Sustainable Computing: Informatics and Systems 4 (2014) 83–93

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

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

Myths in power estimation with Performance Monitoring Counters

Jason Mair*, David Eyers, Zhiyi Huang, Haibo Zhang

Department of Computer Science, University of Otago, PO Box 56, Dunedin 9054, New Zealand

ARTICLE INFO

Article history:
Received 16 September 2013
Received in revised form 18 February 2014
Accepted 21 March 2014

Keywords:
Power estimation
Performance Monitoring Counter (PMC)
Power metering model

ABSTRACT

Many techniques have previously been proposed for using low-level CPU Performance Monitoring Counters in power estimation models. In this paper, we present some apparent myths regarding these techniques, and their potential impact. The underlying misconceptions include: (1) sampling rate and execution time can be left unspecified; (2) thermal effects are insubstantial; (3) memory events correlate well with power consumption; (4) compilation configuration does not need to be reported; and (5) metrics for performance evaluation of models are comparable. We aim to raise the awareness of these interesting issues, which existing power modeling techniques often do not address. Our discussions provide some guidance to avoid these myths and their effects through detailed specification of software and hardware configurations.

1. Introduction

Much previous research has focused on building accurate power estimation models based upon Performance Monitoring Counters (PMCs) [1–6]. These models collect relevant PMC values and map them to power usage based on the sampled $\langle pmc_1, pmc_2, \ldots, pmc_n, P\rangle$ tuples, where P is the measured power under the corresponding PMC values. Usually the PMC values reflect the activities of the system components such as CPU and RAM, which can potentially correlate well with the power usage of the system. These techniques allow runtime power estimation, on a per-application basis, without requiring the use of a power meter or special hardware. They are very useful in power metering of virtual machines in cloud computing [7,8]. Even hardware-based power metering in Intel's Sandy Bridge microarchitecture has to use PMC-based estimation to measure the dynamic power for the processor package [9,10]. However, the research into the use of these techniques needs to be coupled with an improvement in the specification of the hardware and software configurations used in the process of the researchers' power modeling. In previous research, these configuration details were normally given a brief mention, but lacked discussion and analysis on their potential impacts on the resulting power model.

Our experiments show that many of these configuration details are crucial in order to achieve reproducible results. The omissions that we have encountered can often be attributed to reasons such as assumption of background knowledge, limited space for publication, or indifference to the authors interpretation of the experimental results.

In this paper, we discuss the impact of hardware and software configurations and the possible myths in the research area of PMC-based power estimation due to the lack of detailed information. More precisely, we have selected the following five myths to discuss.

- *Sampling rate and execution time can be left unspecified*—In previous research, the sampling rate for taking PMC-power samples is seldom mentioned. Very often, when a benchmark is run, the PMC counters are recorded for the whole execution period of the benchmark and the average power of the benchmark during the monitoring period is used in the sample. However, we will show that the sampling rate can affect the accuracy of the power estimation significantly.
- *Thermal effects are insubstantial*—Thermal effects are often not discussed in power estimation. Many experimental results were given without mentioning the temperature condition of the CPU chips. We will show that the temperature of the CPU chips has an effect on the accuracy of the power model and its effect can be eliminated with proper treatment.
- *Memory events correlate well with power consumption*—Previous research often assumes a correlation between memory events and power consumption. The idea is that cache misses should correlate with energy-expensive memory fetches. We will show

* Corresponding author. Tel.: +64 3 479 8498; fax: +64 3 479 8529.
E-mail addresses: jkmair@cs.otago.ac.nz (J. Mair), dme@cs.otago.ac.nz (D. Eyers), hzy@cs.otago.ac.nz (Z. Huang), haibo@cs.otago.ac.nz (H. Zhang).

http://dx.doi.org/10.1016/j.suscom.2014.03.007
2210-5379/

that this is not the case on our multicore system, which is most likely due to its different memory architecture.

- *Compilation configuration does not need to be reported*—The selection of compiler optimization is often given very little consideration as it is perceived to have a minimal impact, if any, on the accuracy of the power model. We will show that power models can be more compilation dependent than is typically expected.
- *Metrics for performance evaluation of models are comparable*—The results for many existing models are not presented in a manner allowing for them to be fairly compared to alternative models due to the use of different metrics. Additionally, some of the evaluation methods used may result in misleading conclusions. We will show how to avoid such problems, in order to present results in a clear, informative way.

These observations give us inspiration that detailed specification of software and hardware configurations is very important for fair comparison and a deep understanding of different PMC-based power models.

The intention of this paper is not to criticize any of the existing work, or to propose any detailed power estimation methodologies. Instead, our objectives are twofold. First, we would like to raise awareness of the potential pitfalls in the research area. Second, we want to stress the importance of detailed specification of software and hardware configurations in the process of power modeling. If experimental results are published with sufficient context and specification, it helps avoid the myths in power modeling that we will discuss in the following sections.

The remainder of this paper is organized as follows. Section 2 describes our experimental setup and software/hardware configuration. The myths are discussed in Section 3. A general modeling methodology is presented in Section 4. Related work is summarized in Section 5, with the conclusions and future work in Section 6.

2. System configuration

In this section and throughout this paper, we make a conscious effort to specify all relevant configuration information, allowing experimental results to be placed in the appropriate context. It will be shown through this paper that changes in sampling rate, execution time and configuration of benchmarks impact the resulting model. If such information was not explicitly specified, as is often the case in many published papers, it becomes harder to compare fairly among alternative approaches and power models.

Care should be taken to ensure that any neglected details do not impact on the resulting power model or create the potential for erroneous conclusions. Often researchers neglect the detailed specification of system parameters due to limited publication space. Unfortunately, a large number of parameters can have a significant effect on the experimental results. The architecture-specific nature of PMC-based power estimation means it is important to document all relevant hardware and software configurations.

It is for these reasons that this section takes the time to describe the overall experimental setup of hardware and software configuration, including the design of our own micro-benchmark. Certain experiment-specific configuration details are left until the corresponding results sections.

2.1. Experimental setup

The experiments are run on a Dell PowerEdge R905 with four quad-core AMD Opteron 8380 processors (CPUs), with each core having its own floating-point unit (FPU). Each processor is located with 4GiB of memory organized in a NUMA (Non-Uniform Memory Accesses) architecture, providing a total of 16GiB RAM. The processor provides four alternate operating frequencies through DVFS (Dynamic Voltage and Frequency Scaling), however we restrict the frequency to the highest (2.5 GHz) for our experiments as it is the most commonly used frequency. All micro-benchmarks are compiled using gcc 4.6.3 with no optimization enabled, ensuring none of the micro-benchmark operations are optimized away. The NAS Parallel Benchmarks are compiled with gcc 4.6.3, using OpenMP 3.0 [11] and optimization argument `-O3` as the default in our experiments. Some specific experiments use gcc 4.4.7 or optimization level `-O0`, which are explicitly stated. All benchmarks were run on a standard installation of Linux version 2.6.32-25.

Table 1
Types of workload.

Micro-benchmark	Description
FPU	Floating point multiplication
INT	Integer multiplication and division. Represents most micro operations
Memory	Wide-ranging random memory accesses
NOP	Idle loop with NOPs
Cache	Memory accesses with high cache-hit ratio

The power is measured with the Watts Up? PRO.net power meter connected via USB to an external monitoring system. The accuracy of the power meter is ±1.5% +0.3 W [12]. An iSocket (InSnergy Socket)[1] power meter was additionally used to validate power measurements, which has an accuracy of 1%. The idle operating state of the system was measured as 249 W, providing the base/idle power (i.e., static power). The default operating state of the OS is idle, as no sleep or halt states are available on the experimental server.

The external monitoring system was additionally configured to remotely monitor the server's system components and temperatures, while recording the power. This was achieved by altering an instance of IPMItool [13] to log specific measurements at the same rate as the power values. Communication is handled by the Intelligent Platform Management Interface (IPMI) over LAN, bypassing the OS, directly communicating with the Baseboard Management Controller (BMC). Thus, this measurement causes no direct overhead on the test server.

All performance values are collected from the PMCs, which are a set of four, per-core hardware registers [14]. Each register is set to record one of the 120 available performance events at the start of each execution.

2.2. Benchmark configuration

The myths discussed in this paper come from observations made during our work on deriving a runtime power estimation model based on PMC values. The idea behind such a model is to find the relationships between key PMCs, the workload type they represent, and the resulting power use. For example, if the PMC for FPU utilization is high, the current workload is FPU intensive, causing a large power draw. Alternatively, if the number of cache misses is high, there may be many memory accesses, resulting in lower utilization of the processor and a lower power draw from the processor.

To explore these basic principles, we created a micro-benchmark designed to reproduce five key workload types that could be expected in general purpose software, shown in Table 1. The micro-benchmark, as shown in Program 1, consists of a larger outer loop, intended to perform a large number of iterations, and a small `for` loop for short bursts of each workload type. The inner

[1] Institute for Information Industry, http://web.iii.org.tw/, who we thank for providing this measurement equipment.

`for` loops execute quickly, ensuring that whenever the PMCs are sampled during execution, a mix of workload types will be represented. Modest variations in this burst time are provided by the pseudo random numbers. This is then multiplied by a ratio, which is designed to allow extra weight to be added to a single workload type during execution.

Program 1: Pseudo-code of the micro-benchmark.

```
for large_number_of_iterations do
    for pseudo_random_number_n × fpu_ratio do
        fpu_microbenchmark();
    end for
    for pseudo_random_number_n+1 × int_ratio do
        int_microbenchmark();
    end for
    for pseudo_random_number_n+2 × memory_ratio do
        memory_microbenchmark();
    end for
    for pseudo_random_number_n+3 × nop_ratio do
        nop_microbenchmark();
    end for
    for pseudo_random_number_n+4 × cache_ratio do
        cache_microbenchmark();
    end for
end for
```

By default, all ratios are set to one. However, if we are interested in the impact of a more FPU-oriented workload, for example, then we increase the FPU ratio, leaving all other ratios the same. The ratios shown throughout this work are 1, 2, 4, 6 and 8. To avoid any synchronization overheads, 16 independent instances are run concurrently on each of the 16 cores.

3. Myths

Many of the existing methodologies for PMC-based power estimation [1–6] to some degree use a black-box approach to processing PMC data, where the regression function is directly applied to the data without necessarily being visualized. As a result, a detailed specification of the modeling procedure may not be available, hindering other researchers from reproducing the results. In contrast, this section describes some of the observations made during our thorough analysis and visualization of the collected experimental data in the process of deriving our power estimation model. We will also disclose the differences of the resulting model due to the system architecture used and the selection of statistical methods.

In this section, each subsection introduces a myth that could be observed in published literature. Each myth is followed by a series of observations derived from our experiments and our approach to avoid key misconceptions underlying the myth. The first four subsections conclude with a brief discussion of how to address the myth within a modeling process. The final subsection covers statistical methods, and advice on avoiding these problems are threaded throughout it.

3.1. Myth 1: Sampling rate and execution time can be left unspecified

The rate at which PMCs and power samples are taken can have a direct impact on the strength of correlation and the noise within a dataset. This is intuitive, as it will determine the aggregation of results, but is often not documented.

3.1.1. Observation: execution time and sampling rate

The first step in deriving a power model will consist of collecting some initial coarse-grained data for a range of PMCs, sampled once

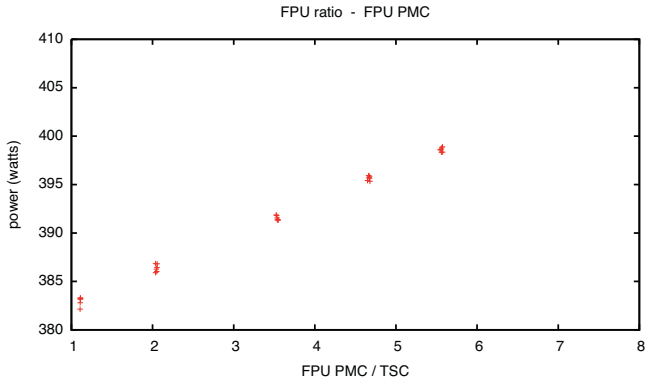


Fig. 1. Intensity of FPU correlated with power measured over entire execution of micro-benchmark running an FPU workload.

when execution begins and again upon completion of each micro-benchmark iteration. The TSC (Time Stamp Counter) is additionally read whenever PMCs are read in order to provide a measure of time. TSC measures the time as the number of cycles since reset, allowing the intensity of the PMC-related event to be calculated during execution.

For example, Fig. 1 shows some intensity values for the FPU activities for multiple iterations of the micro-benchmark, introduced in Section 2.2. The *x*-axis is the intensity of FPU activities, which is calculated by taking the difference in the two FPU_{PMC} measurements of the execution period and dividing it by the difference in the corresponding *TSC* values, which provides the intensity of the activities during the execution period. The *y*-axis is the calculated average power use, measured by the power meter, over the same execution period. In this example, the FPU ratio is adjusted for different iterations, providing the spread of data clusters along the *x* and *y*-axis.

In Fig. 1, the data points form a series of tight clusters along a linear path, which is what was expected according to previous research.

Now knowing that the power/PMC correlation works at a basic level, the logical next step was to increase the rate at which samples are taken. This was chosen to be once every second, in order to match the available data from the power meter. Each PMC value and power measurement are logged to a file during execution, with a corresponding timestamp, to allow synchronization and post-processing. This time, the results shown in Fig. 2 were not what was expected. The modest horizontal spread of points within each cluster is due to the adjustments in the pseudo random number in the algorithm. However, the vertical stripping was not expected

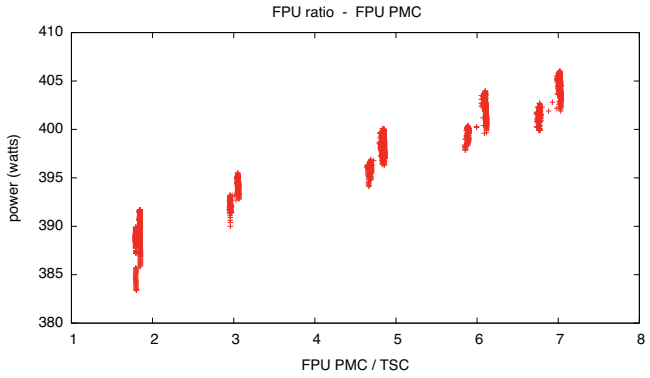


Fig. 2. Intensity of FPU correlated with power, sampled every second for ≈3 min execution of the micro-benchmark running FPU workload. Vertical stripes are a result of a warm-up effect.

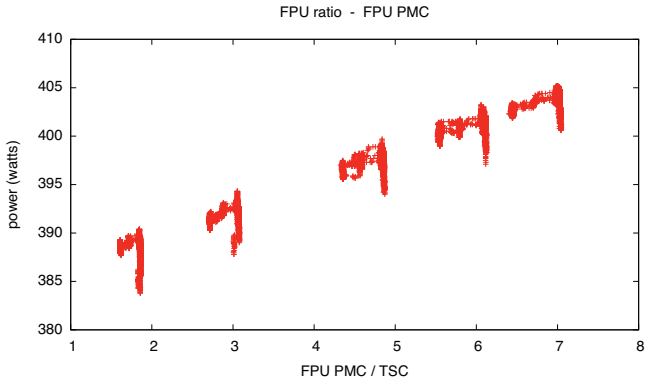


Fig. 3. Intensity of FPU correlated with power, sampled every second for ≈25 min execution of the micro-benchmark running FPU workload. The linear trend is more clear though vertical stripes still exist due to a warm-up effect.

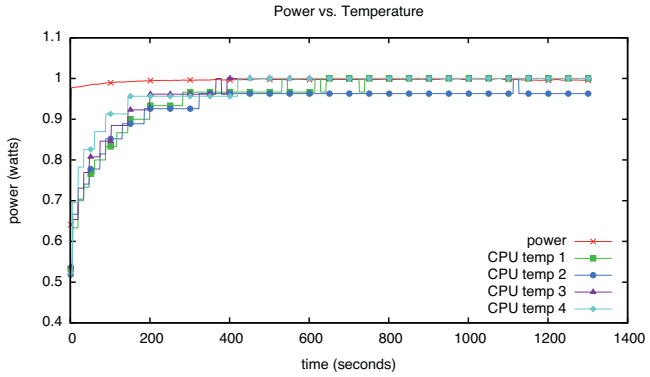


Fig. 4. Normalized CPU temperatures and power meter readings for a long-run execution of the micro-benchmark running FPU workload.

at all. This indicates that something within the system is causing the power values to vary during execution, which was previously obscured by the coarse-grained samples.

Since the micro-benchmark consists of a fairly consistent workload, it was suspected that the stripping was due to an inherent latency in the system responding to starting execution. If such a latency exists, the trend in the data should become more linear with increased execution times. Therefore, each configuration was re-run, increasing the execution time from ≈3 min to ≈25 min. The results in Fig. 3 show that when sampled over a longer period with the same sampling rate, data points return to lying on a linear path, though there is a long vertical tail in each cluster, which will be explained in the next section. The spread of points for each cluster along the *x*-axis is a result of the increased range of pseudo random times made available by the significantly increased iteration count.

In this section we have seen two ways in which important data may be obscured. First, the extreme case of using a sampling rate which is too coarse-grained for the benchmark may omit important samples for training. Second, execution times that are too short may hide longer-term trends within a dataset.

3.1.2. Observation: benchmark configuration and execution time

The previous observation not only illustrated the importance of choosing an appropriate rate at which to sample a benchmark, but also the importance of ensuring a sufficiently long execution time. However, the execution time is dependent on the benchmark configuration, requiring the publication of configuration details for an experiment to be reproducible. Unfortunately, much existing work only goes as far as publishing the execution time, if at all. The problem arises when benchmarks have multiple, configurable execution parameters, such as the problem size or iteration count, where multiple configurations will have varied execution characteristics, but are capable of achieving similar execution times.

To test this hypothesis, we ran an OpenMP instance of the fast Fourier Transform (FT) benchmark from the NAS Parallel Benchmark (NPB) suite in two different configurations, achieving similar execution times. The first configuration had a problem size of $512 \times 512 \times 512$, completing 250 iterations. The mean measured power was 403.6 W with a large standard deviation, 56.19 W, caused by periods of low power use. Alternatively, the second configuration was run with a larger problem size of $1024 \times 512 \times 512$, completing only 100 iterations. The mean power of 425.0 W was much closer to the peak power, with a smaller standard deviation, 36.2 W.

This observation illustrates how a similar execution time can be achieved by two different benchmark configurations, with each having quite different power characteristics. This further highlights the need for a full disclosure of experimental configuration details in the literature, preventing erroneous conclusions from being drawn.

3.1.3. Discussion

When selecting the rate at which to sample the PMC counters, it is important to consider potential smoothing effects if samples are too coarse-grained. Not doing so will obscure data trends and characteristics for the sampled benchmark. Similar to this is the importance of execution time, as this additionally impacts the number of samples for a given rate, potentially further obscuring trends from longer runs.

Documenting both the sampling rate and the execution time help to add context to the commonly reported statistics such as the mean power. The significance of such values can be questionable and statistically inaccurate if the data set is believed to be too small.

3.2. Myth 2: Thermal effects are insubstantial

Thermal effects are often not discussed in power modeling. For those who are aware of the thermal effects on power consumption, it is commonly perceived that the thermal effects can be negated by locking the fan speed, believing the change of fan speed is the main cause of the variation in power due to changes in thermal load. An alternative technique to locking the fan speed is the use of a CPU warm-up phase before the start of each execution. However, we find neither of these can sufficiently negate the thermal effects.

3.2.1. Observation: fan speed and power

Section 3.1 mentioned the presence of a latency in power changes within long runs corresponding to the beginning of each micro-benchmark execution, which causes the long vertical tail of each cluster in Fig. 3. The most likely cause of a delayed effect on power within a system is temperature-related. To explore this, we used IPMI to monitor CPU temperatures once a second during execution of each micro-benchmark. Fig. 4 shows the normalized values for power and temperature of all four CPUs on the *y*-axis. The *x*-axis gives the time in seconds. The power curve steadily increases until around 400 s where it flattens out. The recorded temperatures follow a similar trend where they continue to increase until about the same time, reaching a stable point with the exception of an occasional temperature spike.

These results illustrate a trend between CPU warm-up and the corresponding power latency. The most surprising aspect of this is the length of time required to reach a stable value: well over six

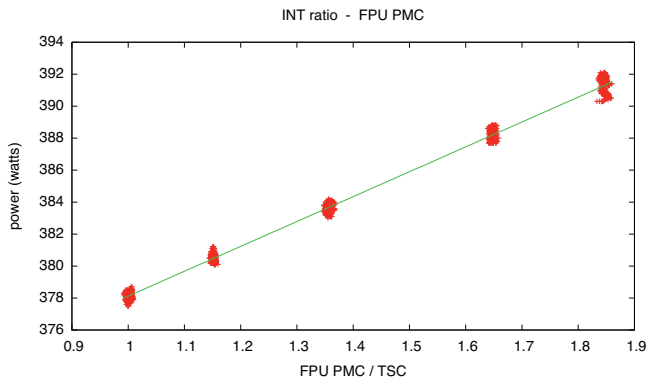


Fig. 5. Intensity of FPU correlated with power, sampled every second for the micro-benchmark running INT workload. Sampling starts after a 60-s period of CPU warm-up.

minutes! In many cases this will be longer than the execution time of the benchmarks.

To further confirm this relationship, IPMI was used to monitor the fan speed for each CPU. Despite the temperatures changing, the fans' speed remains constant at 3600 rpm. This speed is even maintained under a high thermal load running a CPUburn benchmark [15]. Contrary to the common belief, the power latency is not caused by changes in the speed of the fans when they respond to an increased thermal load. Therefore, policies designed to lock fan speeds through the BIOS are not capable of negating all of the dynamic thermal effects on power consumption.

3.2.2. Observation: warm-up and cool-down

In an attempt to remove the effect of the warm-up phase, the micro-benchmark was re-run after a CPU warm-up phase. There is a 15-min cool-down period before each iteration of the micro-benchmark starts to execute, in order to ensure consistent starting temperature for each iteration. After that, an instance of the CPUburn was run on each core for different execution times, providing different durations of CPU warm-up. The results for two different CPU warm-up lengths of 60 and 90 s are shown in Figs. 5 and 6, respectively. The axes are the same as each of the previous figures: the *x*-axis is the intensity of the FPU activities, while the *y*-axis is the power in Watts. This time, the INT ratio is adjusted between iterations, giving the spread of clusters along the *x*-axis.

A CPU warm-up phase of 60 s, as shown in Fig. 5, is enough to eliminate much of the vertical tail caused by the warm-up phase, resulting in a stronger linear correlation. Alternatively, with the CPU warm-up phase of 90 s, as shown in Fig. 6, it begins to

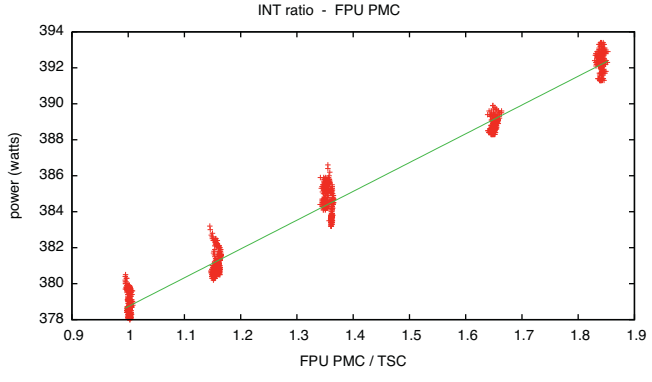


Fig. 6. Intensity of FPU correlated with power, sampled every second for the micro-benchmark running INT workload. Sampling starts after a 90-s period of CPU warm-up.

Table 2
Pearson's Correlations for different workload type with different warm-up times.

CPU warm-up time (s)	FPU-type	INT-type
30	0.969611	0.977467
60	0.992794	0.997928
90	0.996615	0.992321
120	0.993053	0.985091

over-warm the CPU, which causes the opposite effect, a vertical stripping above the main linear trend, instead of the vertical tail.

However, different types of workload in the benchmark need different warm-up periods. A 60-s CPU warm-up phase provides the best results for a workload with lots of integer calculations, as shown in Fig. 5. However, a workload with lots of floating point calculations requires a 90-s CPU warm-up. These results are illustrated by the Pearson's Correlation Coefficient for the two workload types in Table 2. While differences in correlations are not significant they do illustrate the point that no single CPU warm-up policy is sufficient for all workload types.

3.2.3. Discussion

Temperature variations within a system have the potential to adversely impact the accuracy of a power estimation model. For example, an instance of the micro-benchmark running an FPU workload initially uses 397 W, stabilizing at around 405 W. This gives an error of 2% of total power, and more significantly, 5% of dynamic power. Due to the myth of the thermal effects, there is no single solution designed to mitigate all thermal effects.

Unfortunately, it is not likely that a single policy exists to reliably remove all warm-up effects on power consumption. The most likely cause for the warm-up effects is static power leakage from the processor, which is due to the high temperatures. For example, a 12% reduction in CPU (dynamic) power consumption was made, as documented in an Anandtech article [16], by reducing the operating temperature, while maintaining the same voltage and frequency 4.6 GHz.

It might seem that the only way to reliably monitor thermal effects is through embedded temperature sensors. However, since their placement inside the socket is some distance away from the top of CPU, embedded sensors do not provide reliable temperature data either [17]. Also such sensors were not designed for high precision temperature reading; their primary purpose is to provide an early warning system to prevent hardware damage.

In summary, we have made two key observations. First, that benchmarks can experience a large warm-up effect on power consumption during their initial execution start-up time, which is not due to the changes in cooling fan speed. Second, the length of warm-up phase required varies between different workloads, as illustrated through differences in the CPU warm-up times required by FPU and INT workloads. That means using a fixed period of warm-up for all workloads will not achieve the desired result.

3.3. Myth 3: Memory events correlate well with power consumption

The correlation between memory-related PMCs, such as cache miss, and memory activities is intuitive and well known. However, there is a myth that memory-related PMCs correlate equally well with power consumption. This proved not to be true for our multicore system.

3.3.1. Observation: neutrality of memory-related PMCs

Memory-bound and CPU-bound workloads exhibit quite different power characteristics and in most cases are therefore treated differently. For power estimation, it is also common to use those

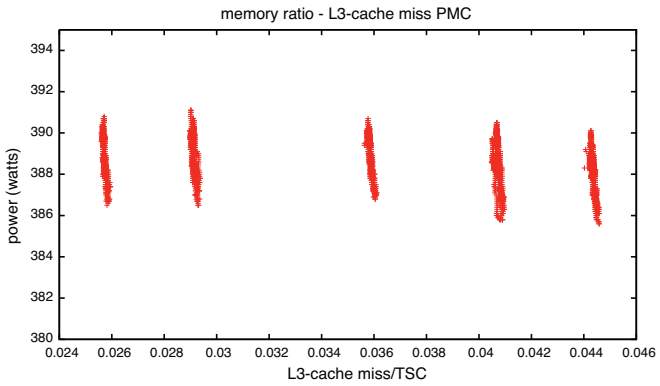


Fig. 7. Intensity of L3-cache misses correlated with power, sampled every second for the micro-benchmark running memory workload.

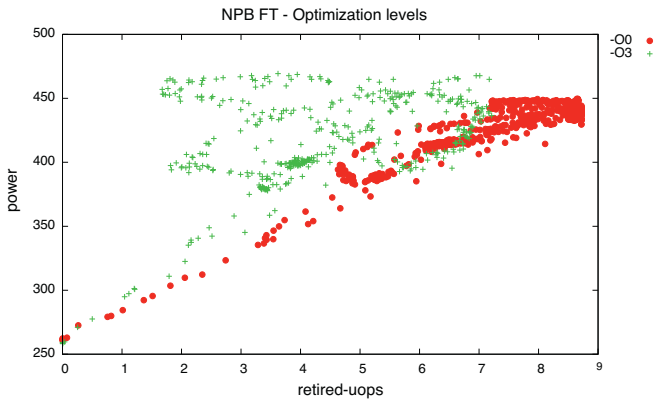


Fig. 8. Scatter plot of power and retired-uops PMC values for two executions of the FT NPB benchmark. Each benchmark is compiled using a different optimization parameter.

PMCs with a direct logical connection to memory use. This intuitively makes sense, and is what we expected to be the case too. In modeling the power use of memory workloads, we used PMCs directly related to memory, such as, instruction cache miss, data cache miss, L2-cache miss, L3-cache miss, DTLB miss and DRAM accesses.

Surprisingly, none of these counters provided a strong correlation between memory use and power consumption. To illustrate this, the results for L3-cache misses are plotted in Fig. 7. Our micro-benchmark has been configured to execute a large number of memory accesses by increasing the ratio for memory accesses, leaving all other micro-benchmark ratios at the default value of one. The *x*-axis is the intensity of cache misses, which is calculated taking the difference of two L3-cache-miss PMC values divided by the elapsed TSC value. The *y*-axis is the measured power. We collect the intensity and power samples once every second. The warm-up effect, seen by the vertical stripping within each cluster, is present since there is no CPU warm-up phase before data collection.

The most notable observation to be made in Fig. 7 is the distinct lack of any vertical offset between clusters. It seems that power is not functionally determined by L3-cache misses. The same results have been found for all other memory-related PMCs mentioned, so we do not repeatedly show the results here.

3.3.2. Discussion

A common approach taken in building a power model is to decompose the processor and the expected workload type into several key components, such as FPU, Memory, Stalls, and Instructions Retired [2]. Each component requires a specific, strongly correlated PMC to represent its power consumption. In the case of memory, a PMC such as 'cache misses' is expected to correlate well with the activities of the memory subsystem. This approach has worked in other power estimation models [4], but was not able to be reproduced on our experimental system, in which there is much less of a link between the aforementioned memory-related PMCs and power consumption.

This difference of results can likely be attributed to the architectural differences, though we are not sure which components actually have caused the difference, as there are several components which could possibly be playing a part. The first component could be the memory architecture. Our system uses NUMA, where, unlike the system architectures used in previous work, there is no single memory bank shared between all processors. The memory in our system is arranged in 4GiB blocks besides each of the four processors. Given a workload of random memory accesses, extra overhead may be incurred if memory accesses are shifted to a remote processor's memory block.

Also the processors in our system lack the ability to sleep, even at low levels. That means the processor maintains a busy loop or executes some other work while waiting for requests from the memory subsystem. Given the high thermal latencies, temperatures will remain high, despite lower levels of utilization.

3.4. Myth 4: Compilation configuration does not need to be reported

The compiler configuration and relevant software versions were not always given for all stages of model building and evaluation in previous work. Yet, as we demonstrate below, different compiler configurations can have a significant impact on power measurement.

3.4.1. Observation: optimization matters

When deriving the power model, both benchmarks and micro-benchmarks are often configured to use no optimizations [2] in order to ensure consistent performance counter readings, thereby improving regression analysis. This results in a model that provides a much stronger fit to the training data. While this is desirable, it implicitly acknowledges that the use of optimization flags will change execution behavior.

As a first step to understanding and potentially incorporating such effects into the model, we looked at the impact of two optimization levels on two different types of workload on benchmarks from the NAS Parallel Benchmark suite. For the optimization levels, `-O0` and `-O3` were chosen. This is because these are the two most commonly used configurations. `-O0` is used to provide no compiler optimizations during the use of micro-benchmarks when deriving the model. Alternatively, `-O3` is one of the most commonly used configurations for standard benchmark execution, providing a reasonable performance improvement in most situations. For the benchmarks, a computation intensive workload is represented by the fast Fourier Transform (FT). In contrast, the Data Cube (DC) benchmark provides a more memory intensive workload.

The use of optimization when compiling the benchmarks contributes a large amount of noise to the PMC data, with much more variation in the power used. An example of this can be seen in Fig. 8, where two instances of the FT benchmark are shown, each using a different optimization argument. The *x*-axis plots the intensity of the retired-uops PMC, while the power for each corresponding one second sample is given on the *y*-axis. From this it can be seen that the use of no optimization has a much narrower spread of power

Table 3
Statistics for different optimization levels.

Power	FT -O0	FT -O3	DC -O0	DC -O3
Mean	430.18725	418.92125	291.1435	284.987
Std	21.340075	33.609875	43.483825	38.831125
Median	439.0	421.125	270.85	266.4
Max	449.3	471.05	410.45	395.025

use for each corresponding PMC utilization value. Alternatively, the use of compiler optimization has a much greater spread of power values, making it much harder to predict power. Overall, the use of optimization causes a shift of the main cluster down the *x*-axis, indicating lower retired-uops than that of the instance using no optimization. From this initial view, it is clear that the use of optimization for a benchmark changes the runtime utilization levels, and power, meaning a power model unaware of optimization may end up being inaccurate.

While the optimization reduced the retired-uops intensity, it did not have an impact on the FPU intensity. The overall impact of optimization can be seen in Table 3, where a much greater variation in power is seen, both from the standard deviation and maximum value, however optimization results in lower power usage overall.

The more memory intensive workload—DC—similarly has overall lower power values when more optimization is performed. However in contrast to FT, this time a much lower maximum power leads to the standard deviation being smaller with the use of optimization. This can most likely be attributed to the optimizer's ability to reduce the impact on power of the periodic, compute intensive phases on the otherwise memory intensive workload.

3.4.2. Observation: impact of compiler version

A further source of power variation not visible to the power estimation model is the compiler version used. Unlike the effects caused by compiler optimizations, the execution characteristics of each benchmark does not appear to change significantly with different compiler versions. Note this may just be due to limited testing regarding both compiler versions and benchmarks.

However, changing the version of `gcc` from 4.6.3 to the older 4.4.7 did result in an increased maximum power level reached. This can be seen in Fig. 9, which plots the power measurements once a second for an execution of the FT benchmark. The main point of difference seen between the two compiler versions is the peak power, where gcc-4.6.3 has a peak power of 449.4 W, while gcc-4.4.7 is 8.5 W higher at 457.9 W. Interestingly, there is little variation in power at other stable points, such as the period of approximately 410 W power consumption towards the end of execution.

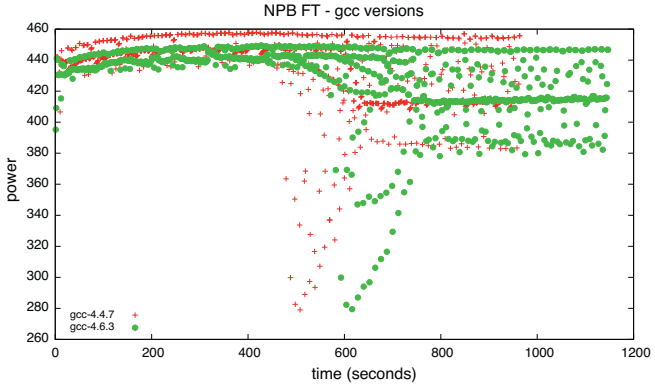


Fig. 9. Power versus time for two executions of the FT benchmark, that use gcc-4.4.7 and gcc-4.6.3 respectively.

3.4.3. Discussion

It is widely acknowledged that PMC-based power estimation models are architecture specific, however it is less well known that they can be significantly compilation specific. The potential for this limitation has been shown here with the observations regarding compiler optimization and the version of compiler used. Such configuration details have not been a significant concern in much of the existing literature, given that the models and benchmarks used during evaluation are strictly controlled by the experimenter. However, if such models are to be used in a more general deployment, they will have to additionally be capable of estimating power for user application binaries, where the compiler version and configuration are not known and may vary significantly from that used when initially deriving the model.

Furthermore, the significance of each compilation parameter will be architecture specific, meaning that other systems may suffer much larger variations in power. For instance, the impact of different MPI libraries was noted in [18], where the two instances had power values of ≈120 W and ≈140 W, which is a large percentage variation.

Some of this impact could potentially be mitigated by incorporating some compilation derived variation into both the training data sets as well as with benchmarks used for evaluating and tuning the model.

3.5. Myth 5: Metrics for performance evaluation of models are comparable

A great deal of care is taken to avoid problems while building the power model, however less care is taken to provide the fairest possible comparison between different models. In this section we will show how to present results with comparable metrics, and make fair comparison among alternative models.

3.5.1. Observation: static versus dynamic power

A commonly omitted metric in a power estimation model is the amount of static versus dynamic power within a system. Static power is the constant, workload-independent power consumption of components like the Power Supply Unit (PSU). Dynamic power changes according to the workload, with the most obvious example being the effect of the CPU. Knowing the amount of static and dynamic power is important, because PMC-based power models essentially only model the dynamic power, which is the response in power to a given change in utilization and performance values. Making this distinction allows for comparisons to be made between models evaluated on different systems.

For example, suppose there are two different power models with an identical mean estimation error of 5%, but for different machines with an equal total power consumption of 200 W. Without further information, the natural conclusion that can be drawn by the reader is both power models are equally accurate. However, a different conclusion can be drawn if it was additionally known that machine 'A' has a dynamic power (aka workload dependent power) of 30%, while machine 'B' has a larger dynamic power of 60%. Given a total error of 10W (5% of total 200W) and the constant static power (aka base power), it is now clear that the model based on machine 'A' has a 16.6% error for dynamic power while the model based on machine 'B' has a much lower error of 8.3% for dynamic power. The extra information on the proportion of static power and dynamic power enables a fair comparison to be made between the two, otherwise apparently identical power estimation models.

Therefore, without this additional contextual information, it is not possible to make a fair, unbiased comparison between different methodologies, derived and evaluated on two separate systems, with different power characteristics.

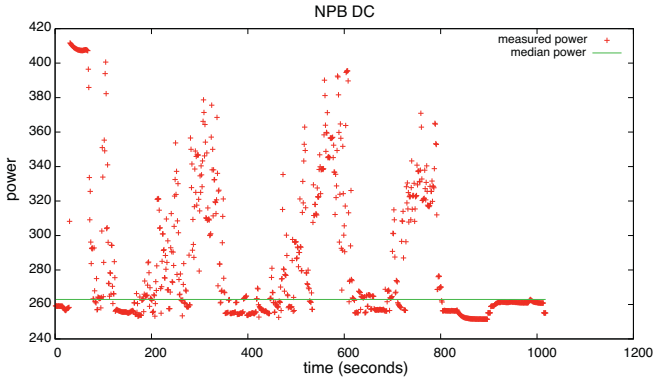


Fig. 10. Power versus time for the DC benchmark.

In some more extreme cases, a high static power can obscure a very inaccurate model. This is more likely on commodity systems that incorporate energy efficient hardware, such as low power processors. As a result, they have a very narrow range of operating power levels, where the dynamic power makes up a small proportion of total system power. Alternatively, this is not possible with more powerful, power hungry enterprise servers, which are capable of having dynamic power ranges in excess of the base static power.

Making a distinction between static and dynamic power allows for comparisons to be made between models, but it can also aid in evaluating the significance of factors contributing errors to power estimation. For instance, in the last observation, it was found that a change in compiler version caused a corresponding change in the maximum power of 8.5 W, or just under 2%, which does not seem very significant. However, if this error is considered as a change in dynamic power, it becomes a more sizable impact of 4%, which is an error worth attempting to mitigate.

3.5.2. Observation: statistical evaluation

Having derived a power estimation model, it is important to adequately evaluate the estimation error for the test benchmark suite. One of the most common measures for estimation is the median error for each test benchmark. However, such a metric is not sufficient to truly represent the closeness of fit for the model and can lead to a misrepresentation of some results. In the most naïve cases, the absolute value for the measured error is not taken, leading to a significantly overstated accuracy of fitting. This is due to the tendency for the positive errors of over-fitting to cancel out the corresponding negative values of under-fitting or vice versa.

However, even if the absolute values of errors are used, the median error is not descriptive enough to give an adequate representation of the actual fitting. This is in a large part due to the type of benchmarks used for model evaluation, scientific workloads, and the structure of distinct workload phases. This is illustrated in Fig. 10, which plots power use of the DC benchmark during execution. The *x*-axis gives time in seconds, where the power is measured once a second, which is then shown on the *y*-axis. It can be seen that DC has two distinct workload phases. The first is a dominant, memory-bound workload, with low levels of power consumption: around 260 W. The other workload is a periodic computation intensive phase represented by the sudden, short spikes in power use. There is an additional horizontal line in the graph at 262.9 W, which represents the median measured power value. From this, it can be concluded that the dominant workload is the memory phases, because the median value is down near the bottom of the graph. Therefore, at least half of the time is spent executing this lower power phase.

Thus, a power model that is capable of a low estimation error for the low-power, memory workload will achieve a very low median estimation error. This in no way provides any indication that the estimation model is able to accurately estimate the less common, computational phases of execution. In fact, it does not have to do so in previous research in order to achieve a low median error.

A better metric to be used for evaluating the power estimation model is the Mean Absolute Error (MAE), which calculates the absolute error for every power measurement before taking the average of all of these values. As opposed to the median, the mean will be influenced by outliers, meaning that an inability to model the changes in workload phases will result in larger errors. While this provides an improvement over median error, a single metric cannot be guaranteed to provide a fair comparison for the quality of fitting in all situations. Therefore, the MAE should be further supplemented with the standard deviation of the errors, providing an illustration of how widely spread the errors are. This can give further insight into the accuracy of the model without requiring a plot of absolute values.

In summary, care must be taken in using metrics in evaluation of models in order to make fair comparison between different models. There is a need in this research field to standardize the metrics used in model evaluation.

4. Methodology

While it may be perceived that adopting the guidance provided with each myth will require significant adaptation to existing modeling methodologies, we do not believe this is the case. To help illustrate this point, this section presents a high-level methodology for deriving a general power model, adhering to all of the points previously discussed. The modeling methodology consists of four primary steps: (1) performance event selection, (2) training data collection, (3) model construction, and (4) model evaluation. We leave a more detailed methodology specification and the corresponding implementation for future work.

4.1. Performance event selection

The first step involved in deriving a power estimation model is to determine the performance events with the strongest correlation to power. To achieve this, performance events are recorded during the execution of a diverse set of application workloads. It is important to ensure that these workloads provide a representative sample of execution characteristics and utilization levels. Therefore, either an extensive selection of micro-benchmarks are used, or more commonly, the applications from a benchmark suite, such as SPEC or NPB. Multiple benchmark iterations will be required when collecting a reasonable number of performance events, potentially requiring long data collection times. This is primarily due to the microarchitectural dependence of some performance events, such as PMCs, which restricts the number of events that can be simultaneously monitored.

Once the data set of power and performance values has been collected, statistical analysis is used to determine the performance events providing the best predictors of power. The two most commonly used statistical methods are: principle component analysis, which is a statistical procedure that determines the principal performance events that account for the variation in power, and correlation analysis, which provides a measure of the strength of relationship or dependence between two variables. For large data sets, this analysis can become time consuming. Therefore, as a commonly used compromise, domain specific knowledge is used to restrict the performance events to those corresponding to key components of interest. In addition to reducing the time taken to

analyze the data set, it will also reduce the number of iterations performed during data collection, saving more time.

4.2. Training data collection

While the benchmarks used in the previous step provided a broad representation of various application utilization levels, they still may not be diverse enough to produce an unbiased, robust power model. Therefore, micro-benchmarks play a more prominent role in the collection of model training data. Incorporating a tailored range of utilization levels ensures the model is not overly influenced by any particular benchmark suite. Configuring the micro-benchmarks to stress different system components or workload characteristics beyond the typical ranges, ensures additional robustness. However, the micro-benchmark workloads have the potential to create misleading effects by utilizing each functional unit in turn, thereby neglecting cross-component interactions which may be observed during typical benchmark execution, meaning benchmarks may still have a small role to play in model training.

It will be impractical to run the suite of micro-benchmarks in all possible configurations, as the potential search space will be significant. A tradeoff between training time, model complexity and potential accuracy is required. A reasonable balance is achieved by staying within typical execution ranges for the majority of the training time, while collecting a limited set of samples for boundary cases. This allows the model to accurately reflect expected workload types, while having some knowledge of alternative workloads.

4.3. Model construction

The power model can be derived by quantifying the strength of relationship between the explanatory variables, the performance events used to estimate power, and the dependent power variable. This problem is well suited to regression analysis, which is why least-squares regression is one of the most broadly adopted modeling techniques. A linear model is fitted to the data from the training set by minimizing the square of the residuals. Despite its simplicity, linear models have been found to provide good estimation accuracy in much of the previous work [2], making it an attractive solution. Alternative models have been used, such as the classification-based Gaussian mixture models used in [8], but the added complexity is often not warranted.

The resulting accuracy of the model will largely depend on the quality of the input training data set collected in the previous step. For instance, if the training data set largely consists of CPU-intensive workloads, the model will make poor estimations of power for memory-intensive workloads, and vice versa. The more extensive the set of performance values and power measurements, the better the model will likely be. However, the appropriateness of the model and training data will only be revealed after performing a thorough model evaluation.

4.4. Model evaluation

Model evaluation is performed on a benchmark suite, where power estimates are compared to the measured power. To ensure an unbiased evaluation, the benchmark suite should not be the same as was used during model training. A well-chosen benchmark suite should be capable of validating how well a model will generalize across a diverse set of workload types and system utilization levels. It will also indicate if the model fails on any particular workload type, which may be a symptom of insufficient or limited training data.

One of the best methods for gaining cursory insights into the accuracy of the power model is to observe the measured and modeled power on scatter plots. However, this method can be time consuming and is only appropriate for initial insights or to gain further insights into potentially anomalous results. The next step is to apply the statistical methods discussed in Section 3.5. That is, MAE and standard deviation, which together provide descriptive insights into the closeness of fit.

4.5. Discussion

Our proposed, general methodology is no more complex than many of the alternatively proposed methodologies. Similar to other models, tradeoffs are required between model complexity, training time and model accuracy. However, these tradeoffs may be implementation specific in nature and vary between different experimental systems. For instance, if the system has little variation in the dynamic power range, a simple model will achieve good results. Whereas the same model, applied to a system with a more extensive range of dynamic power, will likely be less accurate. Therefore, the methodology implementation may require an iterative process to be performed, repeatedly testing and evaluating various input configurations to achieve a reasonable accuracy.

While the presented methodology primarily discusses PMCs, it is general enough to allow for extensions in the type of performance events being monitored without a detrimental impact on model complexity. The only effect will be increased training time due to a larger set of performance events and the likely inclusion of new, targeted training benchmarks. For instance, kernel performance statistics, such as the number of packets received/transmitted or the number of blocks read from/written to disk, could additionally be incorporated into the model, allowing hard drives and NICs to be monitored. The ability to model similar peripheral devices will likely benefit non-processor intensive workloads, like memory-bound tasks and distributed MPI applications. By proposing a suitably general methodology, it can readily be extended to include new performance events, making it similarly suited to incorporate the general guidance provided for each myth without detrimental impact.

5. Related work

Much of the prior work on PMC-based power estimation has taken the approach of using PMCs to model the underlying architectural components, which are applied in a variety of use cases. Singh et al. [2] proposed a model which used microarchitectural knowledge to decompose the processor into its four main functional units: FP Units, Memory, Stalls, and Instructions Retired. PMC selection is made from initial data collected from the execution of the SPEC benchmark suite. A separate micro-benchmark is designed for each of the four PMCs most strongly correlated with power for each functional unit. The micro-benchmark data is used to form a piece-wise linear function.

Bertran et al. [4] take an even finer-grained approach by starting with a set of about 97 micro-benchmarks designed to individually highlight all possible power components. This results in multiple linear equations with an input for each of the seven derived power components. During the runtime period, PMC multiplexing is required, as the micro-architecture does not allow that many counters to be collected simultaneously.

Da Costa et al. [3] present a methodology intended to broaden the range of modeled workloads by supplementing PMC values with process and system level statistics. This means the resulting model, derived through multivariate regression, is not limited to estimating the power of CPU and memory workloads,

allowing accurate power estimation of the network and disk synthetic benchmarks.

Such models derive a single, global power estimation function typically used for monitoring system power on a per application basis. Alternatively, Alonso et al. [5] take a more targeted approach in proposing a framework for instrumenting source code functions with power metering. The API logs PMC data and power values to derive a specific power model offline, enabling execution traces of power to be used during runtime estimation.

Wang et al. [6] take the novel approach of using the fewest PMCs possible. The model was built using only CPU operating frequency and IPC, making it universal across micro-architectures. It is built into the SPAN libraries and interfaces to provide source code power estimation.

Dhiman et al. [8] presented a model for accurate power estimation in a virtualized environment. A performance counter manager is run on each host machine, designed to collect and correlate PMC events to each VM. Power estimates are then periodically made for each VM using classification-based Gaussian mixture models.

Only a select sample of previous work is presented here to demonstrate some alternate uses for PMC-based power estimation models. A more comprehensive survey on hardware, software and hybrid power estimation techniques can be found in [6]. Given this varied use of PMC-based power models, each myth within this paper was presented and discussed without explicit guidance, ensuring all conclusions remain relevant and universally applicable to these varied approaches.

6. Conclusions and future work

In this paper we have presented and discussed five myths in PMC-based power estimation models: sampling rate and execution time can be left unspecified; thermal effects are insubstantial; memory events correlate well with power consumption; compilation configuration does not need to be reported; and metrics for performance evaluation of models are comparable. The underlying misconceptions within each myth are revealed through a series of observations made while deriving our own PMC-based estimation model.

Such myths have arisen due to a lack of configuration specifications and accompanying analysis in published literature. This is of particular importance for PMC-based power estimation as the models derived are largely architecture dependent. However, the myths are not limited to PMCs, but equally apply to many other potential performance events. As we have seen, changes in hardware and software configurations can adversely impact the resulting power models. Therefore, a suitably general modeling methodology, as was discussed, is important for deriving a robust power model.

While failing to disclose all relevant information would not affect the correctness of the previous work, it can lead to erroneous conclusions being drawn from the results. In raising awareness of the impact of this missing, but highly relevant information, we hope researchers in this community more readily document hardware and software configurations in the future. Doing so will prove to be beneficial to the advancement of the research community.

In the future we will implement the PMC-based power estimation model, following the general methodology presented. The general guidance corresponding to each of the myths discussed in this paper will be incorporated into the model. Furthermore, we intend to include additional sources of performance events, enabling the power estimation of highly communicating applications, helping derive a robust and accurate power model.

Acknowledgement

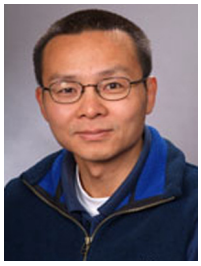
This work was partially supported by the COST (European Cooperation in Science and Technology) framework, under Action IC0804.

References

[1] X. Chen, C. Xu, R. Dick, Z. Mao, Performance and power modeling in a multi-programmed multi-core environment, in: Proceedings of the 47th Design Automation Conference, ACM, 2010, pp. 813–818.
[2] K. Singh, M. Bhadauria, S.A. McKee, Real time power estimation and thread scheduling via performance counters, in: SIGARCH Computer Architecture News, vol. 37, ACM, 2009, pp. 46–55.
[3] G. Da Costa, H. Hlavacs, Methodology of measurement for energy consumption of applications, in: 2010 11th IEEE/ACM International Conference on Grid Computing (GRID), 2010, pp. 290–297.
[4] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguade, Decomposable and responsive power models for multicore processors using performance counters, in: ICS'10: Proceedings of the 24th ACM International Conference on Supercomputing, ACM, New York, NY, 2010, pp. 147–158.
[5] P. Alonso, R.M. Badia, J. Labarta, M. Barreda, M.F. Dolz, R. Mayo, E.S. Quintana-Ortí, R. Reyes, Tools for power and energy analysis of parallel scientific applications, in: Proc of International Conference on Parallel Processing (ICPP), 2012.
[6] S. Wang, H. Chen, W. Shi, SPAN: A software power analyzer for multicore computer systems, Sustain. Comput. Inform. Syst. 1 (1) (2011) 23–34.
[7] A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya, Virtual machine power metering and provisioning, in: Proceedings of the 1st ACM symposium on Cloud computing, 2010, pp. 39–50.
[8] G. Dhiman, K. Mihic, T. Rosing, A system for online power prediction in virtualized environments using Gaussian mixture models, in: Design Automation Conference (DAC), 2010 47th ACM/IEEE, 2010, pp. 807–812.
[9] A. Naveh, D. Rajwan, A. Ananthakrishnan, E. Weissmann, Power management architecture of the 2nd generation Intel® Core™ microarchitecture, formerly codenamed Sandy Bridge.
[10] E. Rotem, A. Naveh, A. Ananthakrishnan, D. Rajwan, E. Weissmann, Power-management architecture of the Intel microarchitecture code-named sandy bridge, IEEE Micro 32 (2) (2012) 20–27.
[11] O.A.R. Board, OpenMP Application Program Interface Version 3.0, 2008 http://www.openmp.org/mp-documents/spec30.pdf
[12] Watts Up? Operators Manual, 2013 http://www.wattsupmeters.com/secure/downloads/manual_rev_9_corded0812.pdf
[13] IPMItool, http://ipmitool.sourceforge.net, 2013.
[14] AMD, BIOS and Kernel Developer's Guid (BKDG) For AMD Family 10h Processors, http://support.amd.com/us/Processor_TechDocs/31116.pdf, 2013.
[15] U. Manuals, CPUburn, http://manpages.ubuntu.com/manpages/precise/man1/cpuburn.1.html, 2013.
[16] Anandtech, Overclocking CPU/GPU/Memory Stability Testing Guidelines, http://forums.anandtech.com/showthread.php?p=34255681, 2013.
[17] Overclockers, Reconciling CPU Temperature Measures, http://www.overclockers.com/reconciling-cpu-temperature-measures/, 2013.
[18] K.W. Cameron, Energy oddities. Part 2: Why green computing is odd, Computer 46 (3) (2013) 90–93.

Jason Mair received his BSc Hons degree from the University of Otago, New Zealand in 2010. He is currently completing a PhD in Computer Science at the University of Otago, New Zealand. His research interests include multicore architectures, green computing, and distributed/cloud computing.

David Eyers received his PhD in Computer Science in 2006 from the University of Cambridge, UK, having previously attained his BE in Computer Engineering from the University of New South Wales in Sydney, Australia. He is a Senior Lecturer at the Department of Computer Science at the University of Otago in New Zealand, and a visiting research fellow at the University of Cambridge Computer Laboratory. He has broad research interests including green computing, information flow control, network security, and distributed and cloud computing.



Zhiyi Huang received the BSc degree in 1986 and the PhD degree in 1992 in computer science from the National University of Defense Technology (NUDT) in China. He is an Associate Professor at the Department of Computer Science, University of Otago, New Zealand. He was a visiting professor at EPFL (Swiss Federal Institute of Technology Lausanne) and Tsinghua University in 2005, and a visiting scientist at MIT CSAIL in 2009. His research fields include parallel/distributed computing, multicore architectures, operating systems, green computing, cluster/grid/cloud computing, high-performance computing, and computer networks.

Haibo Zhang received the MSc degree in Computer Science from Shandong Normal University, China in 2005, and the PhD degree in Computer Science from the University of Adelaide, Australia in 2009. From 2009 to 2010, he was a postdoctoral research associate at Automatic Control Laboratory, KTH, Sweden. Currently he is a lecturer at Computer Science department of University of Otago, New Zealand. His current research interests include real-time industrial wireless communications, wireless sensor/ad hoc networks, cyber-physical systems, green computing, distributed algorithms and protocol design.