Hindawi
Scientific Programming
Volume 2017, Article ID 8686971, 13 pages
https://doi.org/10.1155/2017/8686971

Research Article

Modeling the Power Variability of Core Speed Scaling on Homogeneous Multicore Systems

Zhihui Du,[1] Rong Ge,[2] Victor W. Lee,[3] Richard Vuduc,[4] David A. Bader,[4] and Ligang He[5]

[1]*Department of Computer Science and Technology, Tsinghua University, Beijing, China*
[2]*School of Computing, Clemson University, Clemson, SC, USA*
[3]*Intel Corporation, Santa Clara, CA, USA*
[4]*School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA, USA*
[5]*Department of Computer Science, University of Warwick, Coventry, UK*

Correspondence should be addressed to Zhihui Du; duzh@tsinghua.edu.cn

Received 10 June 2017; Accepted 18 September 2017; Published 25 October 2017

Academic Editor: Thomas Leich

We describe a family of power models that can capture the nonuniform power effects of speed scaling among homogeneous cores on multicore processors. These models depart from traditional ones, which assume that individual cores contribute to power consumption as independent entities. In our approach, we remove this independence assumption and employ statistical variables of core speed (average speed and the dispersion of the core speeds) to capture the comprehensive heterogeneous impact of subtle interactions among the underlying hardware. We systematically explore the model family, deriving basic and refined models that give progressively better fits, and analyze them in detail. The proposed methodology provides an easy way to build power models to reflect the realistic workings of current multicore processors more accurately. Moreover, unlike the existing lower-level power models that require knowledge of microarchitectural details of the CPU cores and the last level cache to capture core interdependency, ours are easier to use and scalable to emerging and future multicore architectures with more cores. These attributes make the models particularly useful to system users or algorithm designers who need a quick way to estimate power consumption. We evaluate the family of models on contemporary x86 multicore processors using the SPEC2006 benchmarks. Our best model yields an average predicted error as low as 5%.

1. Introduction

We consider the problem of how to model the power of a modern multicore processor as a function of the speed of its cores. On its surface, the problem seems simple as it is natural to assume that cores are independent of one another: the classic power model posits that the total processor power is the sum over that of independent cores. However, we find that in practice such modeling methods do not adequately capture what happens on real multicore systems in which there may be interactions among cores.

By way of motivation, let us consider the following classic model and then compare what it predicts to what happens in an actual experiment. In the classic single-core model, the power, P_{SC}, consumed by a core is expressed as the following function of its operating frequency ("speed"), f:

$$P_{\mathrm{SC}}(f) \propto \alpha \cdot f^{\beta}, \tag{1}$$

where α is a workload-dependent factor and $\beta \geq 1$ is a hardware technology-dependent parameter. For simplicity, (1) omits a term for constant (or static) power, but our argument and methods hold with or without the term. This model appears in a variety of papers on the power-aware scheduling problem [1, 2], in particular when the system provides dynamic voltage and frequency scaling (DVFS) [3, 4].

A widely adopted approach used for multicore power modeling extends from the method for single-core power

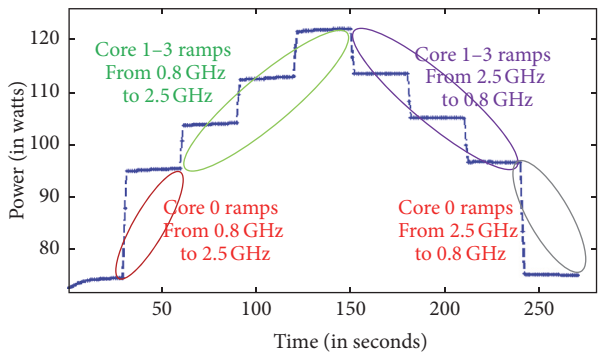


FIGURE 1: A motivating example to demonstrate that the power consumption of each core is determined by its own core speed and the speeds of other cores on the same chip. The employed AMD multicore processor has 4 cores with per-core DVFS. Initially, all cores run at the same frequency of 0.8 GHz. Then, one of the cores scales up its frequency by one step every 30 seconds until the frequency reaches the highest value of 2.5 GHz. This process repeats on another core until all cores run at the highest frequency. After that, the experiment continues in reverse until all cores drop their frequencies to 0.8 GHz.

modeling. It sums the power consumed by individual cores [5–8]. As a result, the power consumption of an N-core processor, denoted by P_{MC}, is calculated by

$$P_{\mathrm{MC}}(f_1, \ldots, f_N) = \alpha \sum_{c=1}^{N} f_c^{\beta}. \quad (2)$$

Critically, this approach assumes *independence*: the power of an individual core does not depend on what is happening on other cores on the same chip. Consider an environment consisting of multiple homogeneous cores, where all cores execute the same workload. In this setting, one may derive two predictions from (2). First, all cores contribute to the total power consumption independently. Second, scaling any core from one speed to another causes the same change in the total power consumption, regardless of the speed of the other cores. In other words, the cores have uniform power effects with speed scaling. For example, suppose a multicore processor has 16 cores with their frequencies set as $\langle f_0, f_1, \ldots, f_{15} \rangle$. If $f_i = f_j$, then changing the frequency of core i from f_i to $f_i + \delta_f$ causes a total power change of $\alpha((f_i + \delta_f)^{\beta} - f_i^{\beta})$, which will have the same value as $\alpha((f_j + \delta_f)^{\beta} - f_j^{\beta})$ if we change the frequency of core j from f_j to $f_j + \delta_f$.

However, the observations made in our experiments contradict these predictions. Figure 1 shows how the total processor power varies with a sequence of frequency scaling on a representative homogeneous multicore processor. In our experiments, all cores execute the same workload. The experimental results may be summarized as follows.

(i) The effect on power from speed-scaling a core *depends* on the states of the other cores. The resulting change in total power depends on whether the scaling updates the maximum speed among the cores. This observation contradicts the first prediction derived from (2).

(ii) The scaling that updates the maximum speed among the cores leads to a significantly larger change in total power than others. That is, the same increase in speed among the cores may have nonuniform power effects. This observation contradicts the second prediction derived from (2).

Thus, we may conclude that power models should account for interdependency and variability among the cores to estimate the power consumption of a multicore processor more accurately. Unfortunately, only a few studies [9–12] have investigated this issue. In general, these studies decompose a processor to its architectural components and use performance counters to infer the power consumption of each component. The effect of core interdependency on power consumption is explicitly captured through shared resources and differentiated behaviors of cores. Due to the use of hardware performance events, the models are detailed and complex. Furthermore, they have only been developed for dual- or quad-core processors. This approach is problematic when applied to emerging and future processors that may have eight or more cores.

Multicore processors that integrate a dozen or more DVFS-capable cores are commonplace today and manycore processors are pervasive. The goal of this study is to propose a family of practical power models that are accurate and easy to use and, at the same time, can be scaled to emerging and future multicore technologies. Our power models use two statistical parameters, *average speed* and *dispersion of speeds*, on cores. The former is used to accurately capture the holistic impact of multicore speeds while the latter captures the core dependencies. The evaluation shows that our models are more accurate than the traditional models by reflecting interdependence among cores but also maintain a similar level of simplicity. Our models are at the system level and eliminate the need to model individual architectural components with hardware performance events.

We explore this family of models systematically, to show how one can "derive" a suitable power model for multicore processors by experiments. We carry out the experiments using SPEC2006 [13] on contemporary multicore processors and ultimately obtain a "basic power model" with an average relative error of 3% (in absolute value) for most benchmarks. These results help bolster the practical case for using our approach. And for those applications in which the basic power model is not as accurate, we find that an improved piecewise model, which partitions the *maximum frequency* among the cores into a small number of segments, best expresses overall power consumption of a multicore processor.

We evaluated our approach systematically on current generations of Intel and AMD processors. To instantiate the model for a given application and processor, one needs to only run the applications on the processor a few times, each with a different setting of core speeds. Once fitted, the power models can be used to predict the power consumption at any settings of core speeds. Further, if in the future the processor

architectures evolve, the proposed family of models can still be applied, since the models take a general form with the statistical values of core speeds as input. In principle, one needs to only rerun the designed experiments to determine the new values of the coefficients in the model.

The model properties and results presented in this paper may enable future researchers to use more appropriate analytical frameworks to tackle a variety of power- and energy-aware algorithms and application design problems, including both classical scheduling algorithms under DVFS and emerging scheduling problems such as the problem of how to assign work to cores and set core speeds to satisfy a power bound [14].

The main contributions of this work are as follows.

(i) The presented family of models accurately captures the nonuniform power effect of frequency scaling on multicore processors. Such models are much needed for power-aware, multicore-based HPC systems.

(ii) By using only a couple of high-level variables, the models are easy to use and can be applied to emerging and future processors with more cores.

(iii) The models are the first to use statistical measurements as model variables, in contrast to the commonly adopted complex approach that models individual cores and other microarchitectural components with hardware performance events.

(iv) The models in the family have different forms with different numbers of variables. It is at users' liberty to choose one that best suits their needs, such as balancing accuracy and complexity.

2. A Family of Multicore Power Models

The discussions of Figure 1 suggest that it may not be correct to model the power consumption of a multicore processor by modeling the power consumed by each individual core and then adding them together. Therefore, we propose a *family* of new models for estimating the power consumption of multicore processors. These models use *statistical measures* of core speeds, such as means and dispersions, as model variables.

Note that we focus on homogeneous multicore processors. Such an environment is common in parallel computing programmed by MPI and OpenMP, which are the dominant parallel programming paradigms for solving scientific and engineering problems. We leave the research on heterogeneous architectures to our future work.

2.1. The Model Family. The general form of the model family is as follows. Let $\overline{f}$ denote the average frequency of the cores in a multicore processor and Δ denote the dispersion of speeds among the cores. Below, we will consider several possible forms of Δ. Assuming that power consumption correlates with $\overline{f}$ and Δ, we posit a general model of the form

$$P_{\mathrm{MC}}\left(\overline{f},\Delta\right) \equiv a_0 + a_1 \cdot \overline{f} + a_2 \cdot \overline{f}^{k_2} + a_3 \cdot \Delta + a_4 \cdot \Delta^{k_4}, \quad (3)$$

where $\{a_0,\ldots,a_4\}$ and $\{k_2,k_4\}$ are the parameters to be estimated. In this general model, the average frequency is simply calculated by $\overline{f} \equiv (1/N)\sum_{c=1}^{N} f_c$, where N is the number of cores and $\{f_1,\ldots,f_N\}$ are their frequencies.

For Δ, a natural choice is the standard deviation among frequencies, denoted by σ. However, we also consider several more possibilities. Let $f^+ \equiv \max_{1\le c\le N} f_c$ denote the maximum frequency setting of any core and $f_- \equiv \min_{1\le c\le N} f_c$ be the minimum frequency. Thus, in addition to σ, we consider the following three measures of speed dispersion:

(i) Δ^+: the difference between the maximum frequency and the average frequency, namely, $f^+ - \overline{f}$.

(ii) Δ_-: the difference between the average frequency and the minimum frequency, namely, $\overline{f} - f_-$.

(iii) Δ_-^+: the difference between the maximum and minimum frequency, namely, $f^+ - f_-$.

In the proposed model family, instead of considering many individual core speeds, we only employ two statistical parameters to capture the typical speed distribution of all cores in a processor.

2.2. Candidate Models. From the general form of (3), we consider several specific cases as candidate models for fitting, denoted as $R1$ through $R5$ below:

$$\begin{aligned}
R1&: a_0 + a_1 \cdot \overline{f} + a_3 \cdot \Delta \\
R2&: a_0 + a_1 \cdot \overline{f} + a_4 \cdot \Delta^{k_4} \\
R3&: a_0 + a_2 \cdot \overline{f}^{k_2} + a_3 \cdot \Delta \\
R4&: a_0 + a_2 \cdot \overline{f}^{k_2} + a_4 \cdot \Delta^{k_4} \\
R5&: a_0 + a_1 \cdot \overline{f} + a_2 \cdot \overline{f}^{k_2} + a_3 \cdot \Delta + a_4 \cdot \Delta^{k_4}.
\end{aligned} \quad (4)$$

Note that $R5$ is the same as (3). The other cases simplify the general form.

Beyond $R1$ through $R5$, we consider two additional classic power models for comparison. One assumes a polynomial relation between power and frequency of each individual core ($R6$), and the other assumes a linear relationship ($R7$):

$$\begin{aligned}
R6&: P_{\mathrm{MC}}\left(f_1,\ldots,f_N\right) = a_0 + a_1 \cdot \sum_{c=1}^{N} f_c^{k_1} \\
R7&: P_{\mathrm{MC}}\left(f_1,\ldots,f_N\right) = a_0 + a_1 \cdot \sum_{c=1}^{N} f_c.
\end{aligned} \quad (5)$$

Note that fitting $R2$, $R3$, $R4$, $R5$, and $R6$ requires nonlinear regression methods, whereas simple linear regression is sufficient to fit $R1$ and $R7$.

2.3. Building the Power Models. The purpose of this work is to propose a methodology for system users or algorithm designers to build accurate and simple power models for current

and even future multicore processors. In this subsection, we present the methodology for building our power models.

The following procedure is used to determine which of the candidate models in Section 2.2 can best represent the power consumption of multicore processors.

In general, the procedure involves designing different frequency settings, running benchmark application(s) on the given modern multicore processor, and recording the power consumption and the corresponding frequency settings. More details of the procedure are described below.

2.3.1. Frequency Settings. We performed an (or approximately) exhausted test in training to understand the relationship between frequency and power. But in model setup runs, we only need to run the experiments with a small number of frequency settings using the following frequency sampling method, the principle of which is that a small number of frequencies still represent the full spectrum of all possible frequencies. If a multicore processor has N homogeneous cores and each core can be set at K different frequencies independently, the total number of frequency settings is $\binom{N+K-1}{N}$. For example, if $K = 16$ and $N = 4$, then $\binom{N+K-1}{N} = \binom{19}{4} = 5168$. For a large K, that is, a core has many different frequency levels, we select the minimum and maximum frequency and 2~3 additional frequencies in between to cover all the speed range. For a large N, that is, there are many cores in a multicore processor, we divide the cores into smaller groups, and all cores in a group are configured with the same frequency setting.

2.3.2. Monitoring Power Consumption. The tool for monitoring power consumption in the experiments can be a hardware power meter device or other software power measurement packages. The exemplar software power measurement packages are Intel's Running Average Power Limit (RAPL) interface [15] and other packages such as `likwid-powermeter` [16]. The accuracy of the RAPL-based power measurement tool is adequate for high-level power prediction.

2.3.3. Regression Analysis. Once the data are measured, we fit the candidate models, $R1$ through $R7$, to them using standard statistical parameter estimation procedures. Fits are specific to a processor, and we report on fit quality both for individual benchmarks and for mixed workloads (see Sections 4.2 and 4.3). Models $R2$ through $R6$ require nonlinear regression methods, whereas $R1$ and $R7$ may be fitted by standard linear regression procedures. Additionally, models $R2$ through $R6$ require determining both the coefficients (i.e., a_0–a_4) and the value of exponents (i.e., k_2 and k_4), whereas in $R1$ and $R7$, only the values of coefficients (i.e., a_0, a_1, and a_3 in $R1$ and a_0 and a_1 in $R7$) need to be determined.

2.3.4. Models Screening. Finally, after fitting each candidate model, we analyze the parameter values and the fitting quality of each model and identify which model best captures the relation between power consumption and core frequencies. Note that we only need to run an application on a multicore processor with a limited number of frequency settings to obtain the experimental data. Once we have established the power model, we can use the model to predict the power consumption under any frequency setting of the multicore processor.

3. Model Analysis and Refinement

In this section, we propose the basic model based on the method in the last section. The analysis shows that the basic model can be used for different optimization purposes. We also show the weakness of the basic model for some cases and how we improve it with the refined model.

3.1. The "Basic Model" and What It Implies. We have conducted extensive experiments on x86 multicore processors (see the experiment results in Section 4). After comparing the results obtained by our candidate models with those by the classic multicore power model, we find that $R1$, combined with the dispersion measure Δ^+, typically exhibits the best fit. Hereafter, we will refer to $R1$ as the *basic model*; that is,

$$P_{\text{basic}}\left(\overline{f}, \Delta^+\right) \equiv a_0 + a_1 \cdot \overline{f} + a_3 \cdot \Delta^+. \tag{6}$$

Observe that the basic model is *linear* with $\overline{f}$ and Δ^+. Although dynamic power is generally nonlinear with frequency, the relation we observed in reality on current processors appears to be linear approximately.

The basic model suggests that two different frequency settings may deliver the same throughput or performance for a given application but cause significantly different power consumption. For example, consider the following two different frequency distributions on four cores, which both have an average of 1.6 GHz: [1.6, 1.6, 1.6, 1.6] and [1.2, 1.6, 1.6, 2.0]. These have Δ^+ values of 0 GHz and 0.4 GHz, respectively. The classic multicore power model such as $R7$ will predict that the same amount of power will be consumed under these two frequency distributions. However, using (6), we can predict that the distribution with greater values of Δ^+ will cause more power consumption.

Among all frequency distributions, those with the minimum $\Delta^+ = 0$ define a theoretical Pareto frontier and will consequently consume the least amount of power. For example, consider Figure 2. This figure shows the measured power of benchmark 410.*bwaves* running on an Intel Core i7-2600K (a quad-core Sandy Bridge processor). The red line is the Pareto frontier obtained by the basic model. Each of the blue dots is the measured power when the application is running with a particular average frequency. It can be observed from this figure that, with the same average frequency, different frequency distributions make a huge difference with power consumption. In this figure, the optimal frequency distribution saves up to 48% of the power, compared with other frequency distributions. For a given power budget, the optimal frequency distribution can outperform naïve ones by up to 37.5%. Assume "F_r" in this figure corresponds to an initial frequency distribution with an average frequency and power consumption. Then, the basic model indicates that we can save power by following the vertical line down to A, or improve performance by

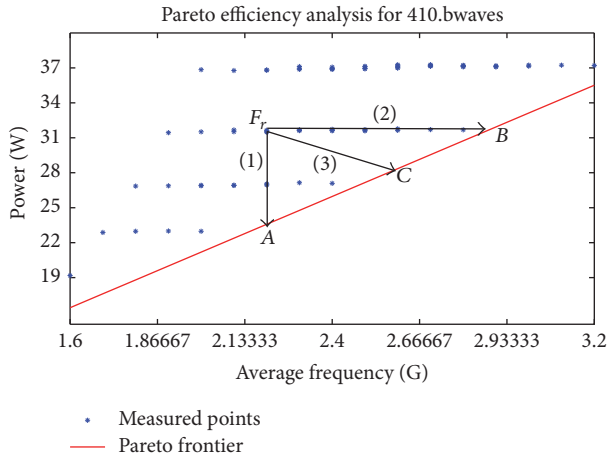


Figure 2: Theoretical Pareto frontier (in red) suggested by our model. From any user specified point F_r, following the vertical line (1), we can reach point A which can provide the same average speed with the lowest power. The power difference between F_r and A is the saved power. Following the horizontal line (2), we can reach point B which can provide the highest throughput (the fastest average speed) with the same power. The speed difference between B and F_r is the increased average speed. Following line (3), we can reach point C which can provide higher speed with less power than F_r.

following the horizontal line rightward to B, or balance both improvements by reaching point C.

3.2. Model Refinements. The basic model can be refined in certain contexts. For some of the benchmarks, such as `458.sjeng` of SPEC2006 [13] (see the experimental results in Table 2), the prediction result of the basic model is not very accurate. Digging deeper, Figure 3 plots the power consumption of `458.sjeng` as a function of $\overline{f}$ and Δ^+; observe that the power surface consists of multiple piecewise planes. Similarly, the contour lines of the measured power surface, shown in Figure 3(b), reveal that the distance between the parallel contour lines is uneven. Again, this observation confirms the piecewise planar nature of the power surface.

These observations further suggest that we might be able to extend our basic model to be piecewise linear. More formally, let $[\phi_-, \phi^+]$ be the interval of all possible frequencies. ϕ_- is the low bound of possible frequencies and ϕ^+ is the up bound of possible frequencies. Consider a k-way partition of this interval into k segments (each segment corresponds to a part of our refined piecewise model) such that

$$\phi_- \equiv \phi_0 < \phi_1 < \phi_2 < \cdots < \phi_{k-1} < \phi_k \equiv \phi^+. \quad (7)$$

Then, a piecewise linear power model can take the following form:

$$P_{\mathrm{MC}}\left(\overline{f}, \Delta^+\right) = \begin{cases} a_{i,0} + a_{i,1} \cdot \overline{f} + a_{i,3} \cdot \Delta^+ \\ \text{when } \phi_{i-1} \le \overline{f} + a \cdot \Delta^+ \le \phi_i, \end{cases} \quad (8)$$

where $a_{i,0}$, $a_{i,1}$, and $a_{i,3}$ are the coefficients of frequency segment $i \in \{1, \ldots, k\}$. For the SPEC2006 benchmarks, we have observed that $k \le 3$ is sufficient to capture any piecewise linear behavior. The coefficient a indicates the line between different pieces when they are projected onto the $\overline{f}$-Δ^+ plane.

In practice, it is not straightforward to determine the exact values of ϕ_i and a in (8). The motivating example in Figure 1 shows that a significant power change occurs when the maximum speed among the cores changes. So, we can replace $\phi_{i-1} \le \overline{f} + a \cdot \Delta^+ \le \phi_i$ with $\phi_{i-1} \le f^+ \le \phi_i$ to simplify the process of determining the values of ϕ_i and a. Experimental results show that this is an effective way to establish the improved piecewise power model.

Table 1: Experimental platform with different microarchitectures.

Processors		Available frequencies (MHz)
Model name	$P \times C$	
AMD Opteron	2×4	2500, 1800, 1300, 800
Xeon Haswell	2×14	2601 (turbo) [2600, 1200] by −100
Xeon Ivy Bridge	2×10	2501 (turbo) [2500, 1200] by −100
Xeon Sandy Bridge	2×8	[2600, 1200] by −100

4. Model Evaluation

We employ 28 benchmarks of SPEC2006 to evaluate the proposed basic model on several different modern multicore processors. The extensive experimental results show that our basic model is accurate for most cases. The refined model can further improve the accuracy of the basic model for some special workloads.

4.1. Experimental Setup

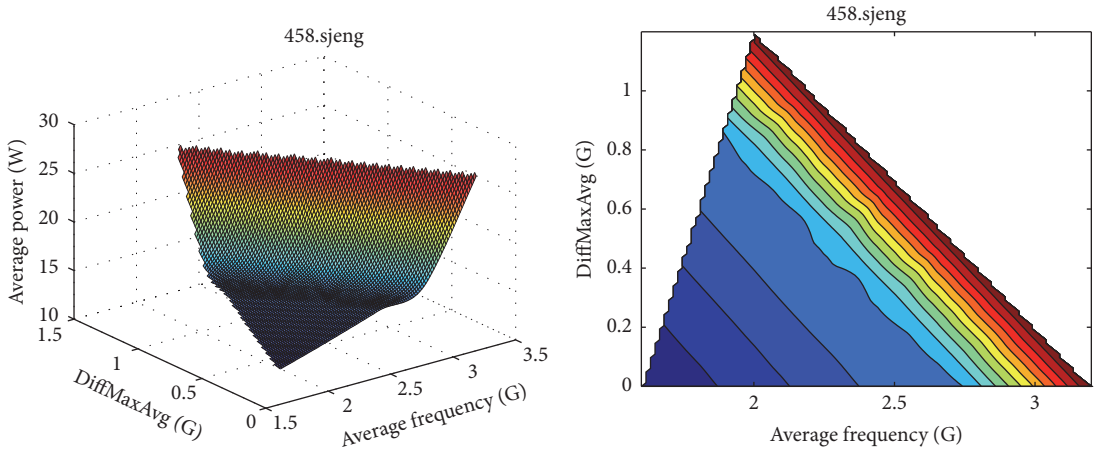
4.1.1. Workloads. We chose the computation-intensive benchmarks from SPEC2006 [13]. SPEC benchmark is used because it represents general-purpose computing. In the future, we would include more different workloads whose power is sensitive to speed. Of the 29 benchmarks in this suite, we omitted `400.perlbench` due to its long execution times. In the experiments, we assigned a benchmark application to run on each core. We considered two assignments: uniform assignment, where the same benchmark is assigned to all cores, and mixed assignment, where different benchmarks are assigned to run on different cores.

4.1.2. Multicore Processors. We carried out our experiments on different generations of Intel x86 microarchitectures and one AMD Opteron architecture. In Table 1, P denotes the number of processors and C denotes the number of cores on each processor.

4.1.3. Speed Scaling and Core Affinity. We used the Linux user-level *cpufreq* interface to set the frequencies of the cores. (To set core i as the frequency of `Fre`, we use the *cpufreq* interface on the following command line: `echo Fre > /sys/devices/system/cpu/cpui/cpufreq/scaling_setspeed`.) We used the Linux command

Table 2: Comparison of different regression models with single benchmark `410.bwaves` as the workload.

Regression model Variables	Reg. func.	#% ≤ 5%	#% ≤ 3%	Max.%	Avg.%	Max. err.	Min. err.	Avg. abs. err.
$\overline{f}, \Delta^+$	$R1$	1.000	0.985	0.042	0.004	0.255	−0.801	0.111
	$R2$	1.000	0.985	0.036	0.004	0.681	−0.356	0.101
	$R3$	1.000	0.980	0.040	0.004	0.758	−0.235	0.103
	$R4$	1.000	0.985	0.036	0.003	0.679	−0.336	0.097
	$R5$	1.000	0.985	0.031	0.003	0.600	−0.308	0.093
$\overline{f}, \Delta^+_-$	$R1$	0.942	0.714	0.078	0.022	1.445	−1.840	0.577
	$R2$	0.942	0.700	0.069	0.022	1.795	−1.500	0.576
	$R3$	0.942	0.714	0.074	0.022	1.802	−1.466	0.575
	$R4$	0.942	0.714	0.067	0.022	1.762	−1.514	0.574
	$R5$	0.557	0.328	0.240	0.054	3.534	−4.547	1.372
$\overline{f}, \Delta_-$	$R1$	0.700	0.485	0.159	0.039	3.172	−3.013	1.009
	$R2$	0.328	0.242	0.316	0.086	5.387	−6.195	2.222
	$R3$	0.700	0.471	0.151	0.039	2.862	−3.169	0.994
	$R4$	0.442	0.285	0.183	0.064	5.108	−5.061	1.692
	$R5$	0.157	0.114	0.369	0.141	8.047	−8.483	3.616
$\overline{f}, \sigma$	$R1$	0.957	0.657	0.080	0.024	1.307	−1.964	0.637
	$R2$	0.928	0.642	0.067	0.024	1.880	−1.407	0.639
	$R3$	0.957	0.671	0.081	0.024	1.948	−1.300	0.637
	$R4$	0.928	0.671	0.066	0.024	1.847	−1.413	0.635
	$R5$	0.728	0.400	0.143	0.042	3.376	−3.271	1.086
$f_1, \ldots, f_N$	$R6$	0.585	0.385	0.191	0.046	3.624	−3.324	1.179
	$R7$	0.585	0.385	0.197	0.046	3.732	−3.236	1.170



(a) The measured power surface is piecewise planar in $\overline{f}$ and Δ^+

(b) The contour lines of the measured power surface in Figure 3(a) are parallel lines, but the distances are not equal

Figure 3: The measured power surface varies linearly with $\overline{f}$ and Δ^+, but with varying slopes.

`taskset` to bind a process to a physical core. (To bind the launched process, `BenchName`, to core i and run it k times, the following command can be used: `taskset -c` i `runspec --config=My.cfg --action onlyrun --size=test --noreportable --iterations=`k `BenchName`.)

4.1.4. Power Measurement. If the multicore systems have power monitoring tools, we will use them directly. For all quad-core Intel processors in Table 1, a clamp ammeter (meter) was equipped to measure the power. For the AMD Opteron processor, the PowerPack tool [17] was installed to get the power. For the platforms that do not provide a power measurement method, such as the machine with dual octacore Sandy Bridge processor and the dual 14-core Haswell processor, we used Intel's Running Average Power Limit (RAPL) interface [15] to obtain the power (PKG Power).

4.2. Model Accuracy. Table 2 shows the results of different candidate models for the benchmark `410.bwaves`, on the quad-core Ivy Bridge platform. Note that we recorded and analyzed a full set of experimental data covering all benchmarks and platforms and the results for other benchmarks show similar trends.

We assess model accuracy using a variety of criteria. In Table 2, "#% $\leq$ 5%" and "#% $\leq$ 3%" refer to the fraction of predictions whose relative error, |model − measured|/measured, is no more than 5% and 3%, respectively. The larger the value is, the more accurate the power model is. "Max.%" and "Avg.%" are the maximum and average values of all relative errors. "Max. err." and "Min. err." mean the maximum and minimum values of all errors. "Avg. abs. err." means the average of the absolute value of residual. The smaller the value is, the more accurate the power model is. According to Table 2, models $R1$ through $R5$ all achieve very high prediction accuracy with variables $\overline{f}$ and Δ^+. But model $R1$ is the simplest one.

Furthermore, the experimental results show that the average relative error of $R1$ is as low as 0.004% and replacing Δ^+ with any other dispersion variable leads to higher prediction errors.

We have also tested the effectiveness of the models using mixed workloads. We generated these mixed workloads using two methods: (i) using four different benchmarks and (ii) using two different benchmarks. (For instance, "two different benchmarks" on a quad-core processor means that one benchmark runs on two cores and another different benchmark runs on the remaining two cores.) The results are similar to those in Table 2. These results suggest that the effectiveness of $R1$ is not just tied to a particular workload. Section 4.3 explores uniform versus mixed workloads in more detail.

The experimental results in Table 2 show that our basic model, $R1$, is accurate. Figure 4 shows the average relative error of the basic model for running the 28 SPEC2006 benchmarks on the seven multicore processors. As can be seen from the figure, except for the Haswell-EP, the basic model achieves very low average relative error (less than 2%) for most benchmarks running on the other six multicore processors, while for Haswell-EP, the average relative error is a little bit high (less than 5%) for most benchmarks.

For the few benchmarks whose average relative errors are greater than 5% (but are all less than 13%), we will employ the refined piecewise model (see Section 3.2) to improve the prediction accuracy.

We compare the prediction accuracy of the basic model and the piecewise model in Table 3. Overall, the piecewise approach improves prediction accuracy. For example, the results of benchmark `458.sjeng` show that the piecewise model reduces the maximum relative error to 0.3% from the original 50.4% of the basic model. They also show that average relative error decreases from 0.094 to 0.001 and the improvement is about 9x on average.

4.3. Uniform versus Mixed Workloads. We consider two benchmarking scenarios: one in which we run the same benchmark on all cores ("uniform" case) and the other in which we run different benchmarks on different cores ("mixed" case).

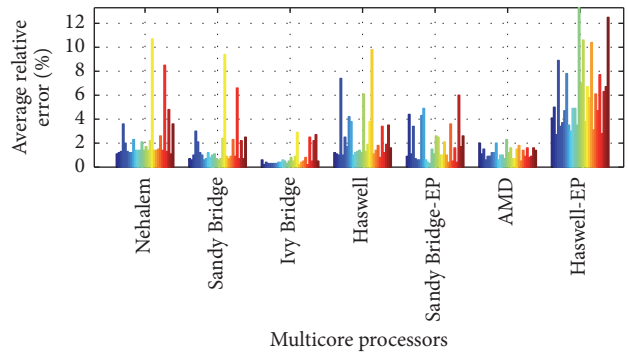


FIGURE 4: The average relative errors of the 28 benchmarks on different multicore processors according to our basic power model.

First, consider the uniform case, for the specific example of the benchmark, `410.bwaves`, running on a quad-core Ivy Bridge processor. The model predictions match very well the actual measurements for various core speeds, as shown in Figure 5(a). In addition, Figure 5(b) shows that the maximum absolute error is less than 0.25 watts and that the maximum relative prediction error is less than 4.2%. Furthermore, more than 98.6% of the predicted values have a relative error within 3%, and the average relative error is less than 0.45%. Though not shown, the test results with 28 SPEC2006 benchmarks show a similar level of model accuracy.

We also find strong linear relationships among power, $\overline{f}$, and Δ^+ in Figures 5(c) and 5(d). Figure 5(c) shows a flat surface (plane) where CPU power increases linearly with $\overline{f}$ and Δ^+. These relationships are easier to see in Figure 5(d), which is a flattened contoured version of the same data; the straight parallel contour lines again reflect linear relationships. These observations essentially confirm that the basic model, $R1$, should be expected to work well.

We also consider mixed workloads, in which each core runs a different application. The model fits under mixed workloads show a similar level of accuracy for $R1$. For example, when running the set of benchmarks, {`410.bwaves`, `433.milc`, `437.leslie3d`, `444.namd`}, one per core on the quad-core Ivy Bridge, the maximum absolute residual is less than 0.31 watts, and the maximum relative error is less than 4.3%. Furthermore, more than 98.5% of the predicted values have a relative error within 3%, and the average relative error is less than 0.5%. Other mixed workloads with two and four different benchmarks exhibit similar degrees of accuracy.

5. Discussion

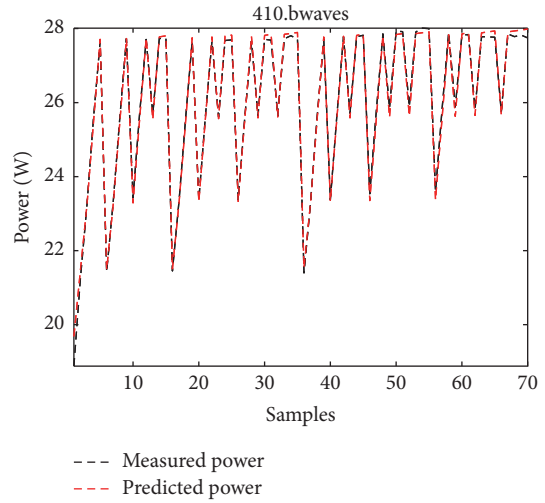
The models proposed in Section 2 raise some natural questions, including *why* the power effect of frequency scaling of a core is dependent on other cores' states and why power models as a linear function of frequency could accurately capture the power effect of frequency scaling empirically.

Table 3: Comparing the errors of the basic and piecewise models for predicting the power of different benchmarks.

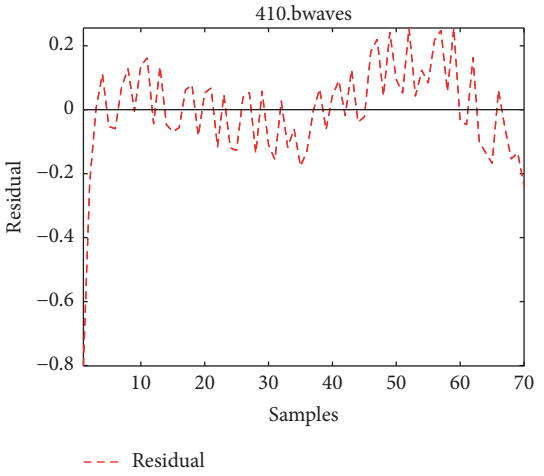
Benchmark	Model	#% ≤ 5%	#% ≤ 3%	Max.%	Avg.%	Max. err.	Min. err.	Avg. abs. err.
401.bzip2	Basic	0.985	0.985	0.055	0.007	0.880	−0.347	0.160
	Piecewise	1.000	1.000	0.013	0.003	0.346	−0.219	0.094
403.gcc	Basic	0.985	0.971	0.158	0.006	0.586	−1.999	0.111
	Piecewise	1.000	1.000	0.010	0.001	0.111	−0.226	0.035
410.bwaves	Basic	0.985	0.985	0.084	0.010	1.620	−0.517	0.308
	Piecewise	1.000	1.000	0.010	0.002	0.269	−0.234	0.078
416.gamess	Basic	0.800	0.685	0.175	0.029	2.819	−2.015	0.716
	Piecewise	0.900	0.842	0.081	0.018	2.056	−1.580	0.460
429.mcf	Basic	0.928	0.800	0.152	0.020	1.160	−2.048	0.379
	Piecewise	1.000	1.000	0.017	0.003	0.346	−0.183	0.082
433.milc	Basic	0.985	0.985	0.095	0.012	1.661	−0.593	0.326
	Piecewise	1.000	1.000	0.011	0.003	0.244	−0.379	0.090
434.zeusmp	Basic	0.985	0.985	0.084	0.010	1.488	−0.459	0.278
	Piecewise	1.000	1.000	0.009	0.001	0.229	−0.224	0.052
435.gromacs	Basic	0.985	0.971	0.159	0.006	0.555	−1.950	0.110
	Piecewise	1.000	1.000	0.010	0.001	0.156	−0.221	0.035
436.cactusADM	Basic	0.985	0.971	0.171	0.006	0.574	−2.122	0.123
	Piecewise	1.000	1.000	0.006	0.002	0.160	−0.137	0.048
437.leslie3d	Basic	0.985	0.985	0.102	0.012	1.884	−0.589	0.354
	Piecewise	1.000	1.000	0.012	0.002	0.253	−0.312	0.069
444.namd	Basic	0.985	0.985	0.068	0.008	1.214	−0.359	0.217
	Piecewise	1.000	1.000	0.008	0.001	0.213	−0.161	0.045
445.gobmk	Basic	0.985	0.985	0.086	0.010	1.690	−0.557	0.321
	Piecewise	1.000	1.000	0.009	0.001	0.274	−0.234	0.062
447.dealII	Basic	1.000	0.928	0.044	0.010	1.269	−0.721	0.415
	Piecewise	1.000	1.000	0.014	0.001	0.501	−0.395	0.074
450.soplex	Basic	0.985	0.985	0.082	0.007	1.563	−0.299	0.189
	Piecewise	1.000	1.000	0.007	0.001	0.192	−0.152	0.051
453.povray	Basic	0.985	0.985	0.120	0.005	0.582	−1.697	0.113
	Piecewise	1.000	1.000	0.005	0.001	0.112	−0.122	0.038
454.calculix	Basic	0.985	0.985	0.050	0.006	0.927	−0.330	0.190
	Piecewise	1.000	1.000	0.009	0.002	0.314	−0.186	0.066
456.hmmer	Basic	0.857	0.800	0.161	0.024	1.269	−2.195	0.458
	Piecewise	0.914	0.914	0.071	0.007	1.368	−1.435	0.165
458.sjeng	Basic	0.642	0.142	0.504	0.094	6.841	−3.331	1.896
	Piecewise	1.000	1.000	0.003	0.001	0.087	−0.080	0.030
459.GemsFDTD	Basic	0.971	0.914	0.131	0.008	1.777	−1.507	0.170
	Piecewise	0.985	0.985	0.064	0.003	1.730	−0.369	0.085
462.libquantum	Basic	0.985	0.985	0.052	0.007	0.973	−0.347	0.199
	Piecewise	1.000	1.000	0.029	0.001	0.754	−0.239	0.054
464.h264ref	Basic	0.985	0.985	0.066	0.008	1.246	−0.422	0.247
	Piecewise	1.000	1.000	0.010	0.001	0.235	−0.374	0.061
465.tonto	Basic	0.900	0.771	0.123	0.023	1.278	−1.977	0.531
	Piecewise	0.971	0.814	0.059	0.015	1.331	−1.225	0.394
470.lbm	Basic	0.985	0.985	0.087	0.009	1.210	−0.310	0.192
	Piecewise	1.000	1.000	0.009	0.001	0.162	−0.173	0.029
471.omnetpp	Basic	0.642	0.385	0.357	0.065	5.816	−3.424	1.652
	Piecewise	0.900	0.757	0.216	0.024	4.616	−1.218	0.611
473.astar	Basic	0.985	0.985	0.050	0.006	0.869	−0.282	0.163
	Piecewise	1.000	1.000	0.008	0.002	0.191	−0.143	0.055
481.wrf	Basic	0.885	0.785	0.156	0.021	1.398	−2.143	0.416
	Piecewise	0.928	0.900	0.071	0.009	1.206	−1.435	0.214

Table 3: Continued.

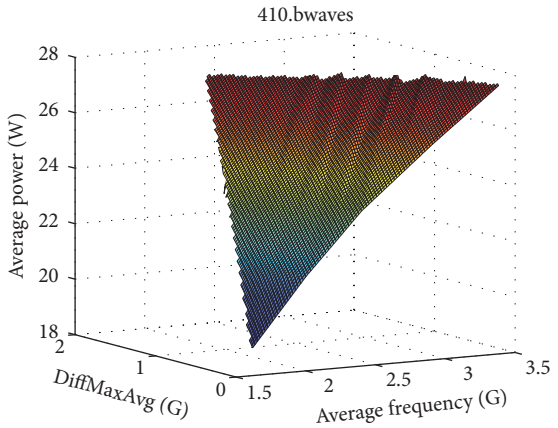
Benchmark	Model	#% ≤ 5%	#% ≤ 3%	Max.%	Avg.%	Max. err.	Min. err.	Avg. abs. err.
482.sphinx3	Basic	0.985	0.985	0.056	0.007	0.786	−0.229	0.141
	Piecewise	1.000	1.000	0.009	0.001	0.165	−0.091	0.030
483.xalancbmk	Basic	0.900	0.628	0.070	0.025	1.577	−1.995	0.744
	Piecewise	0.942	0.742	0.059	0.021	1.420	−1.843	0.651



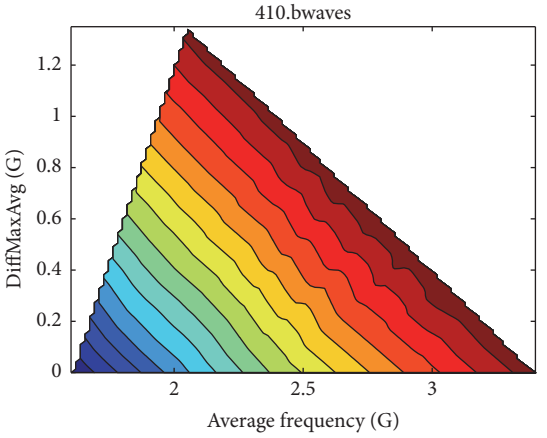
(a) Model prediction versus actual power measurement



(b) The residuals distribution of our power model



(c) A fairly perfect power plane indicating that power linearly increases with $\overline{f}$ and Δ^+



(d) Parallel straight contour lines on the power plane

Figure 5: Model verification with a single benchmark, `410.bwaves`, running on all cores.

5.1. DVFS Interdependency for Multicore Processors. Figure 1 reveals that the same speed scaling from one source frequency to a target may result in different changes for the total processor power. The scaling that updates the maximum frequency among the cores leads to more significant changes for the total power than others. Such differences are explained by two main reasons.

5.1.1. Power of Uncore Devices. The cores on the same processor share uncore devices, which include the last level cache, memory controller, and interconnection links. Uncore device power increases with two main sources. First, when the devices receive more requests from cores, they consume more power to respond [12]. Second, uncore devices on modern processors are equipped with power-aware technologies and can transit among multiple sleep states and performance states [18]. A higher core frequency can trigger the uncore devices to transit from sleep states to active states, or from low performance states to high performance states [18, 19]. Such power state transition leads to a more significant power increase than activity request with the first source.

Uncore device power partly explains the different power effects between the scalings. The scaling that increases the highest speed among the cores not only causes more uncore

Table 4: The power effects of DVFS scaling for different DVFS mechanisms.

	Mech. (i)	Mech. (ii)	Mech. (iii)
Clock	Shared	Individual	Individual
Voltage	Shared	Shared	Individual
DVFS	Dependent	Dependent	Independent
Platforms	Sandy Bridge, Ivy Bridge	Opteron	Haswell

activities *but also* transits uncore devices to higher power states. Consequently, it leads to a larger increase for the whole processor power. In contrast, other scalings only cause uncore activities without updating the uncore performance states and thus increase the uncore device power with a smaller amount.

5.1.2. DVFS on Chip Multiprocessing Cores. The mechanism implementing the DVFS technology is the other reason for the nonuniform power effect of speed scaling on multicore processors. DVFS technology transits the processor cores among different performance states, where a performance state of a core corresponds to a pair of *(frequency, voltage)*. The tuning of the voltages and frequencies for chip multiprocessing cores is implemented by one of the three hardware mechanisms [20–22]: (i) one single shared clock domain and one single shared voltage domain by all the cores, (ii) multiple clock domains and one single shared voltage domain, and (iii) multiple clock domains and multiple voltage domains, or individual per-core DVFS.

Different mechanisms determine the various dependencies between the cores. With mechanism (i), the supplied shared voltage must match the highest frequency among the cores in order for DVFS to work properly. Consequently, if a scaling updates the maximum frequency among the cores, it causes large processor power jump/drop due to the tuned up/down frequency and voltage; other scalings merely change processor power. Mechanism (ii) has a finer power control than mechanism (i) as each core can individually scale its frequency. Mechanism (ii) is effectively Dynamic Frequency Scaling (DFS). Mechanism (iii) deploys individual clock and voltage domain for each of the cores and independently controls per-core frequency and voltage. Table 4 summarizes the interdependencies of power effects of DVFS scaling for these three mechanisms. Note that only mechanism (iii) supports per-core DVFS.

Technology has been shifting from mechanism (i) to mechanism (iii) [20–22]. Mechanism (i) has been mostly adopted by earlier generations of Intel processors such as Xeon Nehalem and SandyBridge architectures to limit the platform and packaging cost. To improve the granularity of DVFS control, AMD processors, as shown in Figure 1, explore mechanism (ii) to change frequencies of individual cores. More recently, per-core DVFS using mechanism (iii) [21, 23] is available on Intel Haswell processors to improve DVFS effectiveness for multithreaded workloads with heterogeneous behavior.

The challenge that users face in designing DVFS scheduling is that, no matter whether the underlying architectures support per-core DVFS or not, operating systems and kernels including *cpufreq* and the Intel *P-State* driver give users an impression that they do. Such discrepancy between user perception and the actual hardware ability can lead to poor DVFS scheduling decisions and adverse application performance degradation. To make better DVFS scheduling decisions, users must first identify the architectural DVFS mechanism and carefully select a proper model specific to the mechanism. Our models resolve this issue as they are applicable to all types of DVFS mechanisms for all generations of modern processors, relieving users from the burden of characterizing the underlying architectural and DVFS mechanisms.

5.2. Cubic Power Model versus Linear Power Model. It has been widely accepted that the dynamic power is a cubic function of frequency for DVFS-capable processors [1–4]; that is,

$$P \propto f^3. \tag{9}$$

This cubic function is derived from two relations. First, the dynamic power of CMOS devices is a function of frequency and transistor's supply voltage [24].

$$P = A \times C \times V^2 \times f, \tag{10}$$

where C is the capacitance being switched per clock cycle, V is the transistor's supply voltage, A is the activity factor indicating the average number of switching events undergone by the transistors in the chip, and f is the frequency.

Second, frequency f depends on supply voltage V in the following relation:

$$f \propto (V - V_{\text{th}})^{\kappa} / V. \tag{11}$$

Here, V_{th} is threshold voltage and κ is a technology-dependent constant accounting for velocity saturation. For 1000 nm technology and older, κ's value could be 2 [25, 26] and supply voltage is much larger than threshold voltage [27]. Consequently, frequency is considered to be proportional to supply voltage and power is considered proportional to the cubic function of frequency.

The power proportional to f^3 relation becomes inaccurate due to technology evolution in two aspects. First, to effectively reduce dynamic power consumption, supply voltage has been reduced over the years and is now only slightly larger than threshold voltage V_{th} [27–29]. Resultantly, supply voltage for DVFS processors has a small range, and its scaling in this range leads to smaller variation for dynamic power. Second, κ reduces over the generations of technology. It is approximately 1.3 in 45 nm technology and could be even smaller in newer generations. Consequently, reducing the voltage by a small percentage will reduce the operating frequency by a larger percentage [29]. Thus, the power effect of voltage scaling is overshadowed by the power effect of frequency scaling, and power is effectively governed by frequency scaling as a linear function, as captured by our models.

6. Related Work

As power becomes a critical constraint at all levels of HPC systems from chip, node to data center, extensive research has been conducted to measure, model, and manage power on computer components and systems. In this section, we briefly present related work in power measurement and architecture-level power modeling and also discuss most closely related work in system-level power modeling.

Direct power measurement is a fundamental approach to quantitative power evaluation and provides an ultimate reference for analytical power modeling [30]. Limited by the availability of power measurement tools, earlier work usually instruments external meters to computer circuits to measure the power consumption of individual components and further the entire system. For example, PowerPack [17] is built with NI data acquisition devices, which are instrumented into the DC power lines to measure the power of computer components including CPU and memory. Similarly, PowerInsight [31] and PowerMon [32] deliver the same functions with self-made pluggable cards in smaller forms. More recently, to meet the increasing demand for power monitoring and measurement, commodity processors including those of Intel and AMD have begun to provide embedded power meters and interfaces [15, 33, 34]. Such embedded meters provide accurate power measurements that are greatly helpful to system and software designers. Nevertheless, direct power measurement is limited to physical devices and components. They cannot separate the power of individual cores on multicore processors to support power management with thread concurrency scaling, which is effective and most needed for future architectures.

Analytical modeling, in contrast to physical measurement, can be performed on both hardware and software units with different granularity. Microarchitecture-level power models are commonly used to investigate and evaluate new power-saving and power-aware hardware and architectures. Such models correlate power to parameters and usage of architectural components including register files, function units, clock, and caches [35–37]. Representative models include Wattch [35] for single-core architectures and McPat [37] for chip multiprocessors. Models with such great details are complex and limited to HPC components and building blocks.

System-level power modeling, which is the research class that our work falls into, is an essential approach for runtime frequency schedulers to achieve power reduction and energy saving on HPC systems. Most previous studies investigate single-core architectures and systems and can be grouped into two basic categories [1, 2, 10, 38]. Models in the first category [1, 2] describe power as a basic polynomial function of CPU frequency in the form of (1). The polynomial degree varies with power-aware technologies and is set to 3 for DVFS-capable processors and otherwise greater than or equal to 1 [1]. Models in the other category [10, 38–41] build correlation between hardware performance events with power and leverage performance monitoring counters available on hardware to collect hardware event data. In general, the techniques in this category require extensive profiling and large volumes of experimental data for model training.

As multicore processors become the building blocks of HPC systems, researchers attempt to understand their power consumption. A widely adopted approach assumes that the cores are independent and the total power consumption of a multicore processor is the sum over the power of individual cores, each of which is estimated by the traditional system-level power models for single cores [5–8, 42]. Nevertheless, as our work and Basmadjian and de Meer's [9] show, simply extending single-core power models without capturing the core interdependency results in inaccurate power estimation.

Little work has been done to capture the heterogeneous power effect from cores interdependency in multicore processor and all requires microarchitectural decomposition and event accounting. Basmadjian and de Meer [9] decomposed a processor to its architectural components including on-chip cores, off-chip caches, and interconnections and modeled the power of each component with the power model in (1). Specifically, in their work, the off-chip caches and interconnections capture the power interdependency between cores. Bertran et al. [10] decomposed a processor further in finer granularity to function units and front end and derived the power of each component with its measurable performance events with performance monitoring counters. This work reflects the power effect of core interdependency by using adjusted model coefficients for single-core processors. Shen et al. [12] similarly used measurable hardware performance events on microarchitectural components to estimate power. Particularly, they paid special attention to chip maintenance power and shared it evenly between active cores.

Our models are distinct from prior efforts in system-level multicore power modeling. Our models are accurate by capturing the interdependency between cores on multicore processors, yet practical and easy to use by only using average frequency and frequency dispersion as model variables. In contrast, existing simple models such as (2) may provide inaccurate power estimations and lead to wrong scheduling decisions, while detailed models such as [9, 12] are not scalable to future architectures that contain a large number of cores. Simple and easy-to-use power models are critical for power optimization and management for future applications and system software [43]. We believe that our models provide a viable solution and can promote the research in energy optimization for traditional and emerging software.

7. Conclusions and Future Work

This work shows that simply extending the traditional single-core power model might not faithfully capture the real power behavior of modern multicore processors. The reason is that the traditional model assumes that individual cores contribute to power consumption *independently*. We show that this assumption is not true. Our proposed alternative uses aggregate statistical measures, *mean frequency* and *dispersion*, to express the *interaction* among cores. Compared to

the existing approaches that explicitly investigate the shared resources among cores and use microarchitectural events to capture heterogeneous power effects of individual core speed scaling, our models are much simpler and scalable to emerging and future multicore technologies. Our experiments validate the effectiveness of the proposed model and show its accuracy.

From our work, we draw several additional high-level conclusions. First, the power consumption of a multicore processor can be accurately predicted by a simple linear model of the average core speed and the speed variation. The linear model indicates that, besides the average speed, greater speed variation can cause more power consumption.

Second, using our method, one can build the power model that is suitable for an underlying multicore processor without needing to know many hardware details.

Third, our power models can be used to analyze and quantify the power characteristics inherent in the applications and the hardware architectures. For new multicore processors, one only needs to run the experiments according to the methodology presented in this paper to determine the best model and value of its parameters from the experimental data. The modeling method proposed in this work requires running an application on the target processor a small number of times.

Looking forward, evaluating not only the core but also the uncore hardware effects (such as cache noise) may further improve the model. To further reduce the number of runs needed to derive the model parameters, future work might combine the modeling approach proposed in this paper with the general modeling approach developed in our prior work [44, 45], thereby yielding power models that are both accurate and generic.

Disclosure

Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to thank Intel in Beijing for providing the Haswell-EP platform for experiments. This research is supported in part by the National Key Research and Development Program of China (nos. 2016YFB1000602 and 2017YFB0701501), National Natural Science Foundation of China (nos. 61440057, 61272087, 61363019, 61073008, and 11690023), and MOE Research Center for Online Education Foundation (no. 2016ZD302). Parts of this work are also supported by the U.S. National Science Foundation (NSF) (Awards nos. 1339745, 1422935, and 1551511) and CAREER (Award no. 0953100).

References

[1] N. Bansal, T. Kimbrel, and K. Pruhs, "Speed scaling to manage energy and temperature," *Journal of the ACM*, vol. 54, no. 1, article 3, 2007.

[2] F. Yao, A. Demers, and S. Shenker, "Scheduling model for reduced CPU energy," in *Proceedings of the 36th IEEE Annual Symposium on Foundations of Computer Science*, pp. 374–382, IEEE, October 1995.

[3] T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," in *Proceedings of the 28th Hawaii International Conference on System Sciences*, vol. 1, pp. 288–297, January 1995.

[4] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in *Proceedings of the IEEE Symposium on Low Power Electronics*, pp. 8–11, October 1994.

[5] S. Cho and R. G. Melhem, "Corollaries to Amdahl's law for energy," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 25–28, 2008.

[6] M. Ghasemazar, H. Goudarzi, and M. Pedram, "Robust optimization of a chip multiprocessor's performance under power and thermal constraints," in *Proceedings of the IEEE 30th International Conference on Computer Design (ICCD '12)*, pp. 108–114, IEEE, Washington, DC, USA, October 2012.

[7] K. Meng, R. Joseph, R. P. Dick, and L. Shang, "Multi-optimization power management for chip multiprocessors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 177–186, ACM, Ontario, Canada, October 2008.

[8] L. Yu, F. Teng, and F. Magoulès, "Node scaling analysis for power-aware real-time tasks scheduling," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2510–2521, 2016.

[9] R. Basmadjian and H. de Meer, "Evaluating and modeling power consumption of multi-core processors," in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, Madrid, Spain, May 2012.

[10] R. Bertran, M. Gonzelez, X. Martorell, N. Navarro, and E. Ayguade, "A systematic methodology to generate decomposable and responsive power models for CMPs," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1289–1302, 2013.

[11] J. C. McCullough, Y. Agarwal, J. Chandrashekar et al., "Evaluating the effectiveness of model-based power characterization," in *Proceedings of the USENIX Annual Technical Conference*, vol. 20, 2011.

[12] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power containers: An OS facility for fine-grained power and energy management on multicore servers," *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 65–76, 2013.

[13] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[14] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th ACM International Conference on Supercomputing (ICS '13)*, pp. 173–182, ACM, Eugene, Ore, USA, June 2013.

[15] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE International Symposium on Low-Power Electronics and Design*, pp. 189–194, IEEE, August 2010.

[16] J. Triebig, "Likwid: Linux tools to support programmers in developing high performance multi-threaded programs," 2012, http://code.google.com/p/likwid.

[17] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, 2010.

[18] V. Gupta, P. Brett, D. Koufaty et al., "The Forgotten "Uncore": on the energy-efficiency of heterogeneous cores," in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '12)*, pp. 367–372, 2012.

[19] H.-Y. Cheng, J. Zhan, J. Zhao, Y. Xie, J. Sampson, and M. J. Irwin, "Core vs. uncore: The heart of darkness," in *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC '15)*, pp. 1–5, IEEE, June 2015.

[20] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, "EnergySmart: Toward energy-efficient manycores for Near-Threshold Computing," in *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture*, pp. 542–553, IEEE, February 2013.

[21] E. Rotem, R. Ginosar, A. Mendelson, and U. Weiser, "Multiple clock and voltage domains for chip multi processors," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 459–468, ACM, December 2009.

[22] A. A. Sinkar, H. Wang, and N. S. Kim, "Workload-aware voltage regulator optimization for power efficient multi-core processors," in *Proceedings of the 15th Design, Automation and Test in Europe Conference and Exhibition*, pp. 1134–1137, IEEE, March 2012.

[23] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture*, pp. 123–134, IEEE, February 2008.

[24] T. Mudge, "Power: a first-class architectural design constraint," *The Computer Journal*, vol. 34, no. 4, pp. 52–58, 2001.

[25] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power CMOS," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, 1997.

[26] J. Burr and A. Peterson, "Ultra low power cmos technology," in *Proceedings of the 3rd NASA Symposium on VLSI Design*, vol. 1, 1991.

[27] H. Iwai, "Roadmap for 22 nm and beyond," *Microelectronic Engineering*, vol. 86, no. 7, pp. 1520–1528, 2009.

[28] Intel Hewlett-Packard, "Microsoft, phoenix, and toshiba. Advanced configuration and power interface specification," 2004.

[29] N. S. Kim, T. Austin, D. Blaauw et al., "Leakage current: Moore's law meets static power," *The Computer Journal*, vol. 36, no. 12, pp. 68–75, 2003.

[30] H. Esmaeilzadeh, T. Cao, Y. Xi, S. M. Blackburn, and K. S. McKinley, "Looking back on the language and hardware revolutions: measured power, performance, and scaling," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 319–332, 2011.

[31] J. H. Laros, P. Pokorny, and D. Debonis, "PowerInsight—a commodity power measurement capability," in *Proceedings of the International Green Computing Conference (IGCC '13)*, pp. 1–6, IEEE, June 2013.

[32] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "PowerMon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proceedings of the IEEE SoutheastCon 2010 Conference: Energizing Our Future*, pp. 479–484, IEEE, Concord, NC, USA, March 2010 (Chinese).

[33] J. Demmel and A. Gearhart, "Instrumenting linear algebra energy consumption via on-chip energy counters," Tech. Rep. UCB/EECS-2012-168, University of California, Berkeley, Calif, USA, 2012.

[34] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the intel microarchitecture code-named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.

[35] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *ACM*, vol. 28, no. 2, pp. 83–94, 2000.

[36] P. Landman, "High-level power estimation," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 29–35, IEEE, August 1996.

[37] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480, IEEE, December 2009.

[38] W. L. Bircher and L. K. John, "Complete system power estimation using processor performance events," *IEEE Transactions on Computers*, vol. 61, no. 4, pp. 563–577, 2012.

[39] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 135–140, ACM, 2001.

[40] C. Möbius, W. Dargie, and A. Schill, "Power consumption estimation models for processors, virtual machines, and servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1600–1614, 2014.

[41] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, p. 46, 2009.

[42] S. Albers, F. Müller, and S. Schmelzer, "Speed scaling on parallel processors," *Algorithmica*, vol. 68, no. 2, pp. 404–425, 2014.

[43] H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley, "Looking back and looking forward: Power, performance, and upheaval," *Communications of the ACM*, vol. 55, no. 7, pp. 105–114, 2012.

[44] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing*, pp. 661–672, Boston, Mass, USA, May 2013, https://smartech.gatech.edu/xmlui/handle/1853/45737.

[45] K. Czechowski and R. Vuduc, "A theoretical framework for algorithm-architecture co-design," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2013*, pp. 791–802, Boston, Mass, USA, May 2013.