estimate the power of workloads running on different CPU frequencies. Their model exhibits low percentage of error, but they do not verify the model on multicore architectures. Bircher et al. [9] explore the run-time events that most likely represent power dissipation. In their experiments, IPC-related metrics are shown to be the most power-informative. Among those metrics, the upos fetched per cycle yields the most accurate results. Other candidates are upos completed per cycle and upos retried per cycle. Wu et al. [54] also use a number of PMCs to deploy a power model on the Pentium 4 functional units. They measure the CPU power via a clamp-on ammeter. However, their model is not validated under different frequencies and multicore architecture.

Dhiman et al. [15] propose an on-line power prediction system on virtualized environments. Instead of using linear regression models on PMCs, they utilize a Gaussian Mixture vector quantization based training and classifying. The estimation error is within 10% in most cases. Bertran et al. [8] demonstrate an alternative approach using PMCs on the CPU power estimation. Rather than directly deriving power models using PMCs, they propose a method to treat each component of the CPU separately, such as FE, INT, and FP. Combining all the training parameters, they develop a fine-grained power model. However, the training process is time-consuming in practical situation. In addition, the power model highly depends on the microarchitecture of the CPU. Bellosa [7] demonstrates the correlation between the recorded performance events and the power dissipation from the synthetic workloads. He shows the most effective factors of system power dissipation are: fuops/s, uops/s, L2 accesses/s, and memory accesses/s. Because he only considers the synthetic workloads, the results could not be sufficiently applied to real applications.

In [42], Powell et al. propose a methodology to reduce the number of performance counters while maintaining certain accuracy of the model. They estimate the hardware activity events of several microarchitectural structures; then, the authors associate the activity events to the power dissipation of such structures. Singh

and Bhadauria [47] describe an approach based on a number of microbenchmarks which stress the particular components of a given processor architecture. Our work differs from all these works in the way that we combine CPU frequency scaling and multicore features in the power model, which fits the trend of recent microprocessor design.

2.3. Hybrid method

Hybrid methods are also globally researched [17,18,25] because both hardware-based and software-based methods have their own limitations. Flinn et al. develop a platform that samples both the power dissipation and the system activities on a profiling computer; then, they generate an energy profile from the data through an off-line analysis [17]. Isci and Martonosi build a platform using sampled multimeter data for overall power measurements and produce per-unit power breakdowns based on the hardware performance counter readings [25]. Their power model uses 22 performance monitoring counters and reaches as low as 5% error rate on the SPEC 2000 benchmarks. However, the large number of performance counters may not be available for sampling simultaneously on some processors. For example, most Intel platforms only support concurrent sampling of two counters. In this case, to retrieve the information from 22 counters, the program has to be run at least 11 times. Ge et al. develop a power measurement and profiling platform to retrieve the power information from the main components, such as the CPU, disk, memory, motherboard, and so forth [18]. Also, they propose a method to map the measured power into the application code and analyze the energy efficiency in a multi-core system. Isci and Martonosi develop an experimental framework to compare the control-flow based with the performance-monitoring-based power-phase detection techniques [26]. Their results show that both the control-flow and the performance statistics provide useful hints of the power phase behaviors.

Chang et al. rely on statistical sampling to help programmers evaluate the energy impact of their design decisions [11]. In [23], they describe an evaluation infrastructure, which combines the advantages of simulations and physical measurements for the OS/compiler power and energy optimizations. In addition, this infrastructure can provide the objective evaluation and semantic connection between the measured power/energy and the source code. Lorch et al. design two programs: PowerMeasure, which is used to measure how much power each component consumes in the predefined state, and StateProfiler, which is used to profile how often each component stays in a specific power state [36,37].

3. Two-level power modeling

The power dissipation of a given platform can be divided into two parts:

- *Baseline power*: the static power dissipation to maintain a system running. To be specific, static power of a motherboard, CPU, memory, CPU fans, and other components contributes to this part of the power dissipation.
- *Dynamic power*: the power dissipation due to a task execution. By executing workloads on different platforms and different frequencies, dynamic power varies considerably. Other contributing factors could be temperatures, characteristics of workloads, and component utilizations.

The first primary goal of this paper is to find a practical power estimation model describing dynamic power on multicore power-aware processors by using as few PMCs as possible. The essence of utilizing PMCs to estimate power dissipation is about information trade-off. The more PMCs information is retrieved, the more detailed and accurate the power model could be. However, collecting PMCs sometimes can be troublesome. First, commonly used processors cannot support retrieving more than a certain number of counters simultaneously. Previous models proposed [14,28] necessitate multiplexing the counters so that several of them can be accessed for one benchmark. Besides, the types and names of the monitored events vary from platforms to platforms [20]. Usually, the power model established on one platform is not necessarily extensible. For example, Goel et al. differentiate PMCs in their power model for four platforms. Additionally, sampling PMCs usually means system overhead, which can be overwhelmed when the number of PMCs becomes large.

On the other hand, in order to describe the power characteristics of a given platform throughout, a fine-grained power model usually is trained by a large set of benchmarks. For example, Bertran et al. [8] develop approximate 97 benchmarks to exercise the power components on a single CPU. As a result, the training process could be unexpected long.

The production of this section is a set of power models with three basic features. First, the models have to provide acceptable high accuracy. Second, the parameters of the power models can be retrieved through a simple training procedure, which can be applied practically. In addition, the model input, the total number of PMCs, has to be maximally reduced to avoid multiplexing counters.

3.1. Observations

Leveraging PMCs, the most obvious method is to discover the possible correlation between a specific PMC and the power dissipation. The training benchmarks fulfill the task of PMCs selecting according to correlation coefficients. After obtaining training data, usually, researchers develop a linear regression model to derive a power model. Previous approaches concentrate on the mathematical methods to eliminate outliers, and to achieve high accuracy. However, few of them focus on direct factors influencing power dissipation, such as frequency.

One example is the argument on IPC. Indeed, an IPC value does reflect the power dissipation with high correlation coefficients for various workloads. However, the relationship between them can be weak under certain circumstances. We generate IPC ranging from 1 to 0.01 by continuously executing a single X86 instruction as Fig. 1(a). The results of the corresponding CPU power dissipation are shown in Fig. 1(b). The overall correlation coefficient is 0.41 in this case. Nevertheless, the standard deviation of power dissipation is only about 0.067 W for the given range of IPC between 1 and 0.01. Rarely could IPC make a representative factor of power dissipation in the similar scenarios.

However, other than those extreme benchmarks, regarding real applications, values of some PMCs reflect the power dissipation well. As Fig. 2 illustrates, the IPC and the power dissipation present the correlation coefficient as high as 0.98 for the NAS parallel benchmark suite.

Given the above results, we observe that the same PMC possibly has changeable effects on the power dissipation prediction. One possible solution could be to profile different PMCs for each task and then select the most related ones, which is probably impractical. In order to minimize the uncertain effects from PMCs on power estimation, we attempt to find an overall frame restricting the power estimation range. Inspired by [48] and based on our observation, the operating frequency fits the position well. Fig. 3 shows the power behavior of four NAS parallel benchmarks executing under the frequency of 2.34 GHz and 2.00 GHz. Clearly, we are able to find the boundaries separating power dissipation, regardless of the types of the benchmarks, according to its operating frequency; thus, we

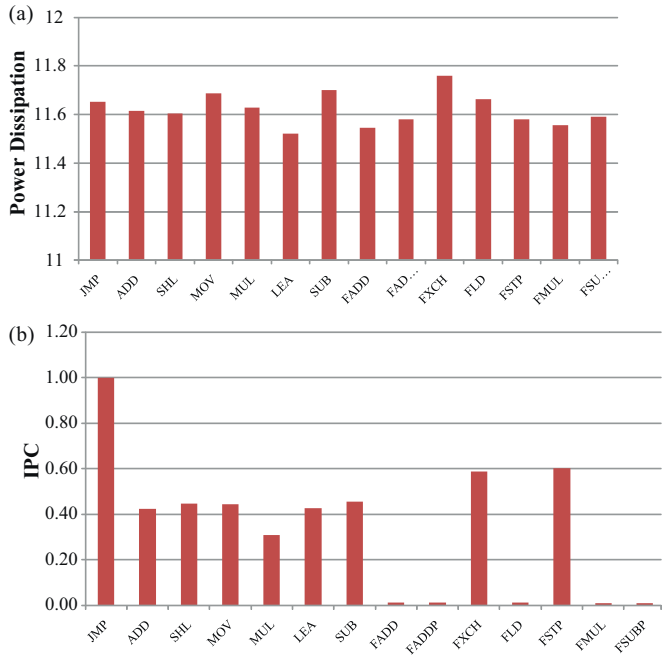

Fig. 1. Different instructions with their (a) IPCs and (b) power dissipation.

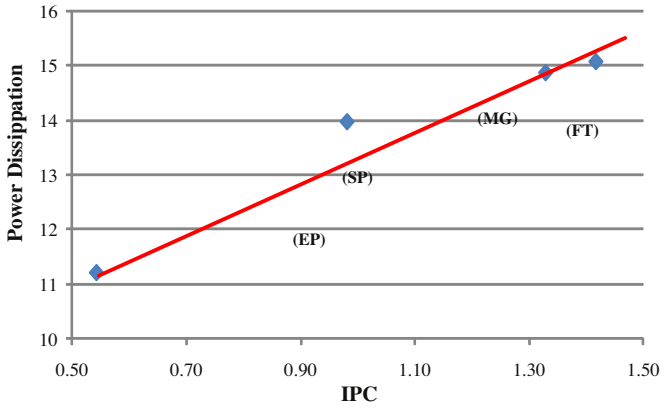

Fig. 2. IPC profile and the power dissipation of NAS parallel benchmarks (the correlation coefficient is as high as 0.98).

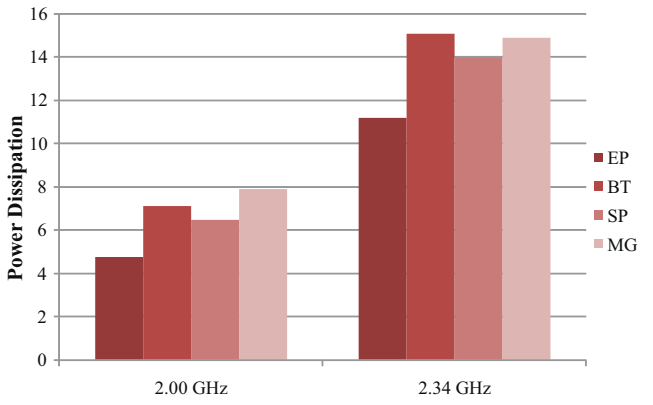

Fig. 3. Power dissipation of NAS parallel benchmarks operating under two frequencies.

establish a power model using frequency as the first level intuitively.

Generally speaking, we expect the power model to fully explore the possible relations between PMCs and power dissipation. Besides, if PMCs fail to provide positive information, the model will be able to minimize the disturbance introduced by it.

3.2. Methodology

In this section, we describe the methodology that produces the power models. In short, our approach follows the common modeling steps: defining the model input, generating microbenchmarks, training the power model, and applying the power model.

Considering inputs, the essential strategy is trade-off. On one hand, high accuracy necessitates more information from PMCs collected as inputs. On the other hand, the less PMCs we use, the more flexible and applicable the power model could be. We adopt only one PMCs to preserve the simplicity and to demonstrate the effectiveness of our two-level modeling. The PMC utilized in our study is IPC, as aforementioned. We design microbenchmarks carefully after selecting the model inputs. Totally, we test over 30 microbenchmarks stressing the CPU. By carefully reviewing, adjusting, and filtering them, we decide to choose 12 benchmarks for the training purpose because sufficient information can be provided from executing them. The training process is highly related to frequencies and IPC; however, we do not use linear directly, which most of the others do.

It should be noted that the main differences of our methodology from previous work are three-fold: we incorporate frequency information in the power model, we use minimum size of PMCs, and the methodology can be applied to other platforms easily.

Basically, we deploy IPC along with frequency as the model inputs. Actually, a strong relationship between Instruction Per Cycle (IPC) and power dissipation is established in previous work [14,34]. Although, in most cases, an IPC value is able to reflect the overall power dissipation, there are two issues by using IPC solely. First, different micro-operations might have various IPC values but similar power dissipation. For example, usually Floating Point Unit executes instructions much slower than Integer Arithmetic Unit nevertheless the power behaviors of them are similar. This problem can be easily solved if we consider each CPU component, such as FP, INT, and BPU (Brunch Prediction Unit) separately. In our case, it is not an option because we target on minimizing the PMCs in the model. Second, as aforementioned, because power behaviors of a CPU are mainly limited by its operating frequencies, by using IPC, there is some marginal effect. As the IPC becomes large or small enough, the effects of IPC on power dissipation drop noticeably.

Our solution for the first issue is using IPC as a second level power indicator that tunes the estimation results obtained according to operating frequencies. In order to eliminate the marginal effect, we divide benchmarks into different categories based on the IPC values; then, we collect data and derive the model separately for each category. We demonstrate our approach as follows in detail.

3.2.1. Power modeling

We denote the CPU frequency as F. Assuming that a CPU supports various frequencies, f_i, $i = 1, 2, 3, \ldots, n$, we attempt to obtain the power dissipation information, $P(f_i)$, for each frequency f_i. Given a set of training benchmarks T with its sub benchmarks t_j, $j = 1, 2, 3, \ldots, m$, executing under frequency f_i, we denote the power dissipation as $P(t_j, f_i)$ respectively. We calculate $P(f_i)$ as the median of $\{P(t_1, f_i), P(t_2, f_i), \ldots, P(t_m, f_i)\}$; thus $P(f_i)$ is resistant to outliers statistically. Besides, we represent IPC of each benchmark as $IPC(t_j, f_i)$. Similarly, the median IPC value of all the training benchmarks is defined as $IPC(f_i)$. In most cases, the benchmarks with the median value of $P(t_j, f_i)$ also contribute the median

value of $IPC(t_j, f_i)$. We describe $P(f_i)$ and $IPC(f_i)$ as *power pilot* for frequency f_i.

Second step, based on the *power pilot*, we compute $\Delta P(t_j, f_i)$ as the difference between $P(f_i)$ and $P(t_j, f_i)$ for each training benchmark. Similarly, we calculate $\Delta IPC(t_j, f_i)$ as the IPC difference of training benchmark t_i to the median value.

$$\Delta P(t_j, f_i) = P(t_j, f_i) - P(f_i) \tag{1}$$

$$\Delta IPC(t_j, f_i) = IPC(t_j, f_i) - IPC(f_i) \tag{2}$$

Targeting on predicting $\Delta P(t_j, f_i)$, we use $\Delta IPC(t_j, f_i)$ as model input to derive linear regression parameters, $P_{inct}(f_i)$ and $P_{\Delta}(f_i)$ as Eq. (3) shows. The final predicted power dissipation is shown in Eq. (4). We simply need to change $\Delta IPC(t_i, f_i)$ to be the actual $\Delta IPC(a_i, f_i)$ before applying the model to the i th benchmark from task set $a_1, a_2, a_3, \ldots, a_n$.

$$\Delta P(t_j, f_i)_{pret} = P_{inct}(f_i) + P_{\Delta}(f_i) \times \Delta IPC(t_j, f_i) \tag{3}$$

$$P(t_j, f_i)_{pret} = \Delta P(t_j, f_i)_{pret} + P(f_i) \tag{4}$$

It is easy to notice that the most majority of power dissipation is determined by $P(f_i)$, which stems from frequency characteristics forced on each training set although the regression model is applied to $\Delta P(t_j, f_i)_{pret}$. Because $P_{inct}(f_i)$ and $P_{\Delta}(f_i)$ usually are small enough, we limit the inaccuracy from those power-irrelevant IPC values while reserving the positive relation between most IPC values and power dissipation.

As aforementioned, one shortcoming of using IPC solely is the low accuracy produced when the values of IPC are either too high or too low. In order to constrict this marginal effect, we have to manipulate the given training benchmark set accordingly. First, we order the training set T with descending IPC, which yields $T_{ordered}$. Second, we divide $T_{ordered}$ into three categories with respect of their IPC values. Heuristic results, based on the average accuracy provided, show that the separating points locate approximately at 0.87 and 1.86. As a result, there are three groups of benchmarks: the one with relative low IPC, T_{low}, with average normal IPC, T_{normal}, and with relative high IPC, T_{high}. For each group, we apply the same method to obtain $P(t_{IPC\_level}, f_i)$, $IPC(t_{IPC\_level}, f_i)$, $P_{inct}(t_{IPC\_level}, f_i)$, and $P_{\Delta}(t_{IPC\_level}, f_i)$, where *IPC_level* represents *low*, *high*, and *normal*.

We use an accumulative approach for modeling multiple cores based on the assumption that each core has similar power behavior. Therefore, we apply the single core model to each core in the system. Specifically, we express the total power dissipation estimation as follows:

$$P(a_j, f_i)_{pret\_total} = \sum_{k=1}^{k=cores} (\Delta P(a_j, f_i, k)_{pret} + P(f_i)) \tag{5}$$

where a_j is the target benchmark. $\Delta P(a_j, f_i, k)_{pret}$ is generated at per core level because different cores might have different $\Delta IPC(t_i, f_i, k)$. Fortunately, the modern multiple processor supports per core level PMCs. According to the modern processor architecture, however, the formula needs to be modified because $P(f_i)$ accounts for the power consumed by shared resources that should not be replicated. One example of the shared resources is L2 cache. To recalculate it, we introduce another parameter that should be determined at the training stage, $P_{shared}(k)$. In order to retrieve information on $P_{shared}(k)$, we re-execute training benchmarks on k cores, and select median value as $P_{shared}(k)$ for each k. The values of $P_{shared}(k)$ are different, which is determined by the total number of cores utilized by a task simultaneously. The bigger k is, the larger $P_{shared}(k)$ could be. The final formula to estimate the power dissipation of a_j of a multicore processor is the following:

$$\begin{aligned} P(a_j, f_i)_{pret\_total} &= \sum_{k=1}^{k=cores} (\Delta P(a_j, f_i, k)_{pret} + P(f_i)) \\ &= \sum_{k=1}^{k=cores} (P_{inct}(f_i) + P_{\Delta}(f_i) \times \Delta IPC(a_j, f_i, k)) \\ &\quad + \sum_{k=1}^{k=cores} P(f_i) - P_{shared}(k) \end{aligned} \tag{6}$$

3.2.2. Design microbenchmarks

The power model we proposed decides which benchmarks we need. This is an important step because inappropriate choices will lead to inaccuracy. First, a wide range of IPC value needs to be covered by training benchmarks. It is extremely important to test two margins of benchmarks with smaller or larger IPC values since we observe different power behaviors affected by IPC at those ranges. Second, an even distribution of benchmarks according to their IPC values is preferred. In the power model, we divide training benchmarks into three groups based on IPC values. It is more informative if the number of training benchmarks resides in each group equally.

However, it is unrealistic to consider all cases especially we only use one PMCs. Even worse, there is no information about which subunit is exercising by only profiling IPC. For example, two workloads stressing integer and cache respectively probably have the same IPC values, yet the integer benchmark might consume less power than the cache operation does. Besides, the power dissipation is also affected by the inputs. The same FFT algorithm might produce more power dissipation for a larger input size. In conclusion, it is critical to generate proper training workloads covering a sufficient variety CPU activities for a linear regression based approach.

In our study, we implement totally 36 benchmarks exercising various CPU components, such as INT, FP, and BPU. In order to emphasize the simplicity and applicability of the power model, we select 12 workloads covering maximum subunits, occupying a wide range of IPC values, and fairly even distributed. We list the benchmarks utilized in our study as Table 2. In general, the workloads exercise most of the processor subunits separately. The last three benchmarks utilize several components together to form mix benchmarks.

In the next section, we will discuss the method of relating the power behavior to the source code level profiling. Basically, based on the power model proposed, we design APIs locating source code function blocks according to the estimated power dissipation.

Table 2
Training benchmarks suite.

Microbenchmark	Description	Approx. IPC
INT(1)	Arithmetic integer operation	0.50
INT(2)	Arithmetic integer operation	1.62
INT(3)	Arithmetic integer operation	2.65
FP(1)	Arithmetic floating point operation	0.48
FP(2)	Arithmetic floating point operation	1.12
FP(3)	Arithmetic floating point	1.43
Cache(1)	Cache line reading	0.12
Cache(2)	Cache line reading	2.15
BP	Brunch prediction	1.00
IS	Insertion sort integers	1.77
ISF	Insertion sort floating point	2.26
QUICK	Quick sort integer	1.01

4. SPAN design and implementation

We are now in a position to automate the process of power profiling and correlate power dissipation to source code functions. We argue that it is crucial to design a PMC-based power estimator to association with source code based on two reasons. On one hand, it is convenient for software developers to identify their source code with actual power dissipation phases before any power/energy optimization. This will give developers more detail information of where their power/energy optimization should target on. On the other hand, PMC-based approach is relatively easy to apply in reality. On the contrary, hardware instrumentation definitely offers high accuracy; however, in practical, this method is limited due to hardware requirements.

We design a tool, SPAN, to provide live, real time power phases information of running applications. Generally, given a power model, there are two methods to enable synchronization between power dissipation and source code. The first approach is run-time instrumentation at binary-level. This method usually has a high granularity control over the execution. Because our approach mainly assists developers, we adopt the second option that specifies a suite of external API calls to correlate power estimation with application source codes. We refer to it as source code level instrumentation. The advantages of this method include the following items: lower overhead, applicability, and independence against instrumentation tools, such as PIN [4]. However, our approach requires developers to add some code manually to call the SPAN APIs.

The basic flow of the SPAN tool is illustrated in Fig. 4(a). The two inputs of SPAN are the application information and the PMC values. At the application level, the app information and the estimation control APIs are passed to the control thread through the designed SPAN APIs. Utilizing the run-time PMC values by calling the system call, the analyzer thread applies the power model proposed in Section 3.2 to estimate the power dissipation. Finally, the SPAN outputs a figure of estimated power dissipation represented by different colors, such as Fig. 4(b) shows.

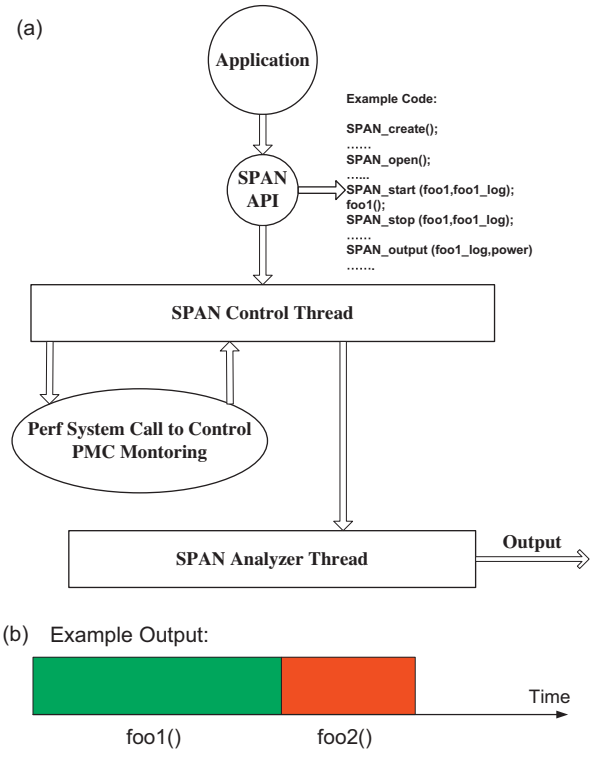

Fig. 4. Design of SPAN. (a) The flow chart of SPAN. (b) An example output of SPAN.

Table 3
SPAN APIs.

APIs	Description
`span_create()`	Prepare a power model profile which records basic parameters
`span_open()`	Initialize a SPAN control thread and targeting PMCs
`span_start(char* func, char* log)`	Record the targeting application function and specify the log file name
`span_stop(char* func, char* log)`	Stop the power estimation for a specified app function
`span_pause()`	Temporally stop reading PMCs
`span_continue()`	Resume reading PMCs
`span_change_rate(int freq)`	Shift the estimation rate, basically this methods control the PMC sampling rate
`span_change_model(float* model, File* model)`	Modify the model parameters in the model file according to the platform
`span_close()`	Close the opened PMCs and SPAN control thread
`span_output(char* log, FILE* power)`	Invoke SPAN analyzer thread and produce the detailed power estimation information with respect to the profiled functions to the destination file

In order to support the proposed mechanism, it is critical to provide a set of flexible APIs to applications. We show some of the designed SPAN APIs in Table 3. Currently, we implemented a preliminary C library of these APIs.

Given these APIs, the SPAN works as follows. First, we prepare a default file describing a set of power model parameters and an estimation frequency by calling span_c-reate(). Once the targeting application runs, PMCs are opened for each core respectively by calling span_open(); then, a SPAN control thread, which stores the row PMC information and the application function information (e.g., function name and start time), is invoked before each profiling function. The recording continues until we call the span_stop or span_pause(). The output is generated and stored into another file finally.

5. Validation and evaluation

We mainly evaluate our approach in two categories. First we need to discuss the accuracy of the power model. The second part covers the evaluation of SPAN on the source code level power estimation.

5.1. Environments

Specifically, we evaluate the power model on two different platforms, *Asus_intel_4* and *HP_amd_6*, where 4 and 6 represent the number of cores on each CPU respectively. The hardware configuration of each platform can be found in Table 4. We estimate the power generated by the `SPEC2008Cjvm`[3] benchmarks to validate the power model. We use Java version 1.6.0_18 on both platforms to launch each benchmark. The warm time is set to 5 min, and the iteration time is 10 min. We change *–bt* option to change the number of threads. We plan to restrict the CPU affinity to one core during the training process originally, which will minimize CPU migrations and provide a set of more optimized model parameters, but the assumption of no CPU migration conflict with the reality. Therefore, the system does not restrict CPU affinity in all of our training and evaluation process. The PMCs values are collected using the

Table 4
System configurations.

	Platform	
	Asus_intel_4	*HP_amd_6*
Model	Asus Essentio CM5570	HP Pavilion Elite HPE-000
CPU	Intel Q8200	AMD Phenom
Core frequencies	2.34 GHz, 2.00 GHz	2.6 GHz, 2.0 GHz, 1.4 GHz
# of cores	4	6
Memory	DDR3 6GB	DDR3 8GB
OS	Linux 2.6.31	Linux 2.6.31

kernel system call [2], _NR_perf_event_open(), which starts to be available in Linux kernel version 2.6.31.

Leakage power becomes a non-trivial portion of the power budget on modern superscalar processors. Experimental results show that leakage current increases exponentially with the supply voltage [48]; however, given a specific CPU frequency and supply voltage, as the input of our model, the leakage power is fixed. Besides, our power model mainly focuses on the dynamic power dissipation generated by a given workload. Therefore, we do not incorporate the leakage power in our power model.

In order to minimize the temperature effect on power, after each valid run, we set 10 min as cooling time. The static power is measured before each execution, and we guarantee the variation of the static power is less than 5% so that the results are comparable. It is worth noticing that there only exists neglectable static power variation for different operating frequencies [12]. Meanwhile, we use hardware measurement to collect power dissipation information on the processor as well. The results are compared with the estimated power dissipation in the next section.

5.2. Power model evaluation

The first step of using our power model is to generate a set of parameters from the training benchmarks. Some of the detailed parameters we derived from the training process are listed in Table 5. We can easily observe that the effects of IPC on power drop considerably at both margins: the IPC below 1.0 and beyond 2.0.

We evaluate our model in terms of accuracy. More and more research on power estimation techniques argues that accuracy is not the only aspect we should focus on [8,20]. However, other characteristics, such as responsiveness, depends on acceptable accuracy. In addition, the power model usually provides reasonable responsiveness if it has high accuracy. We run `SPEC2008Cjvm` benchmarks with multi-threads on possible frequencies to collect data. The errors are reported for the whole processor.

Through Fig. 5(a)–(d) shows the percentage error from a single core to the maximum four cores running 10 different benchmarks on *Asus_intel_4*. As the figures illustrate, generally, there is an incremental relationship between error rate and the number of cores. The possible reason is that we do not consider the shared resource in a fine granularity in the power model due to the PMCs limit. In addition, the inter-core communications, which are another major source of power dissipation, cannot be captured by the power model simply deploying one PMC. Given such limited information, our model achieves 5.17% absolute error rate on average, with standard deviation of 5.40%.

Fig. 5(e) summarizes the estimated error under frequency 2.00 GHz on *Asus_intel_4*. Our model is able to achieve smaller error rate since the power dissipation for each benchmark decreases and falls into a narrow range, which is less unpredictable than the scenario of high frequency. The power dissipation of some particular benchmarks, such as *crypto.aes*, presents a low correlation coefficient to the IPC and extensive usage of other processor components, such as brunch prediction units.

Similarly, from Fig. 6(a)–(d), we report experimental results of our power model on *HP_amd_6*. We control the maximum and average absolute error rate to 11.26% and 4.46% respectively for up to the six-core scenario, with a vast majority of estimates exhibiting very small errors. Besides, it is worth mentioning that our model does not consistently under- or over-estimate the power across the benchmark suite. We summarize the experiment results on frequencies of 2.00 GHz and 1.40 GHz in Fig. 6(e) and (f) respectively, with average error rate of 3.14%.

5.3. SPAN evaluation

After illustrating the error rate of our model, in this subsection, we discuss the effectiveness of SPAN in detail. As it was noted in Section 4, the SPAN is a source code instrumentation technique that keeps tracking power dissipation of each function block. We mainly evaluate two aspects of the SPAN, the overhead and the responsiveness. We focus on two benchmarks for the testing purpose. One is the FT benchmark from NAS parallel benchmark suite. Another is a synthetic benchmark that we designed with the combination of integer operation, *PI* calculation, prime calculation, and bubble sort.

The overhead of instrumentation on both testing benchmarks is negligible. First, we measured the execution with and without the SPAN instrumentation for ten times each. The differences of execution time are within 1% on average. The reasons of low overhead are as follows: the instrumentation is at the source code function-level, which barely adds interruptions during executions; the PMCs used in the model are limited to the minimum values, which further reduce the computation and communication cost of SPAN. Second, we measured the power dissipation of the benchmarks with and without underneath SPAN threads that record counter values. The overall variance across the whole execution lies within 2% in ten valid runs. Considering other factors, such as tem-

Table 5
Derived power model parameters.

System settings	$P(f_i)$	$IPC(f_i)$	$P_{incr}(f_i)$	$P_{\Delta}(f_i)$	$P_{shared}(k)$	IPC range
Asus Essentio CM5570, single-core, 2.36 GHz	15.74	0.49	−1.28E−15	1.79	0	0–1.0
	19.47	1.28	−0.60	4.41	0	1.0–2.0
	21.52	2.21	1.50E−15	1.49	0	Beyond 2.0
Asus Essentio CM5570, two-core, 2.36 GHz	15.74	0.49	−1.28E−15	1.79	10.44	0–1.0
	19.47	1.28	−0.60	4.41	10.44	0–2.0
	21.52	2.21	1.50E−15	1.49	10.44	Beyond 2.0
HP Pavilion Elite HPE-000, single-core, 2.6 GHz	25.55	0.45	0.18	0.87	0	0–1.0
	27.26	1.35	−0.10	1.58	0	0–2.0
	27.50	1.99	0.06	−0.18	0	Beyond 2.0
HP Pavilion Elite HPE-000, two-core, 2.6 GHz	25.55	0.45	0.18	0.87	18.84	0–1.0
	27.26	1.35	−0.10	1.58	18.84	0–2.0
	27.50	1.99	0.06	−0.18	18.84	Beyond 2.0

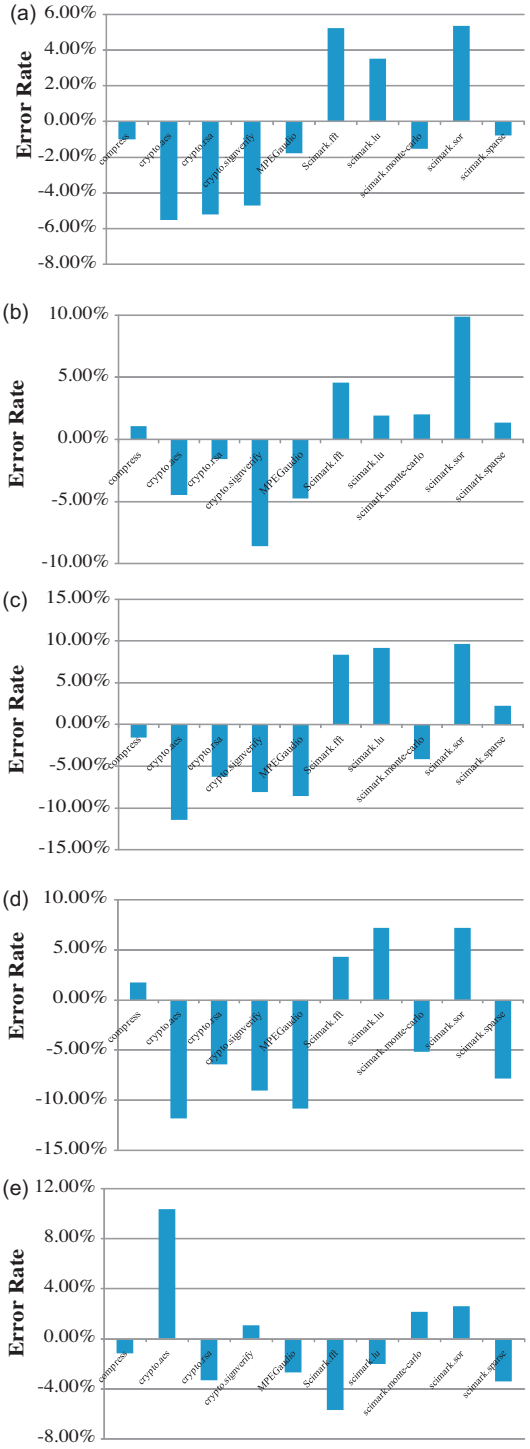

Fig. 5. Estimation error of SPEC 2008Cjvm on *Asus_intel_4*. (a) 2.34 GHz, one-core; (b) 2.34 GHz, two-core; (c) 2.34 GHz, three-core; (d) 2.34 GHz, four-core; (e) summarized, 2.00 GHz.

perature and power supply variation, 2% is a reasonable range in reality.

Though there is no standard method to evaluate the responsiveness of a power model, one of the simple and effective approaches are comparing the continuous measured and estimated power values. We utilize two multimeters storing the power dissipation of the target computer consistently into an assistant computer with the interval of one second. The benchmarks are executed on the *Asus_intel_4* platform with the SPAN source code instrumen-

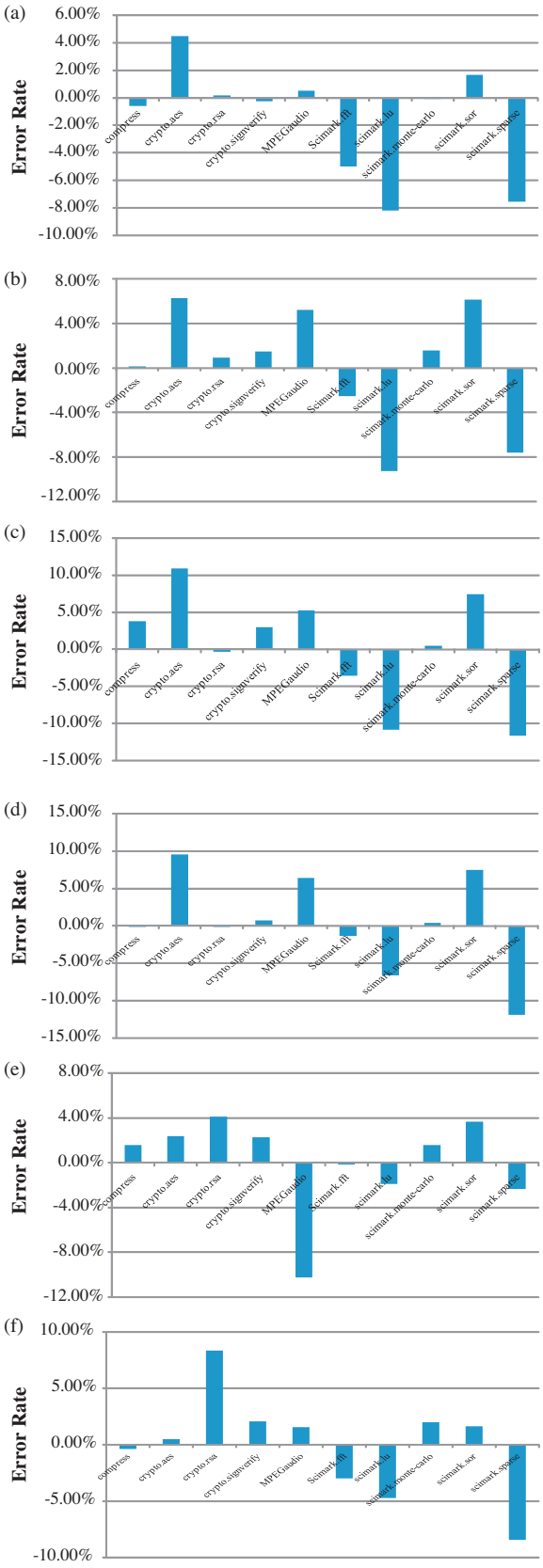

Fig. 6. Estimation error of SPEC 2008Cjvm on *HP_amd_6*. (a) 2.66 GHz, one-core; (b) 2.66 GHz, two-core; (c) 2.66 GHz, four-core; (d) 2.66 GHz, six-core; (e) summarized, 2.00 GHz; (f) summarized, 1.40 GHz.

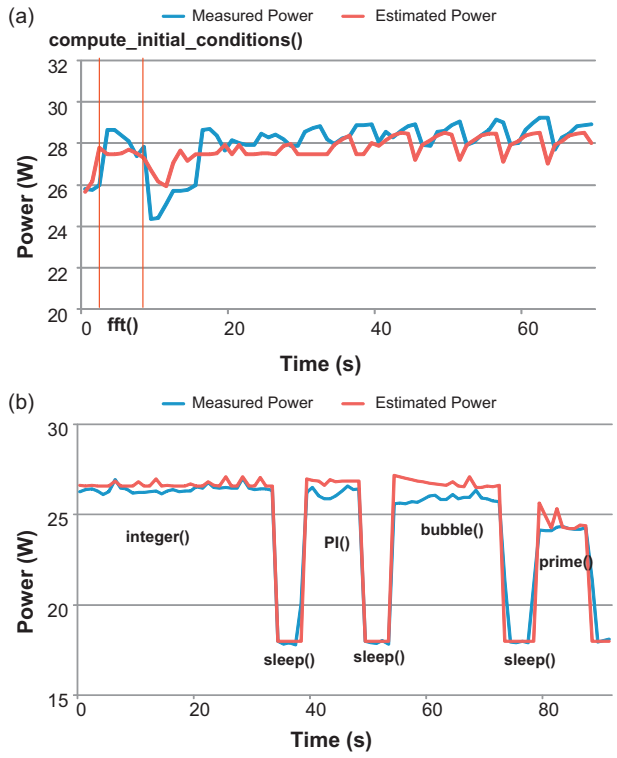

Fig. 7. Results of the SPAN evaluation on two benchmarks: (a) the FT and (b) synthetic benchmark with SPAN instrumentation.

tation to estimate the power. We plot the results in Fig. 7. It is easy to observe that the estimated power is closely related to the measured power dissipation at the overall shape. We also mark the corresponding benchmark functions in each figure. The first iteration of benchmark FT mainly consists of two functions, *compute_initial_conditions*() and *fft*(); then, the rest iterations follow the same procedure, which can be clearly observed from Fig. 7(a). But the estimations present a certain level of delay due to the rapid function changes in the source code. Moreover, in Fig. 7(b), we deliberately insert *sleep*() function between each sub benchmark in the synthetic workload in order to distinguish each one of them easily. We achieve the error rate as low as 2.34% for the two benchmarks on average.

6. Related work

Since we have already summarized a significant amount of work on power profiling, in this section we describe several previous efforts that are most related to *SPAN* from two aspects: *PMC-based power models* and *program power behavior analysis*.

6.1. PMC-based power models

Hardware performance counters are a set of special-purpose registers built into modern microprocessors to store the counts of hardware-related activities within computer systems. Researchers often rely on those counters to conduct low-level performance analysis or tuning.

Frank Bellosa is one of the first proponents of applying PMCs to investigate the energy usage patterns and finding the correlation of hardware events and system power [7]. He uses information about active hardware units to establish a thread-specific energy accounting, and then he uses the power information for energy-aware scheduling policies. Kadayif et al. design a tool called Virtual Energy Counters (vEC), which is built on top of the Perfmon user library. Their power model mainly considers cache related performance counters. In [25], Isci and Martonosi divide the processor into 22 function units and finds the relationship between the counters and those units. Although their results are very accurate, it is hard to be used on new platforms. Contreras and Martonosi [14] discover the power-IPC correlation and use five PMCs to estimate the power of workloads running on different CPU frequencies.

In [42], Powell et al. proposes a methodology to reduce the number of performance counters. They estimate the hardware activity events of several microarchitectural structures. Then, they associate the activity events with the power dissipation of such structures. Bertran et al. [8] demonstrate an alternative approach of using PMCs on CPU power estimation. Rather than directly deriving a power model using PMCs, they propose a method to treat each component of CPU separately, such as FE, INT and FP. Combining all the training parameters, they develop a fine-granularity power model. However, the training process is time-consuming to be extensively used in practical. In addition, the power model highly depends on the microarchitecture of the CPU.

Our main difference from all these works is that we combine the CPU frequency scaling and multicore features in the power model, which fits the trend of microprocessor design recently. Besides, our power model only employs one IPC. Other models [8] can achieve better accuracy and less variance compared with ours by collecting a number of counter values and training with more microbenchmarks, but barely can their models be applied to reality because of the model complexity.

6.2. Program power behavior analysis

Understanding program behavior is at the foundation of computer architecture and program optimization [46]. As energy consumption becomes one of the most important design considerations, researchers also evaluate the power and performance during the software development period. Program power behavior analysis cannot only help us optimize the energy efficiency of the applications, but also help the systems intelligently schedule the tasks by using new power-aware scheduling algorithms [5,32].

PowerScope [17] is one of the first work that map energy consumption to program structure. They develop a user-level daemon process and modify several system calls of the NetBSD kernel to sample process activity. Furthermore, they monitor energy consumption with collected data via a group of multimeters that is connected to the power source. Finally, synchronization with the System Monitor is provided by connecting the multimeter's external trigger input and output to pins on the parallel port of the profiling computer.

Similar with PowerScope, Ge et al. use their platform called PowerPack, a hardware-based power measurement and profiling platform, to analyze the application power behavior [18]. They insert a set of user-level APIs, such as pmeter_start_session and pmeter_end_session, before and after the code region of interest to map the power profile to the source code. Further more they analyze the power efficiency on multi-core platforms. The method they use to map power profile into program code is similar to our work; whereas, our approach is a pure software-based approach, and do not employ any hardware.

Isci et al. use the similarity matrix approach of [46] to deduce power phase behavior over the program runtime. Then they use component-based power breakdowns, computed by their power models, to identify power phases of programs. Their power model, however, is difficult to obtain because of PMCs limits.

7. Conclusion and future directions

In this paper, we first survey the related techniques in the field of power measurements and profiling comprehensively by classifying

the previous work into three main categories, totally five subgroups. We then present a novel practical power modeling method based on performance monitoring counters (PMCs) by employing one PMC and 12 training benchmarks on two recent multicore processors. Based on the model, we design and implement SPAN to map the run-time power dissipation to application functions. We evaluate both the power model and SPAN on two modern multicore systems. Despite the limited information provided by only one PMC, we achieve an absolute error rate of 5.17% and 4.46% on the two platforms by using benchmarks from `SPEC2008Cjvm` suite. In addition, it shows fairly stable accuracy under different frequencies. We also collect empirical data to validate the SPAN tool. Using the FT benchmark from NAS Parallel benchmark suite and the synthetic workload, we reach accuracy as high as 97% on average.

Power measurements and profiling have already been studied extensively at different levels; however, more investigations are needed in the following two areas: improving power profiling techniques and using these strategies in power-aware software design.

Accuracy is not the only important requirement for power measurements and profiling. We envision that simplicity and adaptability are also very interesting aspects. Simple power models are needed to supply live power information for systems, otherwise the overhead will be too high to be used. In addition, as the number of cores on a single chip keeps increasing, on-chip network fabrics become one of the main power dissipation resources. Thus, future research needs to consider this unit and reevaluate the power indicators that are currently used. Furthermore, we still need to break down the power dissipated on shared resources such as caches, and find suitable indicators to break down higher level power information into lower levels.

Currently, power measurements and profiling has been used for making power and performance trade-offs during hardware designs. Exploiting how to use power profiling approaches to design energy efficient software is also an interesting direction, which is highly related to the research activities in operating systems and software engineering fields.

References

[1] Ibm powerexecutive. URL: http://www-03.ibm.com/systems/management/director/about/director52/extensions/powerexec.html.

[2] Perf wiki, September 2009. URL: https://perf.wiki.kernel.org/index.php/Main_Page.

[3] Specjvm 2008 v1.01, June 2009. URL: http://www.spec.org/download.html.

[4] Pin-a binary instrumentation tool, August 2010. URL: http://www.pintool.org/.

[5] I. Ahmad, S. Ranka, S.U. Khan, Using game theory for scheduling tasks on multicore processors for simultaneous optimization of performance and energy, in: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, April, 2008, pp. 1–6, doi:10.1109/IPDPS.2008.4536420.

[6] L.A. Barroso, U. Hölzle, The case for energy-proportional computing, Computer 40 (12) (2007) 33–37, ISSN: 0018-9162. doi: http://dx.doi.org/10.1109/MC.2007.443.

[7] F. Bellosa, The benefits of event: driven energy accounting in power-sensitive systems, in: EW 9: Proceedings of the 9th Workshop on ACM SIGOPS European Workshop, New York, NY, USA, ACM Press, 2000, pp. 37–42, ISBN: 1-23456-789-0. doi: http://doi.acm.org/10.1145/566726.566736.

[8] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguade, Decomposable and responsive power models for multicore processors using performance counters, in: Proceedings of the 24th ACM International Conference on Supercomputing, ACM Press, 2010, pp. 147–158.

[9] W.L. Bircher, M. Valluri, J. Law, L.K. John, Runtime identification of microprocessor energy saving opportunities, in: ISLPED 05: Proceedings of the 2005 International Symposium on Low Power Electronics and Design, ACM Press, 2005, pp. 275–280.

[10] D. Brooks, V. Tiwari, M. Martonosi, Wattch: a framework for architectural-level power analysis and optimizations, in: ISCA'00: Proceedings of the 27th Annual International Symposium on Computer Architecture, New York, NY, USA, ACM Press, 2000, pp. 83–94, ISBN: 1-58113-232-8. doi: http://doi.acm.org/10.1145/339647.339657.

[11] F. Chang, K. Farkas, P. Ranganathan, Energy-driven statistical sampling: detecting software hotspots, in: B. Falsafi, T. Vijaykumar (Eds.), Power-Aware Computer Systems, volume 2325 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2003, pp. 105–108.

[12] H. Chen, S. Wang, W. Shi, Where does the power go in a computer system: experimental analysis and implications. Computing Science Technical Report MIST-TR-2010-004, Wayne State University, Detroit, MI, 2010.

[13] J. Chen, M. Dubois, P. Stenström, Simwattch: integrating complete-system and user-level performance and power simulators, IEEE Micro 27 (4) (2007) 34–48, ISSN: 0272-1732. doi: http://dx.doi.org/10.1109/MM.2007.73.

[14] G. Contreras, M. Martonosi, Power prediction for intel xscale processors using performance monitoring unit events, in: Proceedings of IEEE/ACM International Symposium on Low Power Electronics and Design, 2005, pp. 221–226.

[15] G. Dhiman, K. Mihic, T. Rosing, A system for online power prediction in virtualized environments using Gaussian mixture models, in: Proceedings of the 47th ACM IEEE Design Automation Conference, ACM Press, 2010, pp. 807–812.

[16] N. Eisley, V. Soteriou, L. Peh, High-level power analysis for multi-core chips, in: CASES'06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, New York, NY, USA, ACM Press, 2006, pp. 389–400, ISBN: 1-59593-543-6. doi: http://doi.acm.org/10.1145/1176760.1176807.

[17] J. Flinn, M. Satyanarayanan, Powerscope: a tool for profiling the energy usage of mobile applications, in: WMCSA'99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications, Washington, DC, USA, IEEE Computer Society, 1999, p. 2, ISBN: 0-7695-0025-0.

[18] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, K.W. Cameron, Powerpack: energy profiling and analysis of high-performance systems and applications, in: IEEE Transactions on Parallel and Distributed Systems, 99 (RapidPosts), 2009, pp. 658–671, ISSN: 1045-9219. doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.76.

[19] R.A. Giri, A. Vanchi, Increasing data center efficiency with server power measurements. Technical report, Intel Information Technology, 2010.

[20] B. Goel, S.A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, M. Cesati, Portable, scalable, per-core power estimation for intelligent resource management, in: The First International Green Computing Conference, 2010.

[21] S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, L.K. John, Using complete machine simulation for software power estimation: the softwatt approach, in: HPCA'02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture, Washington, DC, USA, IEEE Computer Society, 2002, p. 141.

[22] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, DRPM: dynamic speed control for power management in server class disks, SIGARCH Comput. Archit. News 31 (2) (2003) 169–181, ISSN: 0163-5964. doi: http://doi.acm.org/10.1145/871656.859638.

[23] C. Hu, D.A. Jimenez, U. Kremer, Toward an evaluation infrastructure for power and energy optimizations, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) – Workshop 11, Washington, DC, USA, IEEE Computer Society, 2005, p. 230.2, ISBN: 0-7695-2312-9. doi: http://dx.doi.org/10.1109/IPDPS.2005.437.

[24] IPMI. Intelligent platform management interface. URL: http://www.intel.com/design/servers/ipmi/index.htm.

[25] C. Isci, M. Martonosi, Runtime power monitoring in high-end processors: methodology and empirical data, in: MICRO 36: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, IEEE Computer Society, 2003, p. 93, ISBN: 0-7695-2043-X.

[26] C. Isci, M. Martonosi, Phase characterization for power: evaluating control-flow-based and event-counter-based techniques, in: The Twelfth International Symposium on High-Performance Computer Architecture, February, 2006, pp. 121–132, doi:10.1109/HPCA.2006.1598119.

[27] B. Jacob, S.W. Ng, D.T. Wang, Memory Systems: Cache, DRAM, Disk, Denise E.M. Penrose, 2007, pp. 43–44.

[28] R. Joseph, M. Martonosi, Run-time power estimation in high-performance microprocessors, in: Proceedings of the 2001 International Symposium on Low Power Electronics and Design, Huntington Beach, CA, United States, 2001.

[29] R. Joseph, D. Brooks, M. Martonosi, Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs, in: Workshop on Complexity Effective Design WCED, held in conjunction with ISCA-28, June, 2001.

[30] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, A. Sivasubramaniam, vEC: virtual energy counters, in: PASTE'01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, New York, NY, USA, ACM Press, 2001, pp. 28–31, ISBN: 1-58113-413-4. doi: http://doi.acm.org/10.1145/379605.379639.

[31] S. Kamil, J. Shalf, E. Strohmaier, Power efficiency in high performance computing, in: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, April, 2008, pp. 1–8, doi:10.1109/IPDPS.2008.4536223.

[32] S.U. Khan, I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, IEEE Trans. Parallel Distrib. Syst. 20 (March (3)) (2009) 346–360, ISSN: 1045-9219, doi:10.1109/TPDS.2008.83.

[33] B.C. Lee, D.M. Brooks, Accurate and efficient regression modeling for microarchitectural performance and power prediction, in: Proceedings of 12th ACM Symposium on Architectural Support for Programming Languages and Operating Systems, October, 2006, pp. 185–194.

[34] T. Li, L.K. John, Run-time modeling and estimation of operating system power consumption, SIGMETRICS Perform. Eval. Rev. 31 (1) (2003) 160–171, ISSN: 0163-5999. doi: http://doi.acm.org/10.1145/885651.781048.

[35] D. Liu, C. Svensson, Power consumption estimation in CMOS VLSI chips, IEEE J. Solid-State Circuits 29 (June (6)) (1994) 663–670, ISSN: 0018-9200. doi:10.1109/4.293111.

[36] J.R. Lorch, A.J. Smith, Energy consumption of Apple Macintosh computers. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1997.
[37] J.R. Lorch, A.J. Smith, Apple Macintosh's energy consumption, IEEE Micro 18 (6) (1998) 54–63, ISSN: 0272-1732. doi: http://dx.doi.org/10.1109/40.743684.
[38] Y.-H. Lu, L. Benini, G. De Micheli, Power-aware operating systems for interactive systems, IEEE Trans. Very Large Scale Integr. Syst. 10 (2) (2002) 119–134, ISSN: 1063-8210. doi: http://dx.doi.org/10.1109/92.994989.
[39] D. Marculescu, R. Marculescu, M. Pedram, Information theoretic measures of energy consumption at register transfer level, in: ISLPED'95: Proceedings of the 1995 International Symposium on Low Power Design, New York, NY, USA, ACM Press, 1995, pp. 81–86, ISBN: 0-89791-744-8. doi: http://doi.acm.org/10.1145/224081.224096.
[40] D. Molaro, H. Payer, D. Le Moal, Tempo: disk drive power consumption characterization and modeling, in: ISCE'09. IEEE 13th International Symposium on Consumer Electronics, May, 2009, pp. 246–250, doi:10.1109/ISCE.2009.5156863.
[41] C.S. Patel, et al., Power constrained design of multiprocessor interconnection networks, in: Proceedings of the 1997 International Conference on Computer Design (ICCD'97), Washington, DC, USA, IEEE Computer Society, 1997, p. 408, ISBN: 0-8186-8206-X.
[42] M. Powell, J. Emer, A. Biswas, S. Mukherjee, B. Sheikh, S. Yardi, CAMP: a technique to estimate per-structure power at run-time using a few simple parameters, in: IEEE 15th International Symposium on High Performance Computer Architecture, HPCA, 2009, pp. 289–300.
[43] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, F. Rawson, Application-aware power management, in: Proceeding of IEEE International Symposium on Performance Analysis of Systems and Software, October, 2006, pp. 39–48.
[44] J.T. Russell, M.F. Jacome, Software power estimation and optimization for high performance, 32-bit embedded processors, in: ICCD'98: Proceedings of the International Conference on Computer Design, Washington, DC, USA, IEEE Computer Society, 1998, p. 328, ISBN: 0-8186-9099-2.
[45] E. Saxe, Power-efficient software, Commun. ACM 53 (2) (2010) 44–48, ISSN: 0001-0782. doi: http://doi.acm.org/10.1145/1646353.1646370.
[46] T. Sherwood, E. Perelman, G. Hamerly, B. Calder, Automatically characterizing large scale program behavior, in: ASPLOS-X: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, New York, NY, USA, ACM Press, 2002, pp. 45–57, ISBN: 1-58113-574-2. doi: http://doi.acm.org/10.1145/605397.605403.
[47] K. Singh, M. Bhadauria, Real time power estimation and thread scheduling via performance counters, SIGARCH Comput. Archit. News 37 (2008) 46–55.
[48] A. Sinha, N. Ickes, A.P. Chandrakasan, Instruction level and operating system profiling for energy exposed software, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 11 (2003) 1044–1057.
[49] T.K. Tan, A. Raghunathan, G. Lakshminarayana, N.K. Jha, High-level software energy macro-modeling, 2001, pp. 605–610, doi:10.1109/DAC.2001.156211.
[50] V. Tiwari, S. Malik, A. Wolfe, Power analysis of embedded software: a first step towards software power minimization, IEEE Trans. Very Large Scale Integr. Syst. 2 (4) (1994) 437–445, ISSN: 1063-8210. doi: http://dx.doi.org/10.1109/92.335012.
[51] A. Vahdat, A. Lebeck, C.S. Ellis, Every Joule is precious: the case for revisiting operating system design for energy efficiency, in: EW 9: Proceedings of the 9th Workshop on ACM SIGOPS European Workshop, New York, NY, USA, ACM Press, 2000, pp. 31–36, ISBN: 1-23456-789-0. doi: http://doi.acm.org/10.1145/566726.566735.
[52] M.A. Viredaz, M.A. Viredaz, D.A. Wallach, D.A. Wallach, Power evaluation of Itsy version 2.3. Technical report, Compaq, Western Research Laboratory, 2000.
[53] H.-S. Wang, X. Zhu, L.-S. Peh, S. Malik, Orion: a power-performance simulator for interconnection networks, in: MICRO 35: Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture, Los Alamitos, CA, USA, IEEE Computer Society Press, 2002, pp. 294–305, ISBN: 0-7695-1859-1.
[54] W. Wu, L. Jin, J. Yang, A systematic method for functional unit power estimation in microprocessors, in: DAC 2006: Proceedings of the 43rd Annual Conference on Design Automation, ACM Press, 2006, pp. 554–557.
[55] T.T. Ye, G. De Micheli, L. Benini, Analysis of power consumption on switch fabrics in network routers, in: DAC'02: Proceedings of the 39th Annual Design Automation Conference, New York, NY, USA, ACM Press, 2002, pp. 524–529, ISBN: 1-58113-461-4. doi: http://doi.acm.org/10.1145/513918.514051.
[56] W. Ye, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, The design and use of simplepower: a cycle-accurate energy estimation tool, in: DAC'00: Proceedings of the 37th Annual Design Automation Conference, New York, NY, USA, ACM Press, 2000, pp. 340–345, ISBN: 1-58113-187-9. doi: http://doi.acm.org/10.1145/337292.337436.
[57] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, R. Wang, Modeling hard-disk power consumption, in: FAST'03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, USENIX Association, 2003, pp. 217–230.
[58] H. Zeng, C.S. Ellis, A.R. Lebeck, A. Vahdat, Ecosystem: managing energy as a first class operating system resource, SIGPLAN Not. 37 (10) (2002) 123–132, ISSN: 0362-1340. doi: http://doi.acm.org/10.1145/605432.605411.

Shinan Wang received his B.E. degree from Beijing University of Posts and Telecommunications (BUPT) in 2008. He is currently a third year Ph.D. student in Wayne State University, Detroit, MI. His current research interests include mobile computing, wireless healthcare, and computer system power profiling and management. He has co-authored a couple of journals and international conference papers including IEEE Transactions on Vehicular Technology and CollaborateCom 2010.

Hui Chen is a Ph.D. candidate in computer science at Wayne State University. His major research interests include power-aware computing, operating systems and computer networks. He received his B.S. degree in School of Mechano-electronic Engineering from Xidian University in 2004 and M.S. degree in computer science and engineering from Beijing Institute of Technology in 2008.

Weisong Shi is an Associate Professor of Computer Science at Wayne State University. He received his B.S. from Xidian University in 1995, and Ph.D. degree from the Chinese Academy of Sciences in 2000, both in Computer Engineering. His current research focuses on computer systems, mobile computing, and high performance computing. Dr. Shi has published more than 100 peer-reviewed journal and conference papers in these areas. He has also served on technical program committees of several international conferences, including WWW, ICDCS, MASS. He is a recipient of Microsoft Fellowship in 1999, the President outstanding award of the Chinese Academy of Sciences in 2000, one of 100 outstanding Ph.D. dissertations (China) in 2002, "Faculty Research Award" of Wayne State University in 2004 and 2005, the "Best Paper Award" of ICWE'04 and IPDPS'05. He is a recipient of the NSF CAREER award and Wayne State University Career Development Chair award.