THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Measurement, Modeling, and Characterization for Power-Aware Computing

BHAVISHYA GOEL

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2014

Measurement, Modeling, and Characterization for Power-Aware Computing
Bhavishya Goel

Technical report 111L
ISSN 1652-876X
Department of Computer Science and Engineering
Computer Architecture Research Group

Division of Computer Engineering
Chalmers University of Technology
SE-412 96 GÖTEBORG, Sweden
Phone: +46 (0)31-772 10 00

Author e-mail: goelb@chalmers.se

Chalmers Reposervice
Göteborg, Sweden 2014

Measurement, Modeling, and Characterization for Power-Aware Computing

Bhavishya Goel

Division of Computer Engineering, Chalmers University of Technology

ABSTRACT

Society's increasing dependence on information technology has resulted in the deployment of vast compute resources. The energy costs of operating these resources coupled with environmental concerns have made power-aware computing one of the primary challenges for the IT sector. Making energy-efficient computing a rule rather than an exception requires that researchers and system designers use the right set of techniques and tools. These involve measuring, modeling, and characterizing the energy consumption of computers at varying degrees of granularity.

In this thesis, we present techniques to measure power consumption of computer systems at various levels. We compare them for accuracy and sensitivity and discuss their effectiveness. We test Intel's hardware power model for estimation accuracy and show that it is fairly accurate for estimating energy consumption when sampled at the temporal granularity of more than tens of milliseconds.

We present a methodology to estimate per-core processor power consumption using performance counter and temperature-based power modeling and validate it across multiple platforms. We show our model exhibits negligible computation overhead, and the median estimation errors ranges from 0.3% to 10.1% for applications from SPEC2006, SPEC-OMP and NAS benchmarks. We test the usefulness of the model in a meta-scheduler to enforce power constraint on a system.

Finally, we perform a detailed performance and energy characterization of Intel's Restricted Transactional Memory (RTM). We use TinySTM software transactional memory (STM) system to benchmark RTM's performance against competing STM alternatives. We use microbenchmarks and STAMP benchmark suite to compare RTM versus STM performance and energy behavior. We quantify the RTM hardware limitations that affect its success rate. We show that RTM performs better than TinySTM when working-set fits inside the cache and that RTM is better at handling high contention workloads.

Keywords: power estimation, energy characterization, power-aware scheduling, power management, transactional memory, power measurement

Preface

Parts of the contributions presented in this thesis have previously been published in the following manuscripts.

▷ **Bhavishya Goel**, Sally A. McKee, Roberto Gioiosa, Karan Singh, Major Bhadauria, Marco Cesati, "Portable, Scalable, Per-Core Power Estimation for Intelligent Resource Management" in *Proceedings of the 1st International Green Computing Conference*, Chicago, USA, August, 2010, pp.135-146.

▷ **Bhavishya Goel**, Sally A. McKee, Magnus Själander, "Techniques to Measure, Model, and Manage Power," in *Advances in Computers 87*, 2012, pp.7-54.

The following manuscript has been accepted but is yet to be published.

▷ **Bhavishya Goel**, Ruben Titos-Gil, Anurag Negi, Sally A. McKee, Per Stenstrom, "Performance and Energy Analysis of the Restricted Transactional Memory Implementation on Haswell" To appear in *Proceedings of 28th IEEE International Parallel and Distributed Processing Symposium*, Phoenix, USA, May, 2014.

The following manuscripts have been published but are not included in this work.

▷ **Bhavishya Goel**, Magnus Själander, Sally A. McKee, "RTL Model for Dynamically Resizable L1 Data Cache" in *Swedish System-on-Chip Conference*, Ystad, Sweden, 2013

▷ Magnus Själander, Sally A. McKee, **Bhavishya Goel**, Peter Brauer, David Engdal, Andras Vajda, "Power-Aware Resource Scheduling in Base Stations" in *19th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Singapore, Singapore, July, 2012, pp.462-465.

Acknowledgments

I will like to thank following people for supporting me during my research career and playing their part in bringing this thesis to fruition.

▷ Professor Sally A. McKee for inspiring me to join the computer architecture research field and giving me the opportunity to work with her. She has been a great adviser and even greater friend. She is always ready to help out with any problems that I bring to her, technical or non-technical, and it is easy to say that I owe my research career to her. She is also an excellent cook and her brownies and palak paneers have got me through various deadlines.

▷ Magnus Själander for helping me with my research and providing me great technical feedback whenever we worked together. His enthusiasm for research is very contagious and has helped me in keeping myself motivated.

▷ Professor Per Larsson-Edefors for teaching some of the most entertaining courses I have taken at Chalmers and for his valuable feedback from time to time.

▷ Professor Lars Svensson for helping me with practical issues during his time as division head.

▷ Professor Per Stenström for introducing me to the field of computer architecture and being an inspiration.

▷ Ruben and Anurag for being wonderful co-authors and helping me in getting started with the basics of transactional memory.

▷ Karan Singh, Vincent M. Weaver and Major Bhadauria for getting me started with the power modeling infrastructure at Cornell and answering my naïve questions patiently.

- ▷ Roberto Gioiosa and Marco Cesati for helping me run experiments for my first paper.
- ▷ Madhavan and Jacob for sharing office space with me and listening to my jokes and rants.
- ▷ Alen, Angelos, Dmitry, Kasyab, Gabriele, Vinay, and Jochen for being wonderful colleagues and friends and for making me look forward to come to work.
- ▷ All the colleagues and staff at Computer Science and Engineering department for creating a healthy work environment.
- ▷ Anthony Brandon for giving me valuable support when I was working with ρVEX processor for ERA project.
- ▷ Eva Axelsson, Tiina Rankanen, Jonna Amgard and Marianne Pleen-Shreiber for providing excellent administrative support during my time at Chalmers.
- ▷ European Union for funding my research.
- ▷ My parents for supporting my career decisions and being great parents in general.

Bhavishya Goel
Göteborg, March 2014

Contents

List of Figures

List of Tables

1
Introduction

Information and Communications Technology (ICT) consumes a large and growing amount of power. In 2007-2008, multiple independent studies calculated the global ICT footprint to be 2% [1] of the total emissions. As per a European Commission press release in 2013 [2], ICT products and services are responsible for 8-10% of the European Union's electricity consumption and up to 4% of its carbon emissions. Murugesan [3] notes that each personal computer in use in 2008 was responsible for generating about a ton of carbon dioxide per year. Data centers consumed 1.5-2% of global electricity in 2011 and this is growing at a rate of 12% per year [4].

These statistics underscore the importance of reducing the energy we expend to avail ICT services. *Power-Aware Computing* is the umbrella term that has come to describe computing techniques that improve energy efficiency. For example, a power-aware system may be designed and optimized to consume lower power for doing a defined set of tasks. Alternatively, techniques can be employed to maximize the performance of the system for a defined constraint on power consumption and/or heat dissipation. The goal of power-aware computing is to avoid energy waste [5] by making informed decisions about power and performance trade-offs.

We classify power-aware techniques as shown in Fig. 1.1. The power-aware strate-

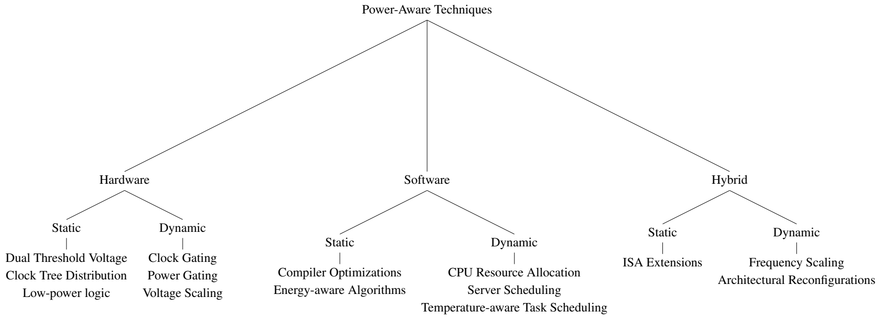


Figure 1.1: *Classification of Power-Aware Techniques*

gies can be implemented in hardware, software, or as a hybrid technique. Hybrid power-aware techniques are controlled by software with hardware support. Static power-aware techniques are controlled at compile-time or design-time for software techniques and during synthesis or design-time for hardware techniques. On the other hand dynamic power-aware techniques are based on the current state of the system during execution.

Power-aware techniques need information about power consumption from the system. The focus of this thesis is how to get this information through power measurement, power modeling, and power characterization.

1.1 Power Measurement

The first-step in formulating power-aware techniques is to measure power consumption. Readings from power measurement infrastructure can be used to identify energy inefficiencies in hardware and software. A setup for measuring system/processor power consumption needs to have the following properties:

- High accuracy;
- High sampling rate;
- Sensitivity to small changes in power consumption;
- Non-interference with the system-under-test;
- Low cost; and
- Ease of setup and use.

In addition to system power consumption, it may be desirable to get information about power consumption of the processor chips. However, hardware power meters are rarely implemented at the chip-level because of the associated hardware costs. The power consumption of the system can be measured in multiple ways. We compare different approaches to measuring the power consumption and discuss advantages and disadvantages of each.

1.2 Power Modeling

Depending on the requirements and costs, it may not be possible to use actual power measurements. For example, it may be too costly to deploy the power measurement setup at multiple machines, or the response time of the power measurement setup may not be fast enough for a particular power-aware technique. An alternative to actual power measurement is to estimate the power consumption using models that provide power estimates based on events in the system. They can be implemented in either hardware or software. Power models should have following properties:

- Small delay between input and output;
- Low overhead — low CPU usage if software model and low hardware area if hardware model;
- High accuracy; and
- Power consumption information of individual components (like cores or microarchitectural components).

Based on the requirements of the power-aware technique and trade-offs among costs, accuracy, and overhead, the system designer must decide whether to implement hardware or software power model. Intel introduced a hardware power model in its Core i7 processors starting with the Sandy Bridge microarchitecture [6]. The values from this power model are available to the software using Model Specific Registers (MSR) through the Running Average Power Limit (RAPL) interface. Similar model-based power estimates are available for AMD processors through their Application Power Management (APM) Interface [7], and on NVIDIA GPU processors through NVIDIA Management Library (NVML) [8]. In addition to these relatively new power models implemented in hardware by chip manufacturers, researchers have proposed software power models [9–18].

The information gained by power metering and power modeling can be used by resource management software to make power-aware decisions. Research studies have made use of live power measurement to schedule tasks and allocate resources at the level of individual processors [11,19,20,20–23] and large data centers [15,24–29]. Power/energy models have been used in many research studies to propose power-aware strategies for controlling DVFS policies [19,23], task scheduling in chip multiprocessors [24] and data centers [27,28], power budgeting [11,21], energy-aware accounting and billing [30], and avoiding thermal hotpsots in chip multiprocessors [31, 32].

In this thesis, we develop a performance event and temperature based per-core power model for chip multiprocessors. Our model estimates power in real time with low overhead.

1.3 Power Characterization

Power characterization analyzes the system's energy expenditure for varying workloads . This can be used to devise software and hybrid power-aware strategies, optimize workloads for energy efficiency, and optimize system design. Some of the challenges for energy characterization are:

- **Choosing representative workloads.** Workloads selected for characterization studies should be representative of those that are likely to be run on the system. The workloads should also exercise the full spectrum of identified system characteristics for extensive analysis.
- **Choosing good metrics.** The selection of the metrics to characterize and compare system energy expenditure depends on the type of characterization study and the emphasis that the researchers wants to put on delay versus energy expenditure. Possible metrics include total energy, average power, peak power, dynamic power, energy-delay product, energy-delay-squared product, and power density.
- **Setting up appropriate power metering infrastructure.** Any energy characterization study requires a means to measure/estimate and log system power consumption. Depending on the type of study, measurement factors like accuracy and temporal granularity must be considered. It may be useful to be able to decompose power consumption figures for different system components but that support may or may not exist. Researchers need to consider these factors and decide between measuring actual power versus modeling the power estimates.

Researchers have used energy characterization to understand energy-efficiency of mobile platforms [33–35] and desktop/server platforms [15, 36–38]. Energy characterization can also be used to analyze the energy efficiency of specific features of the system [39, 40]. The energy behaviors thus characterized can be used, for example, to identify software inefficiencies [33, 34, 36], manage power [15], analyze power/performance trade-offs [41], and compare energy efficiency of competing technologies [39]. Apart from actual power measurement, power models can prove to be useful for energy characterization [39, 41, 42].

In this thesis, we characterize performance and energy of Intel's recent Haswell microarchitecture with a focus on its support for transactional memory.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

- In **Chapter 2**, we explain different techniques to measure power consumption of the system, compare their intrusiveness, and give experimental results to show their accuracy and sensitivity. We test Intel's hardware power model for accuracy, sensitivity, and update granularity and discuss the results.
- In **Chapter 3**, we present a per-core, portable, scalable power model and show the validation results across multiple platforms. We show the effectiveness of the model by implementing it in an experimental meta-scheduler power-aware scheduling.
- In **Chapter 4**, we present performance and energy expenditure characterization results for Restricted Transactional Memory (RTM) implementation on the Intel Haswell microarchitecture. We use microbenchmarks and the STAMP benchmark suite to compare the performance and energy efficiency of RTM to TinySTM — a software transactional memory implementation.
- In **Chapter 5** we present our concluding remarks and discuss future research directions that can be taken from the work presented in this thesis.

2
Power Measurement Techniques

2.1 Overview

Designing intelligent power-aware computing technologies requires an infrastructure that can accurately measure and log the system power consumption and preferably that of the system's individual resources. Resource managers can use this information to identify power consumption problems in both hardware (e.g., hotspots) and software (e.g., power-hungry tasks) and then to address those problems (e.g., through scheduling tasks to even out power or temperature across the chip) [12,43,44]. A measurement infrastructure can also be used for power benchmarking [45, 46], power modeling [12, 13, 15, 43] and power characterization [36,47,48]. In this chapter, we compare different approaches to measuring actual power consumption on the Intel Core$^{\text{TM}}$ i7 platform. We discuss these techniques in terms of their intrusiveness, ease of use, accuracy, timing resolution, sensitivity, and overhead. The measurement techniques demonstrated in this chapter can be applied to other platforms, subject to some hardware support.

In the absence of techniques to measure actual power consumption, model-based power consumption estimation is also a viable alternative. Intel's Running Average Power Limit (RAPL) interface [6], AMD's Application Power Management (APM) in-

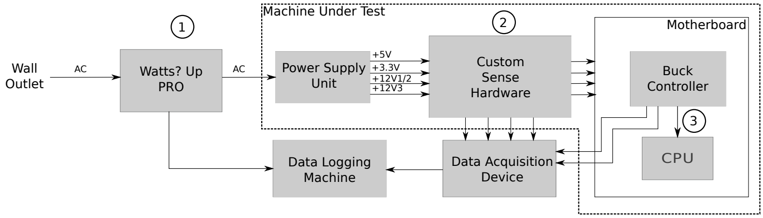


Figure 2.1: *Power Measurement Setup*

terface [7] and NVIDIA's Management Library (NVML) [8] interface make model-based energy estimates available to the operating system and user applications through model-specific registers, thereby enabling the software to make power-aware decisions.

In the rest of this chapter, we first describe the methodology of three techniques to measure actual power consumption, discuss their advantages and disadvantages, and collect experimental results on an Intel Core$^{\text{TM}}$i7 870 platform to compare their accuracy and sensitivity. We then compare one of these measurement techniques — reading power measurement from the ATX (Advanced Technology eXtended) power rails — to Intel's RAPL implementation on Core$^{\text{TM}}$i7 4770 (Haswell).

2.2 Power Measurement Techniques

Power consumption can be measured at various points in a system. We measure power consumption at three points, as shown in Fig. 2.1:

1. The first and least intrusive method for measuring the power of an entire system is to use a power meter like the *Watts up? Pro* [49] plugged directly into the wall outlet;

2. The second method uses custom sense hardware to measure the current on individual ATX power rails; and

3. The third and most intrusive method measures the voltage and current directly at the CPU voltage regulator.

2.2.1 At the Wall Outlet

The first method uses an off-the-shelf (*Watts up? Pro*) power meter that sits between the system under test and the power outlet. Note that to prevent data logging activity from disturbing the system under test, we use a separate machine to collect measurements for all three techniques, as shown in Fig. 2.1. Although easy to deploy and non-intrusive, this meter delivers only a single system measurement, making it difficult to separate the power consumption of different system components. Moreover, the measured power values are inflated compared to actual power consumption due to inefficiencies in the system power supply unit (PSU) and on-board voltage regulators. The acuity of the measurements is also limited by the (low) sampling frequency of the power meter: one sample per second (here on referred to as Sa/s) for the *Watts up? Pro*. The accuracy of the system power readings depends on the accuracy specifications provided by the manufacturer (±1.5% in our case). The overall accuracy of measurements at the wall outlet is affected by the mechanism converting between alternating current (AC) to direct current (DC) in the power supply unit. When we discuss measurement results below, we examine the accuracy effects of the AC-DC conversion done by the PSU.

This approach is suitable for studies of total system power consumption instead of individual components like the CPU, memory, or graphics cards [50,51]. It is also useful in power modeling research, where the absolute value of the CPU and/or memory power consumption is less important than the trends [43].

2.2.2 At the ATX Power Rails

The second methodology measures current on the supply rails of the ATX motherboard's power supply connectors. As per the ATX power supply design specifications [52], the power supply unit delivers power to the motherboard through two connectors, a 24-pin connector that delivers +5.5V, +3.3V, and +12V, and an 8-pin connector that delivers +12V used exclusively by the CPU. Table 2.1 shows the pinouts of these connectors. Depending on the system under test, the pins belonging to the same power region may be connected together on the motherboard. In our case, all +3.3 VDC pins are connected together, as are all +5 VDC pins and +12V3 pins. Apart from that, the +12V1 and +12V2 pins are connected together to supply current to the CPU. Hence, to measure the total power consumption of the motherboard, we can treat these connections as four logically distinct power rails — +3.3V, +5V, +12V3, and +12V1/2.

For our experiments, we develop custom measurement hardware using current transducers from LEM [53]. These transducers use the Hall effect to generate an output voltage in accordance with the changing current flow. The top-level schematic of the hardware is shown in Fig. 2.2, and Fig. 2.3 shows the manufactured board. Note that when designing such a printed circuit board (PCB), care must be taken to ensure that the current capacity of the PCB traces carrying the combined current for the ATX power rails is sufficiently high and that the on-board resistance is as low as possible. We use a PCB with 105 micron copper instead of the more widely used thickness of 35 microns. Traces carrying high current are at least 1 cm wide and are backed by thick-stranded wire

(a) 24-pin ATX Connector Pinout

Pin	Signal	Pin	Signal
1	+3.3 VDC	13	+3.3 VDC
2	+3.3 VDC	14	-12 VDC
3	COM	15	COM
4	+5 VDC	16	PS_ON
5	COM	17	COM
6	+5 VDC	18	COM
7	COM	19	COM
8	PWR OK	20	Reserved
9	5 VSB	21	+5 VDC
10	+12 V3	22	+5 VDC
11	+12 V3	23	+5 VDC
12	+3.3 VDC	24	COM

(b) 8-pin ATX Connector Pinout

Pin	Signal	Pin	Signal
1	COM	5	+12 V1
2	COM	6	+12 V1
3	COM	7	+12 V2
4	COM	8	+12 V2

Table 2.1: *ATX Connector Pinout*

connections, when required. The current transducers need +5V supply voltage, which is provided by the +5VSB (stand by) rail from the ATX connector. Using +5VSB for the transducer's supply serves two purposes. First, because the +5VSB voltage is available even when the machine is powered off, we can measure the base output voltage from the current transducers for calibration purposes. Second, because the current consumed by the transducers themselves (∼28mA) is drawn from +5VSB, it does not interfere with our power measurements. We sample and log the analog voltage output from the current transducers using a data acquisition (DAQ) unit from National Instruments (NI USB-6210 [54]).

As per the LEM datasheet, the base voltage of the current transducer is 2.5V. Our experiments indicate that the current transducer produces an output voltage of 2.494V when zero current is passed through its primary turns. The sensitivity of the current transducer is 25mV/A, hence the current can be calculated as in Eq. 2.1.

$$I_{out} = \frac{V_{out} - BASE\_VOLTAGE}{0.025} \tag{2.1}$$

We verify our current measurements by comparing against the output from a digital multimeter. The power consumption can then be calculated by simply multiplying the current with the respective voltage. Apart from the ATX power rails, the PSU also provides separate power connections to the hard drive, CD-ROM, and cabinet fan. To calculate the total PSU load without adding extra hardware, we disconnect the I/O de-

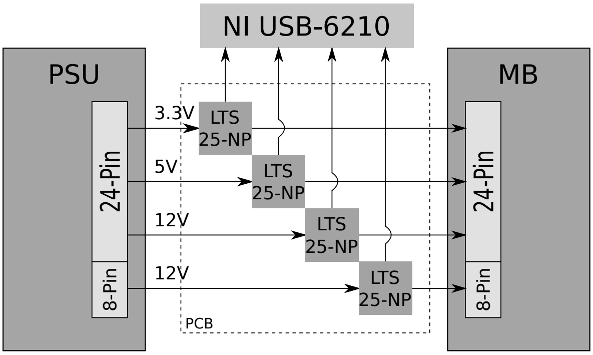


Figure 2.2: *Measurement Setup on the ATX Power Rails*

vices and fan, and we boot our system from a USB memory device powered by the motherboard. The total power consumption of the motherboard can then be calculated as in Eq. 2.2.

$$P = I_{3.3V} * V_{3.3V} + I_{12V3} * V_{12V3} + I_{5V} * V_{5V} + I_{12V1/2} * V_{12V1/2} \quad (2.2)$$

The theoretical current sensitivity of this measurement infrastructure can be calculated by dividing the voltage sensitivity of the DAQ unit (47μV) by the current sensitivity of the LTS-25NP current transducers from LEM (25mV/A). This yields a current sensitivity of 2mA.

This approach improves accuracy by eliminating the complexity of measuring AC power. Furthermore, the approach enjoys greater sensitivity to current changes (2mA) and higher acquisition unit sampling frequencies (up to 250000 Sa/s). Since most modern motherboards have separate supply connectors for the CPU(s), this approach facilitates distinguishing CPU power consumption from that of other motherboard components. Again, this improvement comes with increased cost and complexity: the sophisticated DAQ unit is priced an order of magnitude higher than the power meter, and we had to build a custom board to house the current transducer infrastructure.

2.2.3 At the Processor Voltage Regulator

Although measurements taken at the motherboard supply rails factor out the power supply unit's efficiency curve, they are still affected by the efficiency curve of the on-board voltage regulators. To eliminate this source of inaccuracy, we investigate a third approach. Motherboards that follow Intel's processor power delivery guidelines (Voltage Regulator-Down (VRD) 11.1 [55]) provide a load indicator output (IMON/Iout) from

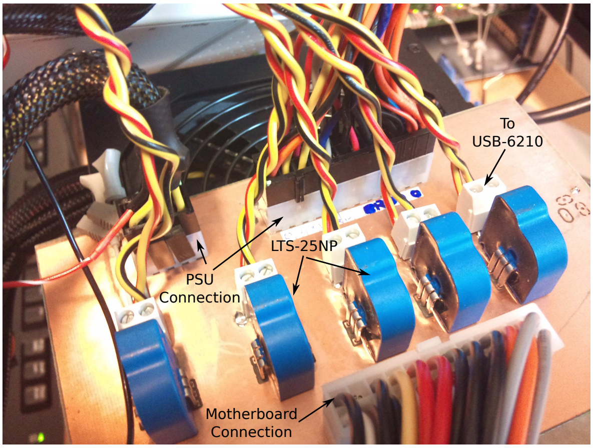


Figure 2.3: *Our Custom Measurement Board*

the processor voltage regulator. This load indicator is connected to the processor for use by the processor's power management features. This signal provides an analog voltage linearly proportional to the total load current of the processor. We use this current sensing pin from the processor's voltage regulator chip (CHL8316, in our case) to acquire real-time information about total current delivered to the processor. We also use the voltage output at the V_CPU pin of the voltage regulator, which is directly connected to the CPU voltage supply input of the processor. We locate these two signals on the motherboard and solder wires at the respective connection points (the resistor/capacitor pads connected to these signals). We connect these two signals and the ground point to our DAQ unit, logging the values read on the separate test machine. This current measurement setup is shown in Fig. 2.4.

The full voltage swing of the IMON output is 900mV for the full-scale current of 140A (for the motherboard under test). Hence, the current sensitivity of the IMON output comes to about 6.42mV/A. The theoretical sensitivity of this infrastructure depends on the voltage sensitivity of the DAQ unit (47μV) and its overall sensitivity to current changes comes to 7mA. This sensitivity is less than that for measuring current at the ATX power rails, but the sensitivity may vary for different voltage regulators on different motherboards. This method provides the most accurate measurements of absolute current feeding the processor, but it is also the most intrusive, as it requires soldering wires on the motherboard, an invasive instrumentation procedure that should only be performed by skilled technicians. Moreover, these power measurements are limited to processor power consumption (we get no information about other system components). For example, for memory-intensive applications, we can account for power consumption

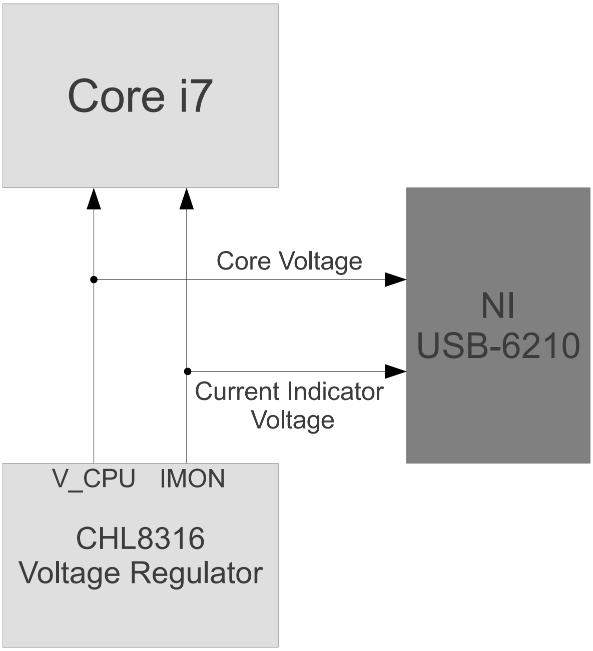


Figure 2.4: *Measurement Setup on CPU Voltage Regulator*

effects of the external bus transactions triggered by off-chip memory accesses, but this method provides no means of measuring power consumed in the DRAMs. The accuracy of the IMON output is specified by the CHL8316 datasheet to be within $\pm 7\%$. This falls far below the 0.7% accuracy of the current transducers at the ATX power rails[1].

2.2.4 Experimental Results

We use an Intel Core$^{\mathrm{TM}}$i7 870 processor to compare power measurement readings at the wall outlet, at the ATX power rails, and directly on the processor's voltage regulator.

The *Watts Up? Pro* measures power consumption of the entire system at the rate of 1 Sa/s, whereas the data acquisition unit is configured to capture samples at the rate of 40000 Sa/s from the four effective ATX voltage rails (+12V1/2, +12V3, +5V and +3.3V) and the V_CPU and the IMON outputs of the CPU voltage regulator. We choose this rate because the combined sampling rate of the six channels adds up to 240K Sa/s, and the maximum sampling rate supported by the DAQ is 250K Sa/s. To remove background

[1] The accuracy specifications of the processor's voltage regulator may differ for different manufacturers.

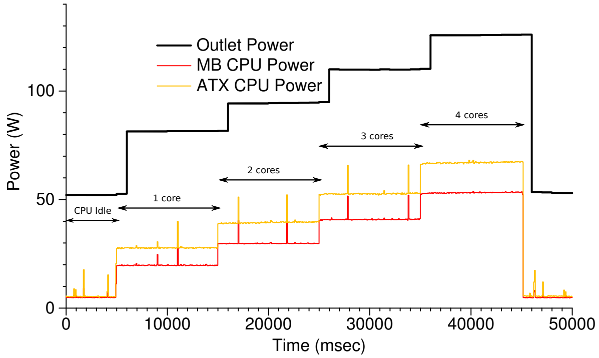


Figure 2.5: *Power Measurement Comparison When Varying the Number of Active Cores*

noise, we average the DAQ samples over a period of 40 samples, which effectively gives 1000 Sa/s. We use a CPU-bound test workload consisting of a 32×32 matrix multiplication in an infinite loop.

Coarse-grain Power Variance. Fig. 2.5 shows power measurement results across the three different points as we vary the number of active cores. Steps in the power consumption are captured by all measurement setups. The low sampling frequency of the wall-socket power meter prevents it from capturing short and sharp peaks in power (probably caused by background OS activity). The power consumption changes we observe at the wall outlet are at least 13 watts from one activity level to another. At such coarse-grained power variation, the power readings at wall outlet are strongly correlated with power readings at the ATX power rails and CPU voltage regulator.

Fine-grain Power Variance. Fig. 2.6 depicts measurement results when we vary CPU frequency every five seconds from 2.93 GHz to 1.33 GHz in steps of 0.133 GHz. The power measurement setup at the ATX power rails and the CPU voltage regulator capture the changes in power consumption accurately, and apart from the differences in absolute values and the effects of the CPU voltage regulator efficiency curve, there is not much to differentiate measurements at the two points. However, the power measurements taken by the power meter at the wall outlet fail to capture the changes faithfully, even though its one-second sampling rate is enough to capture steps that last five seconds. This effect is even more visible when we introduce throttling (at eight different levels for each CPU frequency), as shown in Fig. 2.7. Here, each combination of CPU frequency and throttling level lasts for two seconds, which should be long enough for the power meter to capture steps in the power consumption. But the power meter performs worse as power consumption decreases. We suspect that this effect is due to the AC to DC conversion circuit of the power supply unit, probably due to the smoothing effect of

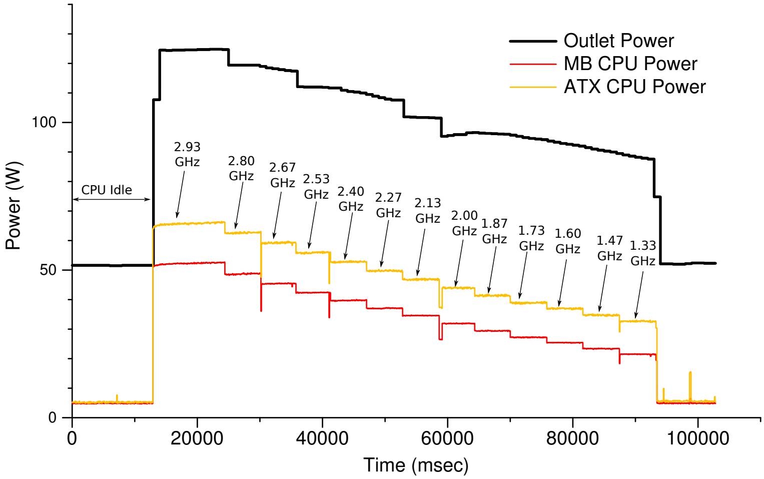


Figure 2.6: *Power Measurement Comparison When Varying Core Frequency*

the capacitor in the PSU. These effects are not visible between measurement points at the ATX power rails and CPU voltage regulator.

Fig. 2.8 shows the efficiency curve of the CPU voltage regulator at various load levels. The voltage regulator on the test system employs dynamic phase control to adjust the number of phases with varying load current to try to optimize the efficiency over a wide range of loads. The voltage regulator switches to one-phase or two-phase operation to increase the efficiency at light loads. When the load increases, the regulator switches to four-phase operation at medium loads and six-phase operation at high loads. The sharp change in efficiency visible in Fig. 2.8 is likely due to adaptation in phase control. Fig. 2.9 shows the obtained efficiency curve of the cabinet PSU against total power consumption calculated on the ATX power rails. The total system power never goes below 30W, and the efficiency of the PSU varies from 60% to around 80% in the output power range from 30W to 100W.

Fig. 2.10 shows the changes in CPU and main memory power consumption while running *gcc* from SPEC CPU2006. Power consumption of the main memory varies from around 7.5W to 22W across various phases of the *gcc* run. Researchers and practitioners who wish to assess main memory power consumption will at least want to measure power at the ATX power rails.

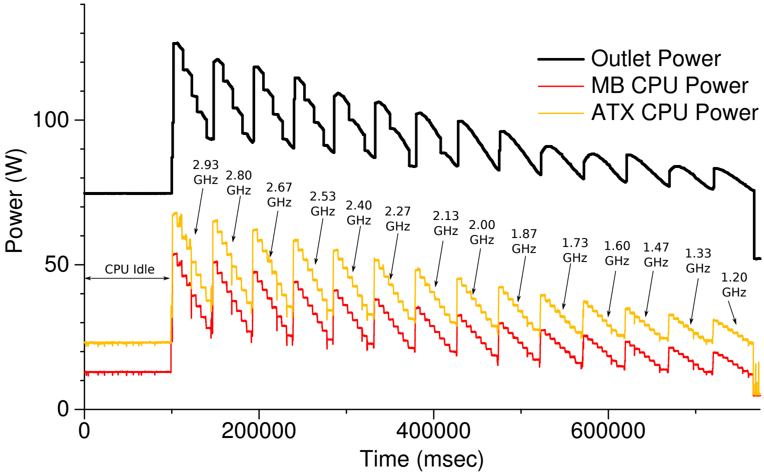


Figure 2.7: *Power Measurement Comparison When Varying Core Frequency Together with Throttling Level*

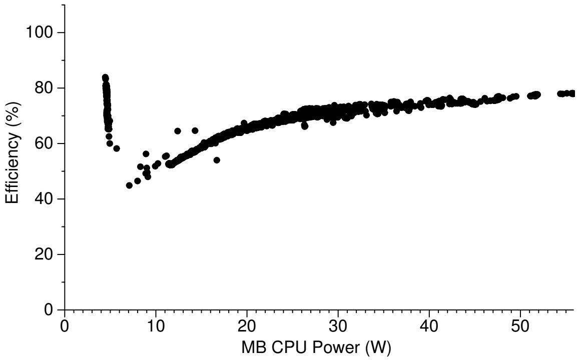


Figure 2.8: *Efficiency Curve of CPU Voltage Regulator*

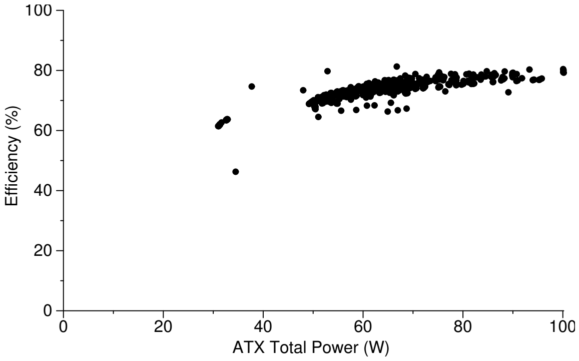


Figure 2.9: *Efficiency Curve of the PSU*

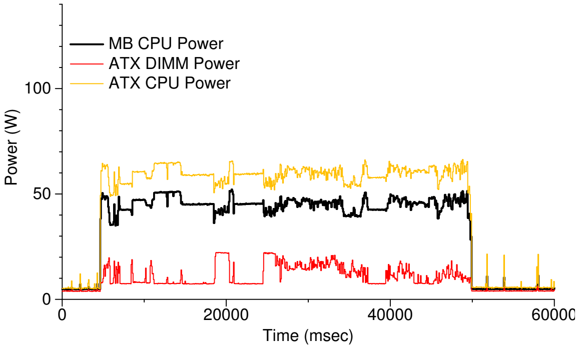


Figure 2.10: *Power Measurement Comparison for the CPU and DIMM (Running gcc)*

2.3 RAPL power estimations

2.3.1 Overview

Intel introduced Running Average Power Limit (RAPL) interface on the Sandy Bridge microarchitecture. The programmers can use the RAPL interface for enforcing power consumption constraint on the processor. This interface includes non-architectural Model-specific registers (MSRs) for setting the power limit and reading the energy consumption status of the processor power domains like processor die and DRAM[2]. The energy consumption information provided by the RAPL interface is not based on actual current measurement from physical sensors, but a power model based on performance events and probably other inputs like temperature, voltage, etc. As per Intel specifications, the RAPL energy status counter is updated *approximately* every 1ms, giving a maximum sampling frequency of 1000 Sa/s. Since Intel has not specified the details of the power model, the only way to measure the accuracy of the model is through experimentation. In the following sections, we compare power measurement readings from RAPL and the ATX power rails.

2.3.2 Experimental Results

We use Intel Core$^{\text{TM}}$i7 4770 processor to compare power measurement results at the ATX power rails and Intel's RAPL energy counter. For reading the RAPL energy counter, we use the x86-MSR driver to read the RAPL energy counter MSR called `MSR\_PKG\_ENERGY\_STATUS`. We set up a real-time timer that raises `SIGALRM` at configured intervals to read the MSR. The ATX power measurement infrastructure is the same as described for Intel Core$^{\text{TM}}$i7 870 processor. Below we describe our experiments to validate the RAPL energy counter readings.

RAPL Overhead. One of the major differences between measuring power using RAPL and other techniques shown in Figure 2.1 is that the RAPL measurements are done on the same machine that is being tested. As a result, reading the RAPL energy counter at frequent intervals adds some overhead to the system power. To test this, we run our RAPL counter reading tool at the sampling frequency of 1 Sa/s, 10 Sa/s and 1000 Sa/s and monitor the change in system power consumption on the ATX CPU power rails. The results from this experiment are shown in Fig. 2.11. The spikes in power consumption visible in the figure indicate the starting of the RAPL reading utility. These spikes mainly result from loading dynamic libraries and can be ignored, assuming that the RAPL utility is started before running any benchmark under test. These

[2] The DRAM energy status counter is only available on server platforms.

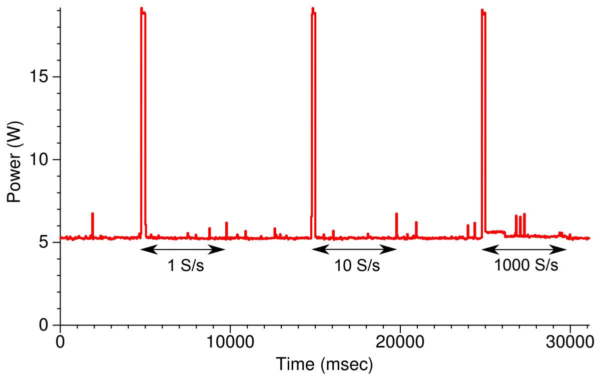


Figure 2.11: *Power Overhead Incurred While Reading the RAPL Energy Counter*

spikes however act as useful synchronization points to overlap readings from RAPL and the ATX power rails. For calculating the RAPL power overhead, we concentrate on the CPU power consumption after the initial power spike. As seen from the figure, reading the RAPL MSR every second and every 100ms adds no discernible power consumption to the idle system power, but reading the RAPL counter every 1ms adds 0.1W of power overhead. However, this small power overhead comes mainly from generating `SIGALRM` every millisecond and not from reading the MSR. Hence, if the reading of RAPL energy counter is done as part of the existing software infrastructure like a kernel scheduler, this will not add discernible overhead to CPU power consumption.

RAPL Resolution. As per the Intel specification manual [56], the RAPL energy counter is updated about every 1ms. To test this update frequency, we run the matrix multiplication application described in Section 2.2.4 while reading the RAPL energy status counter every 1ms. We sample the ATX power rails every 10μS and average over 100 samples. The results from this experiment are shown in Fig. 2.12. Although the RAPL readings follow the same curve as those from the ATX power rails, two things are of note here. First, the RAPL readings show more deviation from the mean than the ATX readings. Second, there are huge deviations from the mean at periodic intervals of around 1.6-1.7 seconds. Hähnel et al. [48] observe that the RAPL counter is not updated exactly at 1ms. As per their experiments, there can be a deviation of a few tens of thousands of cycles in the periodic interval at which the RAPL counter is updated on a given platform. This explains the small deviations from the mean we see in our RAPL readings. They also observe that when the CPU switches to System Management Mode (SMM), the RAPL update is delayed until after the CPU comes out of SMM mode. This results in the RAPL energy counter showing no increment between two successive

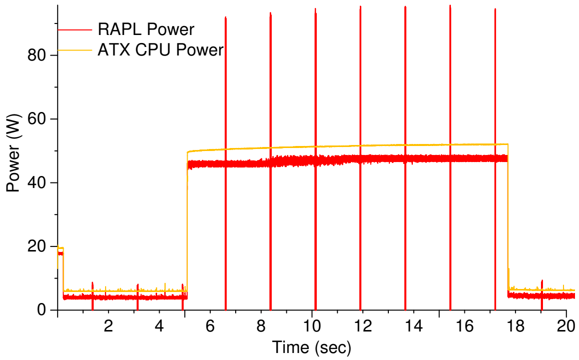


Figure 2.12: *Power Measurement Comparison for the ATX and RAPL at the RAPL Sampling Rate of 1000 Sa/s*

readings, creating a negative deviation. The next update to the counter increments the energy value expended by the processor in last 2ms instead of 1ms, creating a positive deviation. On our system, the CPU switches to SMM every 1.6-1.7 seconds, which causes inaccurate energy readings.

RAPL Accuracy. To test the accuracy of the RAPL readings, we first repeat the DVFS test from Fig. 2.6 in Section 2.2.4 and take the power measurement readings from the ATX power rail and RAPL. Each DVFS operating point is maintained for five seconds. The ATX power is sampled at 10000 Sa/s while the RAPL counter is sampled at 10 Sa/s. The results from this test are shown in Fig. 2.13. The RAPL measurement curve follows the ATX measurement curve faithfully, although their y-axis scales are necessarily different. This test shows that the RAPL model works well across different DVFS frequencies.

Next, we simulate a real-time application in which a task is started at fixed periodic intervals. We compare power measurement from RAPL and ATX while varying the task period. We set up a timer to raise `SIGALRM` signal, and a matrix multiplication task is started at every timer interrupt. The matrix multiplication loop is configured to occupy about 66% of the time period. We run this experiment for three different periods: 100ms, 10ms, and 1ms. We gather samples between two consecutive SMM switches. We read the RAPL counter every 1ms. We sample the ATX power rails every 5μS and average over 20 samples. The results from this experiment are shown in Fig. 2.14. For the 100ms and 10ms period, the RAPL readings closely follow the ATX readings, although they are off by 1ms when the power consumption changes suddenly. As expected, for the 1ms period test, RAPL is unable to capture the power consumption curve but accurately estimates the average power consumption.

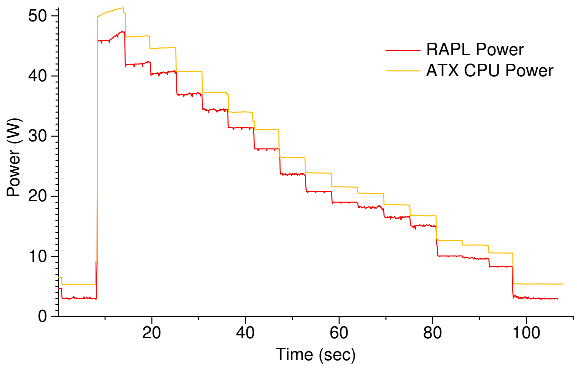
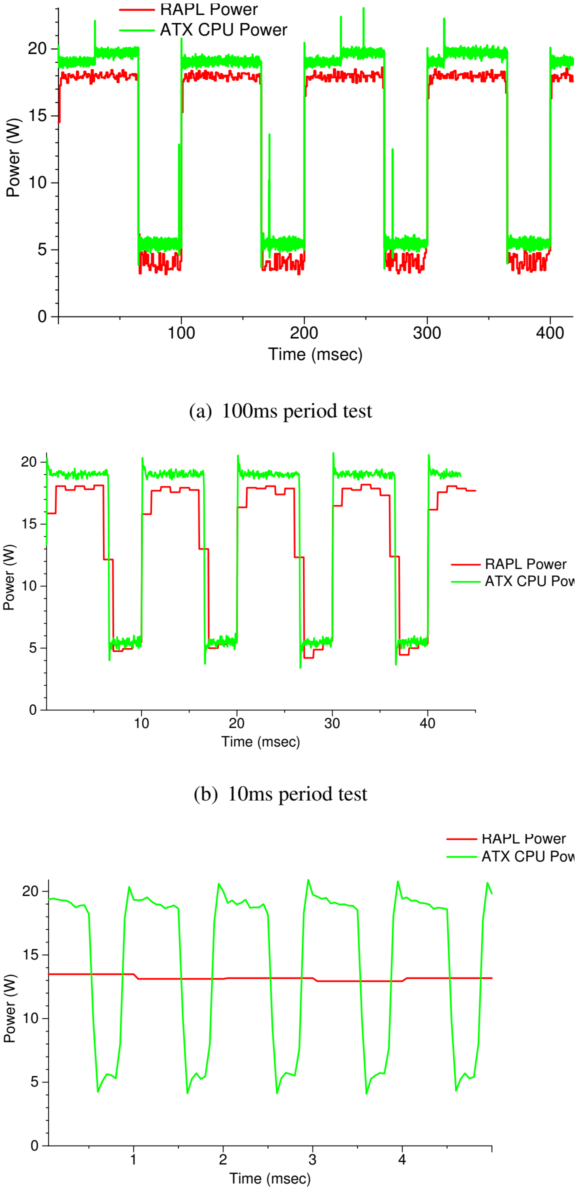


Figure 2.13: *Power Measurement Comparison for ATX and RAPL When Varying Core Frequency*

Since the RAPL energy counter values are based on a performance event based power model [6], we also test the RAPL accuracy using a microbenchmark that runs integer operations, floating-point operations and memory operations in different phases. For this experiment, we sample the RAPL energy counter at 100Sa/S. We sample the ATX CPU power rails at 10000 Sa/s and then average over 100 samples. We also sample the ATX 5V power rails, which on our system mainly supplies the DRAM memory. The results are shown in Fig. 2.15. The deviations that we see in Fig. 2.12 due to CPU switching to SMM are visible in this Fig. 2.15 as well. Because we sample the RAPL counter every 10ms instead of 1ms, we only see 10% deviation from the mean instead of 100%. Apart from these deviations that occur every 1.7 second (effectively one out of 170 samples), the RAPL energy readings are accurate for all of the integer, floating-point and memory-intensive phases. To quantify the RAPL power estimation accuracy, we calculate Spearman's rank correlation coefficient for RAPL and ATX CPU readings for the microbenchmark run. The correlation coefficient is ρ=0.9858. In comparison, the correlation coefficient between the ATX CPU power and motherboard CPU power shown in Fig. 2.7 is ρ=0.9961. Sample values from the ATX show that DRAM power is significant, which is not captured by the RAPL package energy counter. Power-aware computing methodologies must take this into consideration. The RAPL interfaces on Intel processors for server platforms include an energy counter for the *DRAM domain* called `MSR\_DRAM\_ENERGY\_STATUS`. This can be useful for assessing the power consumption of the DRAM memory.

Based upon our experimental results, we conclude that the RAPL energy counter is good for estimating energy for a duration of more than 1ms. However, when the researcher is interested in instantaneous power trends, especially peak power trends,



(a) 100ms period test

(b) 10ms period test

(c) 1ms period test

Figure 2.14: *Power Measurement Comparison for ATX and RAPL with Varying Real-time Application Period*

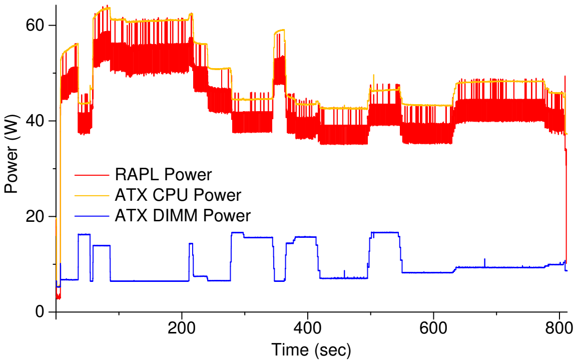


Figure 2.15: *Accuracy Test of RAPL for Custom Microbenchmark*

sampling the actual power consumption using an accurate measurement infrastructure (like ours) is a better choice.

2.4 Related Work

Hackenberg et al. [57] compare several measurement techniques for signal quality, accuracy, timing resolution, and overhead. They use both Intel and AMD systems in their study. Their experimental results complement our results. There have been many interesting studies on power-modeling and power-aware resource management. These employ various means to measure empirical power. Rajamani et al. [11] use on-board sense resistors located between the processor and voltage regulators to measure power consumed by the processor. They use a National Instruments isolation amplifier and data acquisition unit to filter, amplify, and digitize their measurements. Isci and Martonosi [58] measure current on the 12V ATX power lines using clamp ammeters, that are hooked to a digital multimeter (DMM) for data collection. The DMM is connected to a data logging machine via an RS232 serial port. Contreras and Martonosi [14] use jumpers on their Intel XScale^{TM} development board to measure the power consumption of the CPU and memory separately. They feed the measurements to a LeCroy oscilloscope for sampling. Cui et al. [59] also measure the power consumption at the ATX power rails. They use current-sense resistors and amplifiers to generate sense voltages (instead of using current transducers), and they log their measurements using a digital multimeter. Bedard et al. [51] build their own hardware that combines the voltage and current measurements and host interface into one solution. They use an Analog Devices ADM1191 digital power monitor to sense voltage and current values and an Atmel$^{®}$ microcontroller to send the measured values to a host USB port. Hähnel et al. [48] experiment with the update granularity of the RAPL interface and report practical considerations for using the RAPL energy counter for power measurement. We find their results useful in explaining our experiments.

2.5 Conclusion

In this chapter, we demonstrate different techniques that can be employed to measure power consumption of the full system, CPU and DRAM memory. We compare the different techniques in terms of accuracy, level sensitivity and temporal granularity and discuss their advantages and disadvantages. We test Intel's RAPL energy counter results for overhead, accuracy, and update granularity. We conclude that the RAPL counter for reading package energy incurs almost no power overhead and is fairly accurate for estimating energy over periods of more than few tens of milliseconds. However, because of irregularities in the frequency at which the RAPL counter is updated, it shows deviations from the average power when it is sampled at a faster rate.

3 Per-core Power Estimation Model

3.1 Overview

Power measurement techniques described in Chapter 2 are essential for analyzing the power consumption of systems under test. However, these measurement techniques do not provide detailed information on the power consumption of individual processor cores or microarchitectural units (e.g., caches, floating point units, integer execution units). To develop resource-management mechanisms for an individual processor, system designers need to analyze power consumption at the granularity of processor cores or even the components within a processor core. This information can be provided by placing on-die digital power meters, but that increases the chip's hardware cost. Hence, support for such fine-grained digital power meters is limited by chip manufacturers. Even when measurement facilities exist — e.g., the Intel Core$^{\text{TM}}$i7-870 [60] features per-core power monitoring at the chip-level — they are rarely exposed to the user. Indeed, power sensing, actuation, and management support is more often implemented at the blade level with a separate computer monitoring output [61, 62].

An alternative to hardware support for fine-grained digital power meters is to estimate the power consumption at the desired granularity using software power models.

Such models can identify various power-relevant events in the targeted microarchitecture and then track those events to generate a representative power consumption value. We can characterize software power estimation models by the following attributes:

- **Portability.** The model should be easy to port from one platform to another;
- **Scalability.** The model should be easy to scale across varying number of active cores and across different CPU voltage-frequency points;
- **CPU Usage.** The model's CPU footprint should be negligible, so as not to pollute the power consumption values of the system under test;
- **Accuracy.** The model's estimated values should closely follow the empirically measured power of the device that is modeled;
- **Granularity.** The model should provide power consumption estimates at the granularity desired for the problem description (per core, per microarchitectural module, etc.); and
- **Speed.** The model should supply power estimation values to the software at minimal latency (preferably within microseconds).

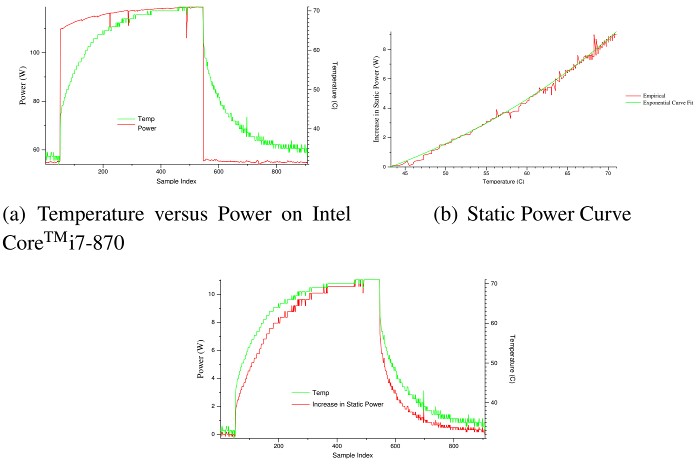
In this thesis, we develop a power model that uses performance counters and temperature to generate accurate per-core power estimates in real time, with no off-line profiling or tracing. We build upon the power model developed by Singh et al. [16, 63] and include some of their data for comparison. We validate their model on AMD K8 and Intel Core$^{\text{TM}}$i7 architectures. Below we explain our choice of using performance counters and temperature to develop the model.

Performance Counters. We use performance counters for model formation because the power models need a mechanism to track CPU activities with low overhead. Most modern processors are equipped with a Performance Monitoring Unit (PMU) providing the ability to count the microarchitectural events of the processor. PMCs are available individually for each core and hence can be used to create core-specific models. This allows programmers to analyze core performance, including the interaction between programs and the microarchitecture, in real time on real hardware, rather than relying on simplified and less accurate performance results from simulations. The PMUs provide a wide variety of performance events. These events can be counted by mapping them to a limited set of Performance Monitoring Counter (PMC) registers. For example, on Intel and AMD platforms, these performance counter registers are accessible as Model Specific Registers (MSRs). Also called Machine Specific Registers, these are not necessarily compatible across processor families. Software can configure the performance counters to select which events to count. The PMCs can be used to count events like cache misses, micro-operations retired, stalls at various stages of an out-of-order pipeline, floating point/memory/branch operations executed, and many more. Although, the counter values are not error-free [64,65] or even deterministic [66], if used correctly, the errors are small enough to make PMCs suitable candidates for estimating power consumption. The number and variety of performance monitoring events available for modern processors are increasing with each new architecture. For example, the number of traceable performance events available in the Intel Core$^{\text{TM}}$i7-870 processor is about ten times the number available in the Intel Core Duo processor [67]. This comprehensive coverage of event information increases the probability that the available performance monitoring events will be representative of overall microarchitectural activity for the purposes of performance and power analysis. This makes the use of performance counters to develop power models for hardware platforms very popular among researchers.

Temperature. Processor power consumption consists of both dynamic and static elements. Among these, the static power consumption is dependent on the core temperature. Eq. 3.1 shows that the static power consumption of a processor is a function of both leakage current and supply voltage. The processor leakage current is a summation of subthreshold leakage and gate-oxide leakage current: $I_{leakage} = I_{sub} + I_{ox}$ [68]. The subthreshold leakage current can be derived using Eq. 3.2 [68]. The V_θ component in the equation is thermal voltage, and it increases linearly with temperature. Since V_θ is in the exponents, subthreshold leakage current has an exponential dependence on temperature. With the increase in processor power consumption, processor temperature increases. This increase in temperature increases leakage current, which, in turn, increases static power consumption. To study the effects of temperature on power consumption, we ran a multithreaded program executing *MOV* operations in a loop on our Intel Core$^{\text{TM}}$i7-870 machine. The rate of instructions retired, instructions executed, pipeline stalls and memory operations remains constant over the entire run of the program. We also keep the CPU operating frequency constant and observe that CPU voltage remains constant as well. This indicates that the dynamic power consumption of the processor does not change over the run of the program. Fig. 3.1(a) shows that the total power consumption of the machine increases during the program's runtime, and it coincides with the increase in chip temperature. The total power consumption increases by almost 10%. To account for this increase in static power, it is necessary to include the temperature in power models.

$$P_{static} = \sum I_{leakage} * V_{core} \tag{3.1}$$

$$I_{sub} = K_1 W e^{-V_{th}/nV_\theta}(1 - e^{-V/V_\theta}) \tag{3.2}$$



(a) Temperature versus Power on Intel Core^{TM}i7-870

(b) Static Power Curve

(c) Temperature versus Static Power on Intel Core^{TM}i7-870

Figure 3.1: *Temperature Effects on Power Consumption*

where $I_{sub} = Subthreshold\ leakage\ current$

$W = Gate\ width$

$V_\theta = Thermal\ voltage$

$V_{th} = Threshold\ voltage$

$V = Supply\ voltage$

$K_1 = Experimentally\ derived\ constant$

$W = Experimentally\ derived\ constant$

As per Eq. 3.1, the static power consumption increases exponentially with temperature. We confirm this empirically by plotting the net increase in power consumption when the program executes at the higher temperature, as shown in the Fig. 3.1(b). The non-regression analysis gives us Eq. 3.3 and the curve fit shown in the Fig. 3.1(b) closely follows the empirical data points with determination coefficient $R^2 = 0.995$.

$$P_{staticInc} = 1.4356 \times 1.034^T, \quad \text{when } V_{core} = 1.09V \tag{3.3}$$

Plotting these estimates of the increments in static power consumption, as in Fig. 3.1(c), explains the gradual rise in total power consumption when the dynamic behavior of a program remains constant. Instead of using a non-linear function, we approximate the static power increase as a linear function of temperature. This is a fair approximation considering that the non-linear equation given in Eq. 3.3 can be closely approximated with the linear equation given in Eq. 3.4 with determination coefficient $R^2 = 0.989$ (for the range in which die temperature changes occur). This linear approximation avoids the added cost of introducing an exponential term in the power model computation.

$$P_{staticInc} = 0.359 \times T - 16.566, \quad \text{when } V_{core} = 1.09V \tag{3.4}$$

Modern processors allow programmers to read temperature information for each core from on-die thermal diodes. For example, Intel platforms report relative core temperatures on-die via Digital Thermal Sensors (DTS), which can be read by software through the MSRs or the Platform Environment Control Interface (PECI) [69]. This data is used by the system to regulate CPU fan speed or to throttle the processor in case of overheating. Third party tools like *RealTemp* and *CoreTemp* on Windows and open-source software like *lm-sensors* on Linux can be used to read data from the thermal sensors. As Intel documents indicate [69], the accuracy of temperature readings provided by the thermal sensors varies, and the values reported may not always match the actual core temperatures. Because of factory variation and individual DTS calibration, reading accuracy varies from chip to chip. The DTS equipment also suffers from slope errors, which means that temperature readings are more accurate near the T-junction max (the maximum temperature that cores can reach before thermal throttling is activated) than at lower temperatures. DTS circuits are designed to be read over reasonable operating temperature ranges, and the readings may not show lower values than 20°C even if the actual core temperature is lower. Since DTS is primarily created as a thermal protection mechanism, reasonable accuracy at high temperatures is acceptable. This affects the accuracy of power models that use core temperature. Researchers and practitioners should read the processor model datasheet, design guidelines, and errata to understand the limitations of their respective thermal monitoring circuits, and they should take corrective measures when deriving their power models, if required.

3.2 Modeling Approach

Our approach to power modeling is workload-independent and does not require application modification. To show the effectiveness of our models, we perform two types of studies:

- We demonstrate accuracy and portability on five CMP platforms;

- We use the model in a power-aware scheduler to maintain a power budget

Studying sources of model error highlights the need for better hardware support for power-aware resource management, such as fine-grained power sensors across the chip and more accurate temperature information. Our approach nonetheless shows promise for balancing performance, power, and thermal requirements for platforms from embedded real-time systems to consolidated data centers, and even to supercomputers.

In the rest of the chapter, we first present the details of how we build our model, and then we discuss how we validate them. Our evaluation analyzes the model's computation overhead(Section 3.4.1) and accuracy(Section 3.4.2). We then explain the meta-scheduler that we use as a proof-of-concept for showing the effectiveness of our model in Section 3.5.

3.3 Methodology

3.3.1 Counter Selection

Selecting appropriate PMCs to use is extremely important with respect to accuracy of the power model. We choose counters that are most highly correlated with measured power consumption. The chosen counters must also cover a sufficiently large set of events to ensure that they capture general application activity. If the chosen counters do not meet these criteria, the model will be prone to error. The problem of choosing appropriate counters for power modeling has been handled in different ways by previous researchers.

Research studies that estimate power for an entire core or a processor [11, 14, 15], use a small number of PMCs. Those that aim to construct decomposed power models to estimate the power consumption of sub-units of a core [10, 12, 13] tend to monitor more PMCs. The number of counters needed depends on the model granularity and the acceptable level of complexity. Also, most modern processors allow simultaneous counting of only a limited number (two/four/eight) microarchitectural events. Hence, using more counters in the model requires interleaving the counting of events and extrapolating the counter values over the total sampling period. This reduces the accuracy of absolute counter values but allows researchers to track more counters.

To choose appropriate performance counters for developing our power-estimation model, we divide the available counters into four categories and then choose one counter from each category based upon statistical correlation [16, 63]. This ensures that the chosen counters are comprehensive representations of the entire microarchitecture and are not biased towards any particular section. Caches and floating point units form a large part of the chip real estate, and thus PMCs that keep track of their activity factors would be useful additions to the total power consumption information. Depending on the platform, multiple counters will be available in both these categories. For example, we can count the total number of cache references as well as the number of cache misses for various cache levels. For floating point operations, depending upon the processor model, we can count (separately or in combination) the number of multiplication, addition, and division operations. Because of the deep pipelining of modern processors, we can also expect out-of-order logic to account for a significant amount of power. Stalls due to branch mispredictions or an empty instruction decoder may reduce average power consumption over a fixed period of time. On the other hand, pipeline stalls caused by full reservation stations and reorder buffers will be positively correlated with power because these indicate that the processor has extracted enough instruction-level parallelism to keep the execution units busy. Hence, pipeline stalls indicate not just the power usage of out-of-order logic but of the executions units, as well. In addition, we would like to use a counter that can cover all the microarchitectural components not covered by the above three categories. This includes, for example, integer execution units, branch prediction units, and Single Instruction Multiple Data (SIMD) units. These events can be monitored using the specific PMCs tied to them or by a generalized counter like total instructions/micro-operations (UOPS) retired/executed/issued/decoded. To construct a power model for individual sub-units, we need to identify the respective PMCs that represent each sub-unit's utilization factors.

To choose counters via statistical correlation, we run a training application while sampling the performance counters and collecting empirical power measurement values. Fig. 3.2 shows simplified pseudo-code for the microbenchmark [16] that we use to develop our power model. Here, different phases of the microbenchmark exercise different parts of the microarchitecture. Since the number of relevant PMCs will most likely be more than the limit on simultaneously monitored counters, multiple training runs are required to gather data for all desired counters.

Once the performance counter values and the respective empirical power consumption values are collected, we use a statistical correlation to establish the correlation between performance events (counter values normalized to the number of instructions executed) and power. This guides our selection of the most suitable events for use in the power model. Obviously, the type of correlation method used can affect the model accuracy. We use Spearman's rank correlation [70] to measure the relationship between each counter and power. The performance counters and power values can be linear or non-linear. Using the rank correlation, as opposed to using correlation methods like Pearson's, ensures that this non-linear relationship does not affect the correlation coefficient.

For example, Table 3.1 shows the most power-relevant counters divided categori-

```
for (i=0;i<interval*PHASE_CNT;i++) {
        phase = (i/interval) % PHASE_CNT;
        switch(phase) {
                 case 0:
                        /* do floating point operations */
                  case 1:
                        /* do integer arithmetic operations */
                  case 2:
                        /* do memory operations with high
                             locality */
                  case 3:
                        /* do memory operations with low locality
                              */
                  case 4:
                        /* do register file operations */
                  case 5:
                        /* do nothing */
                  .
                  .
                  .
        }
}
```

Figure 3.2: *Microbenchmark Pseudo-Code*

(a) FP Operations

Counters	ρ
FP_COMP_OPS_EXE:X87	0.65
FP_COMP_OPS_EXE:SSE_FP	0.04

(b) Total Instructions

Counters	ρ
UOPS_EXECUTED:PORT1	0.84
UOPS_ISSUED:ANY	0.81
UOPS_EXECUTED:PORT015	0.81
INSTRUCTIONS_RETIRED	0.81
UOPS_EXECUTED:PORT0	0.81
UOPS_RETIRED:ANY	0.78

(c) Memory Operations

Counters	ρ
MEM_INST_RETIRED:LOADS	0.81
UOPS_EXECUTED:PORT2_CORE	0.81
UOPS_EXECUTED:PORT234_CORE	0.74
MEM_INST_RETIRED:STORES	0.74
LAST_LEVEL_CACHE_MISSES	0.41
LAST_LEVEL_CACHE_REFERENCES	0.36

(d) Stalls

Counters	ρ
ILD_STALL:ANY	0.45
RESOURCE_STALLS:ANY	0.44
RAT_STALLS:ANY	0.40
UOPS_DECODED:STALL_CYCLES	0.25

Table 3.1: *Intel Core^{TM}i7-870 Counter Correlation*

cally according to the correlation coefficients obtained from running the microbenchmark on the Intel Core^{TM}i7-870 platform. Table 3.1(a) shows that only `FP_COMP_OPS_EXE:X87` is a suitable candidate from the floating point (FP) category. Ideally, to get total FP operations executed, we should count both x87 FP operations(`FP_COMP_OPS_EXE:X87`) and SIMD (`FP_COMP_OPS_EXE:SSE_FP`) operations. The microbenchmark does not use SIMD floating point operations, and hence we see high correlation for the x87 counter but not for the SSE (Streaming SIMD Extensions) counter. Because of the limit on the number of counters that can be sampled simultaneously, we have to choose between the two. Ideally, chip manufacturers would provide a counter reflecting both x87 and SSE FP instructions, obviating the need to choose one. In Table 3.1(b), the correlation values in the total instructions category are almost equal, and thus these counters need further analysis. The same is true for the top three counters in the stalls category, shown in Table 3.1(d). Since we are looking for counters providing insight into out-of-order logic usage, the `RESOURCE_STALLS:ANY` counter is our best option. As for memory operations, choosing either `MEM_INST_RETIRED:LOADS` or `MEM_INST_RETIRED:STORES` will bias the model towards load- or store-intensive applications. Similarly, choosing `UOPS_EXECUTED:PORT1` or `UOPS_EXECUTED:PORT0` in the total instructions category will bias the model towards addition- or multiplication-intensive applications. We therefore omit these counters from further consideration.

Table 3.1 shows that correlation analysis may find counters from the same category with very similar correlation numbers. Our aim is to make a comprehensive power model using only four counters, and thus we must make sure that the counters chosen convey as little redundant information as possible. We therefore analyze the correlation among

(a) MEM versus INSTR Correlation

	UOPS_EXECUTED:PORT234	LAST_LEVEL_CACHE_MISSES
UOPS_ISSUED:ANY	0.97	0.14
UOPS_EXECUTED:PORT015	0.88	0.2
INSTRUCTIONS_RETIRED	0.91	0.12
UOPS_RETIRED:ANY	0.98	0.08

(b) FP versus INSTR Correlation

	FP_COMP_OPS_EXE:X87
UOPS_ISSUED:ANY	0.44
UOPS_EXECUTED:PORT015	0.41
INSTRUCTIONS_RETIRED	0.49
UOPS_RETIRED:ANY	0.43

(c) STALL versus INSTR Correlation

	RESOURCE_STALLS:ANY
UOPS_ISSUED:ANY	0.25
UOPS_EXECUTED:PORT015	0.30
INSTRUCTIONS_RETIRED	0.23
UOPS_RETIRED:ANY	0.21

Table 3.2: *Counter-Counter Correlation*

all the counters. To select a counter from the memory operations category, we analyze the correlation of UOPS_EXECUTED:PORT234_CORE and LAST_LEVEL_CACHE_MISSES with the counters from the total instructions category, as shown in Table 3.2(a). From this table, it is evident that UOPS_EXECUTED:PORT234_CORE is highly correlated with the instructions counters, and hence LAST_LEVEL_CACHE_MISSES is the better choice. To choose a counter from the total-instructions category, we analyze the correlation of these counters with the FP and stalls counters (in Table 3.2(b) and Table 3.2(c), respectively). These correlations do not clearly recommend any particular choice. In such cases, we can either choose one counter arbitrarily or choose a counter intuitively. UOPS_EXECUTED:PORT015 does not cover memory operations that are satisfied by cache accesses, instead of main memory. The UOPS_RETIRED:ANY and INSTRUCTIONS_RETIRED counters cover only retired instructions and not those that are executed but not retired, e.g., due to branch mispredictions. A UOPS_EXECUTED:ANY counter would be appropriate, but since such a counter does not exist, the next best option is UOPS_ISSUED:ANY. This counter covers all instructions issued, so it also covers the instructions issued but not executed (and thus not retired).

We use the same methodology to select representative performance counters for all the machines that we evaluate. Table 3.4 shows the counters we select for the different platforms.

3.3.2 Model Formation

Having identified events that contribute significantly to consumed power, we create a formalism to map observed microarchitectural activity and measured temperatures to measured power draw. We re-run the microbenchmark sampling just the chosen PMCs, collecting power and temperature data at each sampling interval. We normalize each time-sampled PMC value, e_i, to the elapsed cycle count to generate an *event rate*, r_i.

We then map rise in core temperature, T, and the observed event rates, r_i, to core power, P_{core}, via a piece-wise model based on multiple linear regression, as in Equation 3.5. We apply non-linear transformations to normalized counter values to account for non-linearity, as in Equation 3.6. The normalization ensures that changing the sampling period of the readings does not affect the weights of the respective predictors. We develop a piecewise power model that achieves better fit by separating the collected samples into two bins based on the values of either the memory counter or the FP counter. Breaking the data using the memory counter value helps in separating memory-bound phases from CPU-bound phases. Using the FP counter instead of the memory counter to divide the data helps in separating FP-intensive phases. The selection of a candidate for breaking the model is machine-specific and depends on what gives a better fit. Regardless, we agree with Singh et. [16, 63] that piecewise linear models better capture processor behavior.

$$\hat{P}_{core} = \begin{cases} F_1(g_1(r_1), \cdots, g_n(r_n), T), & if \text{ condition} \\ F_2(g_1(r_1), \cdots, g_n(r_n), T), & else \end{cases} \tag{3.5}$$

where $\quad r_i = e_i/(cycle\ count), T = T_{current} - T_{idle}$

$$F_n = p_0 + p_1 * g_1(r_1) + ... + p_n * g_n(r_n) + p_{n+1} * T \tag{3.6}$$

As an example, the piecewise linear regression model for the Intel Core$^{\text{TM}}$i7-870 is shown in Eq. 3.7. Here, r_{MEM} refers to the counter LAST_LEVEL_CACHE_MISSES, r_{INSTR} refers to the counter UOPS_ISSUED, r_{FP} refers to the counter FP_COMP_OPS_EXE:X87, and r_{STALL} refers to the counter RESOURCE_STALLS:ANY. The piecewise model is broken based on the value of the memory counter. For the first part of the piece-wise model, the coefficient for the memory counter is zero (due to the very low number of memory operations we sampled).

$$\hat{P}_{core} = \begin{cases} 10.9246 + 0 * r_{MEM} + 5.8097 * r_{INSTR} + & \\ 0.0529 * r_{FP} + 6.6041 * r_{STALL} + 0.1580 * T, & if\ r_{MEM} < 1e-6 \\ 19.9097 + 556.6985 * r_{MEM} + 1.5040 * r_{INSTR} + & \\ 0.1089 * r_{FP} + -2.9897 * r_{STALL} + 0.2802 * T, & if\ r_{MEM} \geq 1e-6 \end{cases} \tag{3.7}$$

3.4 Validation

We evaluate our models by estimating per-core power for the SPEC 2006 [71], SPEC-OMP [72, 73], and NAS [74] benchmark suites. In our experiments, we run multi-threaded benchmarks with one thread per core, and single-threaded benchmarks with an instance on each core. We use gcc 4.2 to compile our benchmarks for a 64-bit architecture with optimization flags enabled, and we run all benchmarks to completion. We use the *pfmon* utility from the *perfmon2* library [75] to access the hardware performance counters. Table 3.3 gives system details of the machines on which we validated our power model. System power is based on measuring the power supply's current draw from the power outlet when the machine is idle. When we cannot find published values for idle processor power, we sum power draw when powered off and power saved by turning off cdrom and hard disk drives, removing all but one memory DIMM, and disconnecting fans. We subtract idle core power from idle system power to get *uncore* (baseline without processor) power. Change in the uncore power itself (due to DRAM or hard drive accesses, for instance) is included in the model estimations. Including temperature as a model input accounts for variation in uncore static power. We always run in the active power state (C0).

We use the *sensors* utility from the *lm-sensors* library to obtain core temperatures, and we use the Watts Up? Pro power meter [49] described in Chapter 2 to gather power data. This meter is accurate to within 0.1W, and it updates once per second. Although measuring the power at wall outlet has limitations in terms of sensitivity to fine-grained changes in power consumption, we use it for our experiments because of its low cost, non-intrusive use, and easy portability. Moreover, we observe that for the benchmarks in our experiments, the power consumption of the system does not change at the granularity of less than one second, and hence sampling power consumption at the rate of 1 sample/second meets our requirement for this modeling approach. Our modeling approach can easily be adapted to other power measurement techniques, such as those described in Chapter 2.

We incorporate the power model into a proof-of-concept, power-aware resource manager (a user-level meta-scheduler of Singh et al. [16, 63]) designed to maintain a specified power envelope. The meta-scheduler manages processes non-invasively, requiring no modifications to the applications or OS. It does so by suspending/resuming processes and, where supported, applying dynamic voltage/frequency scaling (DVFS) to alter core clock rates. For these experiments, we degrade the system power envelope by 10%, 20%, and 30% for Core$^{\text{TM}}$i7-870 platform. Lower envelopes render cores inactive, and thus we do not consider them. Finally, we incorporate the model in a kernel scheduler, implementing a pseudo power sensor per core. The device does not exist

(a) Configuration Parameters for Intel Platforms

	Intel Q6600	Intel Xeon E5430	Intel Core$^{\text{TM}}$i7-870
Cores/Chips	4, dual dual-core	4	
Frequency (GHz)	2.4	2.0, 2.66	2.93
Process (nm)	65	45	45
L1 Instruction	32 KB 8-way	32 KB 8-way	32 KB 4-way
L1 Data	32 KB 8-way	32 KB 8-way	32 KB 8-way
L2 Cache	4 MB 16-way shared	6 MB 16-way shared	256 KB 8-way exclusive
L3 Cache	N/A	N/A	8 MB 16-way shared
Memory Controller	off-chip, 2 channel	off-chip, 4 channel	on-chip, 2 channel
Main Memory	4 GB DDR2-800	8 GB DDR2-800	16 GB DDR3-1333
Bus (MHz)	1066	1333	1333
Max TDP (W)	105	80	95
Linux Kernel	2.6.27	2.6.27	2.6.31
Idle System Power (W)	141.0	180.0	54.0
Idle Processor Power (W)	38.0	27.0	10.0
Idle Temperature ($^{\circ}$C)	36	45	30

(b) Configuration Parameters for AMD Platforms

	AMD Phenom$^{\text{TM}}$9500	AMD Opteron$^{\text{TM}}$8212
Cores/Chips	4	8, quad dual-core
Frequency (GHz)	1.1, 2.2	2.0
Process (nm)	65	90
L1 Instruction	64 KB 2-way	64 KB 2-way
L1 Data	64 KB 2-way	64 KB 2-way
L2 Cache	512 KB 8-way exclusive	1024 KB 16-way exclusive
L3 Cache	2 MB 32-way shared	N/A
Memory Controller	on-chip, 2 channel	on-chip, 2 channel
Main Memory	4 GB DDR2-800	16 GB DDR2-667
Bus (MHz)	1100, 2200	1000
Max TDP (W)	95	95
Linux Kernel	2.6.25	2.6.31
Idle System Power (W)	84.1	302.6
Idle Processor Power (W)	20.1	53.6W
Idle Temperature ($^{\circ}$C)	36	33

Table 3.3: *Machine Configuration Parameters*

(a) PMCs Selected for Intel Platforms

Category	Intel Q6600	Intel E5430	Intel Core$^{\text{TM}}$i7-870
Memory	L2_LINES_IN	LAST_LEVEL_CACHE_MISSES	LAST_LEVEL_CACHE_MISSES
Instructions Executed	UOPS_RETIRED	UOPS_RETIRED	UOPS_ISSUED
Floating Point	X87_OPS_RETIRED	X87_OPS_RETIRED	FP_COMP_OPS_EXE:X87
Stalls	RESOURCE_STALLS	RESOURCE_STALLS	RESOURCE_STALLS:ANY

(b) PMCs Selected for AMD Platforms

Category	AMD Phenom$^{\text{TM}}$9500	AMD Opteron$^{\text{TM}}$8212
Memory	L2_CACHE_MISS	DATA_CACHE_ACCESSES
Instructions Executed	RETIRED_UOPS	RETIRED_INSTRUCTIONS
Floating Point	RETIRED_MMX_AND_FP_INSTRUCTIONS	DISPATCHED_FPU:OPS_MULTIPLY
Stalls	DISPATCH_STALLS	DECODER_EMPTY

Table 3.4: *PMCs Selected for Power-Estimation Model*

Benchmark	baseline	model (10msec)	model (100msec)	model (1sec)
ep.A serial	35.68	35.57	36.04	35.59
ep.A OMP	4.84	4.89	4.77	4.74
ep.A MPI	4.77	4.72	4.73	4.75
cg.A serial	5.82	5.83	5.83	5.83
cg.A OMP	1.95	1.95	1.95	1.95
cg.A MPI	2.19	2.20	2.20	2.20
ep.B serial	146.53	146.52	145.52	146.77
ep.B OMP	19.45	19.33	19.35	19.54
ep.B MPI	18.95	19.41	19.12	19.18
cg.B serial	559.58	560.50	560.11	560.10
cg.B OMP	91.29	92.64	96.90	89.92
cg.B MPI	79.11	79.18	79.18	79.05

Table 3.5: *Scheduler Benchmark Times for Sample NAS Applications on the AMD Opteron^{TM}8212 (sec)*

in hardware but is simulated by the power model module. Reads to the pseudo-device retrieve the power estimate computed most recently.

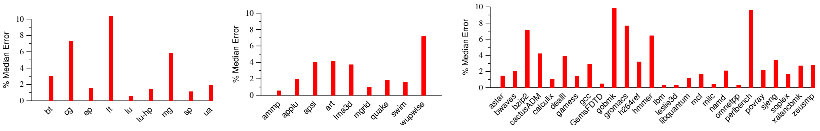
3.4.1 Computation Overhead

If computing the model is expensive, its use becomes limited to coarse timeslices. In this experiment we study the overhead of our power model to evaluate its use in an OS task scheduler. We use the scheduler developed by Boneti et al. [76] that is specifically tailored to High Performance Computing (HPC) applications that require that the OS introduce little or no overhead. In most cases, this scheduler delivers better performance with better predictability, and it reduces OS noise [76, 77]. The model's overhead depends on 1) the frequency with which the per-core power is updated, and 2) the complexity of the operations required to calculate the model. We calculate overhead by measuring execution time for our kernel scheduler running with and without reading the PMCs and temperature sensors. We vary the sample period from one minute to 10 msec. We time applications for five runs and take the average (differences among runs are within normal execution time variation for real systems). Table 3.5 gives measured times for the scheduler with and without evaluating the model. These data show that computing the model adds no measurable overhead, even at 10ms timeslices.

3.4.2 Estimation Error

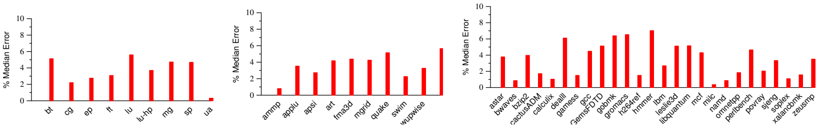
We assess model error by comparing our system power estimates to power meter output (which our power meter limits to a one-second granularity). We estimate the power consumption per core, and then sum up the power consumption for all cores with uncore power to compare the estimated power consumption with measured values. Figure 3.3 through Figure 3.7 [1] show percentage median error for the NAS, SPEC-OMP, and SPEC

[1] Data for Intel Q6600



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.3: *Median Estimation Error for the Intel Q6600 system*

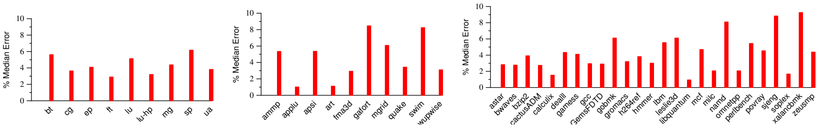


(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.4: *Median Estimation Error for Intel E5430 system*

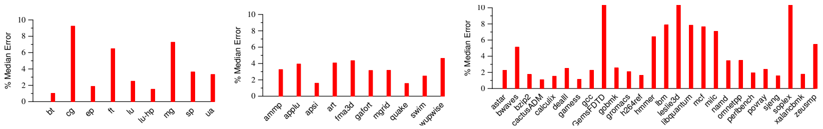
2006 applications on all systems. Figure 3.8 through Figure 3.12 show standard deviation of error for each benchmark suite. The occasional high standard deviations illustrate the main problem with our current infrastructure: instantaneous power measurements once per second do not reflect continuous performance counter activity since the last meter reading.

Estimation error ranges from 0.3% (*leslie3d*) to 7.1% (*bzip2*) for the Intel Q6600 system, from 0.3% (*ua*) to 7.0% (*hmmer*) for the Intel E5430 system, from 1.02% (*bt*) to 9.3% (*xalancbmk*) for the AMD Phenom$^{\text{TM}}$system, from 1.0% (*bt.B*) to 10.7% (*soplex*) for the AMD Opteron$^{\text{TM}}$8212 system and from 0.17% (*art*) to 9.07% (*libquantum*) for the Intel Core$^{\text{TM}}$i7-870 system. On Intel Q6600, only five (out of 45) applications exhibit median error exceeding 5%; on the Intel E5430, only six exhibit error exceeding 5%; on the AMD Phenom$^{\text{TM}}$, eighteen exhibit error exceeding 5%; and on the AMD Opteron$^{\text{TM}}$8212, thirteen exhibit error exceeding 5%. For the Intel Core$^{\text{TM}}$i7-870, median estimations for only seven applications exceed 5% error. Table 3.6 shows the summary of power estimation errors for our model across all platforms. These data suggest that our model works better on Intel machines compared to AMD machines.



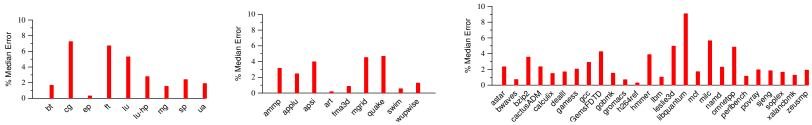
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.5: *Median Estimation Error for the AMD Phenom^{TM}9500*



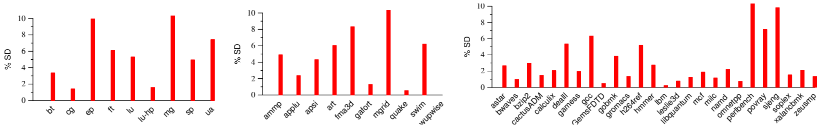
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.6: *Median Estimation Error for the AMD Opteron^{TM}8212*



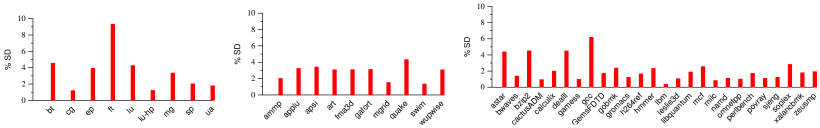
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.7: *Median Estimation Error for the Intel Core^{TM}i7*



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.8: *Standard Deviation of Error for the Intel Q6600 system*

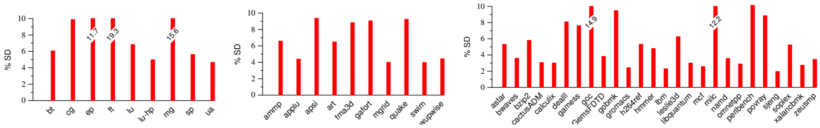


(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.9: *Standard Deviation of Error for Intel E5430 system*

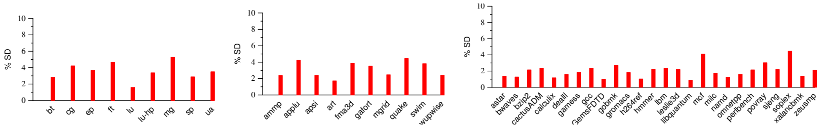
Benchmark	Intel Q660	Intel E5430	Intel Core^{TM}i7-870	AMD Phenom^{TM}	AMD 8212
SPEC 2006	1.1	2.8	2.22	3.5	4.80
NAS	1.6	3.9	3.11	4.5	2.55
SPEC OMP	1.6	3.5	2.02	5.2	3.35
Overall	1.2	3.6	2.07	3.8	4.38

Table 3.6: *Estimation Error Summary*



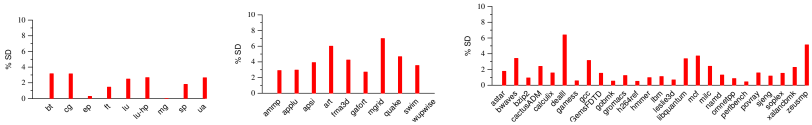
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.10: *Standard Deviation of Error for the AMD Phenom^{TM}9500*



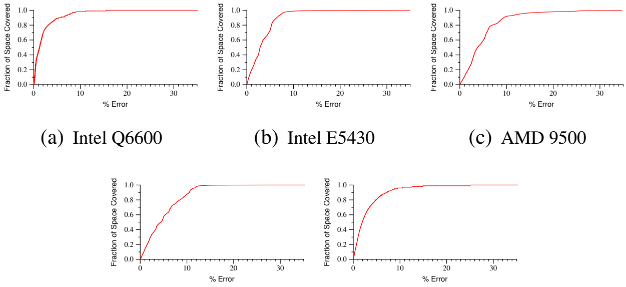
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.11: *Standard Deviation of Error for the AMD Opteron^{TM}8212*



(a) NAS (b) SPEC-OMP (c) SPEC 2006

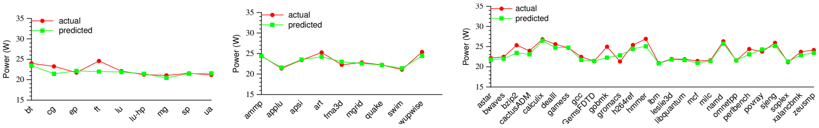
Figure 3.12: *Standard Deviation of Error for the IntelCore^{TM}i7*



(a) Intel Q6600 (b) Intel E5430 (c) AMD 9500

(d) AMD 8212 (e) Core$^{\text{TM}}$i7

Figure 3.13: *Cumulative Distribution Function (CDF) Plots Showing Fraction of Space Predicted (y axis) under a Given Error (x axis) for Each System*

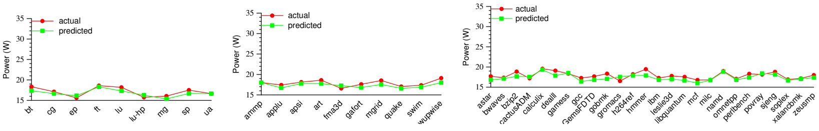


(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.14: *Estimated versus Measured Error for the Intel Q6600 system*

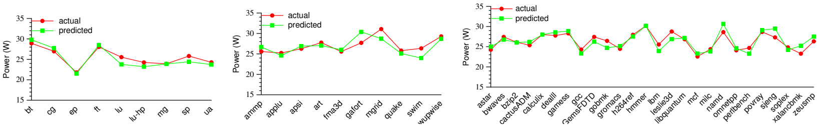
Figure 3.13 shows model coverage via Cumulative Distribution Function (CDF) plots for the suites. On Q6600, 85% of estimates have less than 5% error, and 98% have less than 10%. On E5430, 73% have less than 5% error, and 99% less than 10%. On Phenom$^{\text{TM}}$, 59% have less than 5% error, and 92% less than 10%. On 8212, 37% have less than 5% error, and 76% have less than 10%. For the Core$^{\text{TM}}$i7-870, 82% of estimates have less than 5% error and 96% have less than 10% error. The vast majority of estimates exhibit very small error.

These errors are not excessive, but lower is better. Prediction errors are not clustered, but are spread throughout application execution. Model accuracy depends on the



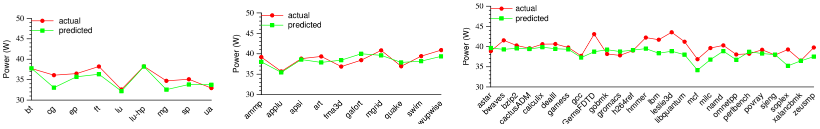
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.15: *Estimated versus Measured Error for Intel E5430 system*



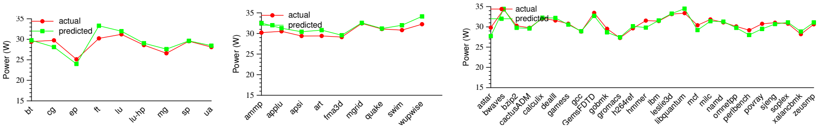
(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.16: *Estimated versus Measured Error for the AMD Phenom^{TM} 9500*



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.17: *Estimated versus Measured Error for the AMD Opteron^{TM} 8212*



(a) NAS (b) SPEC-OMP (c) SPEC 2006

Figure 3.18: *Estimated versus Measured Error for the Intel Core^{TM} i7-870*

particular PMCs available on a given platform. If available PMCs do not sufficiently represent the microarchitecture, model accuracy will suffer. For example, the AMD Opteron$^{\text{TM}}$8212 processor supports no single counter giving total floating point operations. Instead, separate PMCs track different types of floating point operations. We therefore choose the one most highly correlated with power. Model accuracy would likely improve if a single PMC reflecting all floating point operations were available. For processor models supporting only two Model Specific Registers for reading PMC values, capturing the activity of four counters requires multiplexing the counting of PMC-related events. This means that events are counted for only half a second (or half the total sampling period), and are doubled to estimate the value over the entire period. This approximation can introduce inaccuracies when program behavior is changing rapidly. The model estimation accuracy is also affected by accuracy of hardware sensors. For instance, the Opteron$^{\text{TM}}$8212 temperature sensor data suffers with low accuracy [78].

Similarly, even though the microbenchmark tries to cover all scenarios of power consumption, the resulting regression model will represent a generalized case. This is especially true for a model that tries to estimate power for a complex microarchitecture using limited number of counters. For example, floating point operations can consist of add, multiply, or divide operations, all of which use different execution units and hence consume a different amounts of power. If the application being studied uses operations not covered by the training microbenchmark, model accuracy could also suffer.

A model can be no more accurate than the information used to build it. Performance counter implementations display nondeterminism [66] and error [64]. All of these impact model accuracy. Given all these sources of inaccuracy, there seems little need for more complex, sophisticated mathematics when building a model.

3.5 Management

In the previous section, we discussed our approach to estimating the power consumption of processor resources using power modeling. In this section, we discuss the use of our power model by resource managers that perform task scheduling.

To demonstrate one use of our on-line power model, we experiment with the user-level meta-scheduler from Singh et al. [16, 63]. This live power management application maintains a user-defined system power budget by power-aware scheduling of tasks and/or by using DVFS. We use the power model to compute core power consumption dynamically. The application spawns one process per core. The scheduler reads PMC values via *pfmon* and feeds the sampled PMC values to the power model to estimate core power consumption.

The meta-scheduler binds the affinity of the processes to a particular core to simplify

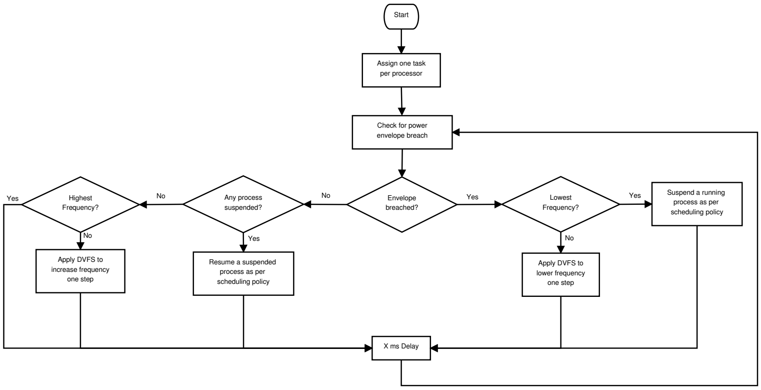

Start
Assign one task per processor
Check for power envelope breach
Envelope breached?
Yes
Lowest Frequency?
Yes
Suspend a running process as per scheduling policy
No
Apply DVFS to lower frequency one step
No
Any process suspended?
Yes
Resume a suspended process as per scheduling policy
No
Highest Frequency?
Yes
No
Apply DVFS to increase frequency one step
X ms Delay

Figure 3.19: *Flow diagram for the Meta-Scheduler*

task management and power estimation. It dynamically calculates power values at a set interval (one second, in our case) and compares the system-power envelope with the sum of power for all cores together with the uncore power.

When the scheduler detects a breach in the power envelope, it takes steps to control the power consumption. The scheduler employs two knobs to control system-power consumption: dynamic voltage-frequency scaling as a fine knob, and process suspension as a coarse knob. When the envelope is breached, the scheduler first tries to lower the power consumption by scaling down the frequency. If power consumption remains too high, the scheduler starts suspending processes to meet the envelope's demands. The scheduler maintains the record of the power being consumed by the process at the time of suspension. When the estimated power consumption is less than the target power envelope, the scheduler checks whether any of the suspended processes can be resumed based on the power they were consuming at the time of suspension. If the gap between the current power consumption and the target power budget is not enough to resume a suspended process, and if the processor is operating at a frequency lower than maximum, the scheduler scales up the voltage-frequency. Fig. 3.19 shows the flow diagram of the meta-scheduler.

3.5.1 Sample Policies

When the scheduler suspends a process, it tries to choose the process that will have the least impact on completion time of all the processes. We explore the use of our power model in a scheduler via two sample policies for process suspension.

The **Throughput** policy targets maximum power efficiency (max instructions/watt) under a given power envelope. When the envelope is breached, the scheduler calculates

Benchmark Category	Benchmark Applications	Peak System Power (W)
CPU-Bound	ep, gamess, namd, povray	130
Moderate	art, lu, wupwise, xalancbmk	135
Memory-Bound	astar, mcf, milc, soplex	130

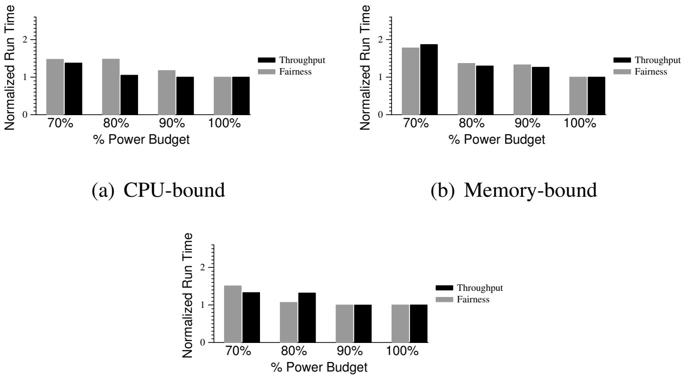
Table 3.7: *Workloads for Scheduler Evaluation*

the ratio of instructions/UOPS retired to the power consumed for each core and suspends the process having committed the fewest instructions per watt of power consumed. When resuming a process, it selects the process (if there is more than one suspended process) that had committed the maximum instructions/watt at the time of suspension. This policy favors processes that are less often stalled while waiting for load operations to complete. This policy thus favors CPU bound applications.

The **Fairness** policy divides the available power budget equally among all processes. When applying this policy, the scheduler maintains a running average of the power consumed by each core. When the scheduler must choose a process for suspension, it chooses the process having consumed the most average power. For resumption, the scheduler chooses the process that had consumed the least average power at the time of suspension. This policy strives to regulate core temperature, since it throttles cores consuming the most average power. Since there is high correlation between core power consumption and core temperature, this makes sure that the core with highest temperature receives time to cool down, while cores with lower temperature continue working. Since memory-bound applications are stalled more often, they consume less average power, and so this policy favors memory-bound applications.

3.5.2 Experimental Setup

For the scheduler experiments, we use an Intel Core$^{\text{TM}}$i7-870 system, using DVFS to vary frequencies between 2.926, 2.66, 2.394, 2.128, 1.862, 1.596 and 1.33 GHz. We form separate power models for different frequencies. The power manager can thus better implement policy decisions by estimating power for each frequency. If all core frequencies have been reduced but the power envelope is still breached, we suspend processes to reduce power. We compare against runtimes with no enforced power envelope. We divide our workloads into three sets based on *CPU intensity*. We define CPU intensity as the ratio of instructions retired to last-level cache misses. The three sets are designated as CPU-Bound, Moderate, and Memory-Bound workloads (in decreasing order of CPU intensity). The workloads categorized in these sets are listed in Table 3.7.



(a) CPU-bound

(b) Memory-bound

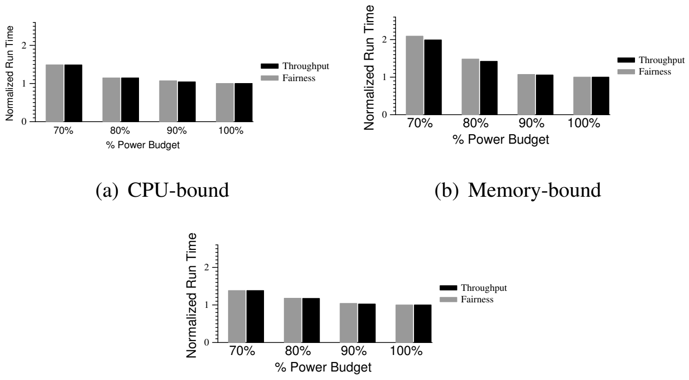
(c) Moderate

Figure 3.20: *Runtimes for Workloads on the Intel Core^{TM}i7-870 (without DVFS)*

3.5.3 Results

Fig. 3.20 shows the normalized runtimes for all the workloads when only process suspension is used to maintain the power envelope. The Throughput policy should favor CPU-bound workloads, while the Fairness policy should favor memory bound workloads, but this distinction is not always visible. This is because of the differences in the runtimes of the various workloads. This can be seen from the execution times of the CPU-bound benchmarks when using the Throughput policy. The CPU bound applications *ep* and *gamess* have the lowest computational intensities and execution times. As a result, these two applications are suspended most frequently, which does not affect the total execution time, even when the power envelope is set to 80% of peak usage.

Fig. 3.21 shows the results when the scheduler uses both DVFS and process suspension to maintain the given power envelope. As noted, the scheduler uses DVFS as a fine knob and process suspension as a coarse knob in maintaining the envelope. The Intel Core^{TM}i7-870 processor that we use for our experiments supports fourteen different voltage-frequency points. These frequency points range from 2.926 GHz to 1.197 GHz. For our experiments, we make models for seven frequency points (2.926, 2.66, 2.394, 2.128, 1.862, 1.596 and 1.33 GHz), and we adjust the processor frequency across these points. Our experimental results show that for CPU-bound and moderate benchmarks, there is little difference in execution time under different suspension policies. This suggests that for these applications, the scheduler hardly needs to suspend the processes and regulating DVFS points proves sufficient to maintain the power envelope. Performance for DVFS degrades compared to cases where no DVFS is used. The explanation for



(a) CPU-bound

(b) Memory-bound

(c) Moderate

Figure 3.21: *Runtimes for Workloads on the Intel Core^{TM}i7-870 (with DVFS)*

this lies in the difference between runtimes of different applications within the workload sets. When no DVFS is used, all processes run at full speed. And even when one of the processes is suspended, if that process is not critical, it still runs at full speed later in parallel with the critical process. But in the case of DVFS being given higher priority over process suspension, when the envelope is breached, all processes are slowed, and this affects total execution time.

3.5.4 Related Work

Much previous work leverages performance counters for power estimation. Our work is based on the approach of Singh et al. [16,63]; we augment that work by porting and validating the model on many platforms, improving accuracy of the model by augmenting the microbenchmark and counter selection methodology, analyzing estimation errors, and exploiting multiple DVFS operating points for scheduler validation.

Joseph and Martonosi [10] use PMCs to estimate power in a simulator (SimpleScalar [79]) and on real hardware (a Pentium Pro). Their hardware supports two PMCs, while they require twelve. They perform multiple benchmark runs to collect data, forming the model offline. Multiplexing twelve PMCs would require program behavior to be relatively constant across the entire sampling interval. They cannot estimate power for 24% of the chip, and so they assume peak power for those structures.

Contreras and Martonosi [14] use five PMCs to estimate power for different frequencies on an XScale system (with an in-order uniprocessor). Like Joseph and Martonosi, they gather data from multiple benchmark runs. They derive power weights for frequency-voltage pairs, and form a parameterized linear model. Their model exhibits low percentage error, like ours, but they test on only seven applications, and their methodology is only demonstrated for a single platform. This methodology, like that of Joseph and Martonosi, is not intended for on-line use.

Economou et al. [15] use PMCs to predict power on a blade server. They profile the system using application-independent microbenchmarks. The resulting model estimates power for the CPU, memory, and hard drive with 10% average error. Wu et al. [80] also use microbenchmarks to develop a model using a variable number of PMCs to estimate active power of Pentium 4 functional units. They measure CPU power with a clamp-on ammeter. Such invasive approaches are neither applicable to systems in the field, nor scalable to HPC clusters.

Merkel and Bellosa [81] use PMCs to estimate power per processor in an eight-way SMP, shuffling processes to reduce overheating of individual processors. They do not state which PMCs they use, nor how they are chosen. Their goal is not to reduce power, but to distribute it. Their estimation method suffers less than 10% error in a linux kernel implementation.

Lee and Brooks [18] predict power via statistical inference models. They build correlations based on hardware parameters, using the most significant parameters to train their model. They profile their design space a priori, and then estimate power on random samples. This methodology requires training on the same applications for which they want to estimate power, and so their approach depends on having already sampled all applications of interest.

Spiliopoulos et al. [23] use instructions executed and instructions retired counter to develop a power model. Their model shows high errors for memory-bound benchmarks since they do not consider memory operations in their model.

Instead of applying DVFS per core, Rangan et al. [19] study thread placement and migration among independent voltage and frequency domains. This *thread motion* permits much finer-grain control over power management and delivers better performance than conventional DVFS for a given power budget. Grochowski et al. [82] study four methods to control energy per instruction (EPI), finding that a combination of VFS and asymmetric cores offers the most promising approach to balancing latency and throughput in an energy-constrained environment. Annavaram et al. [83] leverage this work to throttle EPI by scheduling multithreaded tasks on an asymmetric CMP, using more EPI in periods of low parallelism to make better use of a fixed power budget.

Rajamani et al. [11] use their power estimation model to drive a power management policy called *PerformanceMaximizer*. For a given power budget, they exploit the DVFS levels of the processor to try to maximize the processor performance. For each performance state (P-state), they apply their power model to estimate power consumption at the current P-state. They use the calculated power value to estimate the power consumption at other P-states by linearly scaling the current power value with frequency. The scheduler increases the performance state to highest level that it estimates would be safely below the power budget value.

Banikazemi et al. [20] use a power-aware meta-scheduler. Their meta-scheduler monitors the performance, power and energy of the system by using performance counters and in-built power monitoring hardware. It uses this information to dynamically remap the software threads on multi-core servers for higher performance and lower energy usage. Their framework is flexible enough to substitute the hardware power monitor with a performance-counter based model.

Isci et al. [21] analyze global power management policies to enforce a given power budget and to minimize power consumption for the given performance target. They conduct their experiments on the Turandot [84] simulator. They assume the presence of on-core current sensors to acquire core power information, while they use performance counters to gather core performance information. They have developed a global power manager that periodically monitors the power and performance of each core and sets the operating mode (akin to DVFS performance states) of the core for the next monitoring interval. They assume that the power mode can be set independently for each core. They experiment with three policies to evaluate their global power manager. The *Priority* policy assigns different priorities to different cores in a multi-core processor and tries to speed up the core with highest priority while slowing down the lowest priority core when the the power consumption overshoots the assigned budget. The policy called *pullHipushLo* is similar to our Fairness policy from the above case study; it tries to balance the power consumption of each core by slowing down the core with highest power consumption when the power budget is exceeded and speeds up the core with lowest power consumption when there is a power surplus. *MaxBIPS* tries to maximize the system throughput by choosing the combination of power modes on different cores that is predicted to provide maximum overall BIPS (Billion Instructions Per Second).

Meng et al. [22] apply a multi-optimization power saving strategy to meet the constraints of a chip-wide power budget on reconfigurable processors. They run a global power manager that configures the CPU frequency and/or cache size of individual cores. They use risk analysis to evaluate the trade-offs between power saving optimizations and potential performance loss. They select the power-saving strategies at design time to create a static pool of candidate optimizations. They make an analytic power and performance model using performance counters and sensors that allows them to quickly evaluate many power modes and enables their power manager to choose a global power mode at periodic intervals that can obey the processor-wide power budget while maximizing the throughput.

3.6 Conclusion

We derive statistical, piece-wise multiple linear regression power models mapping PMC values and temperature readings to power, and demonstrate their accuracy for the SPEC 2006, SPEC-OMP, and NAS benchmark suites. We write microbenchmarks to exercise the PMCs in order to characterize the machine and run those microbenchmarks with a power meter plugged in to generate data for building the models. For our regression models, we select the PMCs that correlate most strongly with measured power. Because they are built on microbenchmark data, and not actual workload data, the resulting models are application independent. We apply the models to 45 benchmarks (including multithreaded applications) on five CMPs containing dual- or quad-core chips totaling four or eight cores. In spite of our generality, estimation errors are consistently low across five different systems. We observe overall median errors per machine between 1.2% and 4.4%.

We then show the effectiveness of our power model for live power management on an Intel Core$^{\text{TM}}$system with and without DVFS. We suspend and resume processes based on per-core power usage, ensuring that a given power envelope is not breached. We also scale core frequencies to remain under the power envelope.

As numbers of cores and power demands continue to grow, efficient computing requires better methods for managing individual resources. The per-core power estimation methodology presented here extends previously published models in both breadth and depth, and represents a promising tool for helping meet those challenges, both by providing useful information to resource managers and by highlighting opportunities for improving hardware support for energy-aware resource management. Such support is essential for fine-grained, power-aware resource management.

4

Characterization of Intel's Restricted Transactional Memory

4.1 Introduction

Transactional memory (TM) [85] simplifies some of the challenges of shared-memory programming. The responsibility for maintaining mutual exclusion over arbitrary sets of shared-memory locations is devolved to the TM system, which may be implemented in software (STM) or hardware (HTM). TM presents the programmer with fairly easy-to-use programming constructs that define a *transaction* — a piece of code whose execution is guaranteed to appear as if it occurred atomically and in isolation.

The research community has explored this design space in depth, and a variety of proposed systems take advantage of transaction characteristics to simplify implementation and improve performance [86–89]. Hardware support for transactional memory has been implemented in Rock [90] from Sun Microsystems, Vega from Azul Systems [91], and Blue Gene/Q [92] and System z [93] from IBM. Haswell is the first Intel product to provide such hardware support. Intel's Transactional Synchronization Extensions (TSX) allow programmers to run transactions on a best-effort HTM implementation, i.e., the platform provides no guarantees that hardware transactions will commit successfully, and thus the programmer must provide a non-transactional path as a fallback mechanism. Intel TSX supports two software interfaces to execute atomic blocks: Hardware Lock Elision (HLE) is an instruction set extension to run atomic blocks on legacy hardware, and Restricted Transactional Memory (RTM) is a new instruction set interface to execute transactions on the underlying TSX hardware.

The previous chapters have highlighted the need for better hardware introspection into power consumption. As better hardware support for such introspection becomes available, it is important to evaluate accuracy and acuity so that users can choose among various combinations of measurement and modeling techniques. Since Intel's Core i7 4770 microarchitecture is among the first to support both power modeling and support for transactional memory, it makes an interesting platform for analyzing power/performance trade-offs.

As an initial study, we compare the Haswell RTM performance and energy of the Haswell implementation of RTM to those of other approaches for controlling concurrency. We use a variety of workloads to test the susceptibility of RTM's best-effort nature to performance degradation and increased energy consumption. We compare RTM performance to TinySTM, a software transactional memory implementation that uses time to reason about the consistency of transactional data and about the order of transaction commits.[1] We highlight these crossover points and analyze the impact of thread scaling on energy expenditure.

We find that RTM performs well with small to medium working sets when the amount of data (particularly that being written) accessed in transactions is small. When data contention among concurrent transactions is low, TinySTM performs better than RTM, but as contention increases, RTM consistently wins. RTM generally suffers less overhead than TinySTM for single-threaded runs, and it is more energy-efficient when working sets fit in cache.

4.2 Experimental Setup

The Intel 4th Generation Core$^{\text{TM}}$ i7 4770 processor comprises four physical cores that can run up to eight simultaneous threads when hyper-threading is enabled. Each core has two eight-way 32 KB private L1 caches (separate for I and D), a 256 KB private L2 cache (for combined I and D), and an 8 MB shared L3 cache, with 16 GB of physical memory on board. We compile all microbenchmarks, benchmarks, and synchronization

[1] We choose TinySTM because during our experiments we find that it consistently outperforms other STM alternatives like TL2 (to which RTM was compared in another recent study [94]).

libraries using gcc v4.8.1 with *-O3* optimization flag. We use the *-mrtm* flag to access the Intel TSX intrinsics. We schedule threads on separate physical cores (unless running more than four threads) and fix the CPU affinity to prevent migration.

We modify the *task* example from *libpfm4.4* to read both the performance counters and the processor package energy via the Running Average Power Limit (RAPL) [56] interface. As we concluded in Chapter 2, RAPL is fairly accurate when the time between successive RAPL counter reads is more than tens of milliseconds. Since all our benchmarks run for at least few seconds, we decided to use RAPL for our energy measurements. We implement Intel TSX synchronization as a separate library and add RTM definitions to the STAMP *tm.h* file. When transactions fail more than eight times, we invoke reader/writer lock-based fallback code to ensure forward progress. If the return status bits indicate that an abort was due to another thread's having acquired the lock (in the fallback code), we wait for the lock to be free before retrying the transaction. The following shows pseudocode for a sample transaction.

Algorithm 1 Implementation of `BeginTransaction`

```
while true do
    nretries $\leftarrow$ nretries + 1
    status $\leftarrow$ _xbegin()
    if status = _XBEGIN_STARTED then
        if arch_read_can_lock(serialLock) then
            return
        else
            _xabort(0)
        end if
    end if
    {*** fall-back path ***}
    while not arch_read_can_lock(serialLock) do
        _mmpause()
    end while
    if nretries $\geq$ MAX_RETRIES then
        break
    end if
end while
arch_write_lock(serialLock);
return
```

4.3 Microbenchmark analysis

4.3.1 Basic RTM Evaluation

We first quantify RTM's hardware limitations that affect its performance using microbenchmark studies. We detail the results of these experiments below.

RTM Capacity Test. To test the limitations of read-set and write-set capacity for RTM, we create a custom microbenchmark, results for which are shown in Fig. 4.1. The abort rate of write-only transactions tops out at 512 cache blocks (the size of L1 data

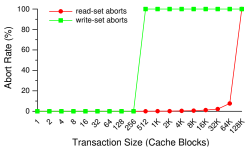


Figure 4.1: *RTM Read-Set and Write-Set Capacity Test*

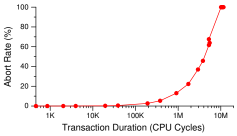


Figure 4.2: *RTM Abort Rate versus Transaction Duration*

cache). We suspect this is because write-sets are tracked only in L1, and so evicting any transactionally written cache line from L1 results in a transaction abort. For read-sets, the abort rate saturates at 128K cache blocks (the size of L3 cache). This suggests that evicting transactionally read cache lines from L3 (but not L1) triggers transaction aborts, and thus RTM maintains performance for much larger read-sets than write-sets.

RTM Duration Test. Since RTM aborts can be caused by system events like interrupts and context switches, we study the effects of transaction duration (measured in CPU cycles) on success rate. For this analysis, we use a single thread, set the working-set size to 64 bytes, and set the number of writes inside the transaction to 0. This tries to ensure that the number of aborts due to memory events and conflicts remains insignificant. We gradually increase the duration by increasing the number of reads within the transaction. Fig. 4.2 shows that transaction duration begins to affect the abort rate at about 30K cycles and that durations of more than 10M cause all transactions to abort (note that these results are likely machine dependent).

RTM Overhead Test. Next we quantify performance overheads for RTM compared to spin locks and the atomic compare-and-swap (CAS) instruction. For this test, we create a microbenchmark that removes elements from a queue (defined in the STAMP [95] library). We initialize the queue to 1M elements, and threads extract elements until the queue is empty. Work is not statically divided among threads. We first compare RTM against the spinlock implementation in the Linux kernel (`arch/x86/include/asm/spinlock.h`). We then compare against a version of the queue implementation modified to use CAS in `queue\_pop()`. For RTM, we simply retry the transaction on aborts.

We perform three sets of experiments. To observe the cost of starting an RTM transaction in the absence of contention, we first run single-threaded experiments. We repeat the experiment with four threads to generate a high-contention workload. Finally, we lower contention by making threads work on local data for a fixed number of operations after each critical section. Table 4.1 summarizes execution times normalized to those of the lock-based version.

	Type of synchronization			
Contention	None	Lock	CAS	RTM
None	0.64	1	1.05	1.45
Low	N/A	1	0.64	0.69
High	N/A	1	0.64	0.47

Table 4.1: *Relative Overhead of RTM versus Locks and CAS*

Table 4.1 shows that the cost of starting a transaction makes RTM perform worse than the other alternatives when executing non-contended critical sections with few instructions. RTM suffers about a 45% slowdown compared to using locks and CAS, and it takes over twice the time of an unsynchronized version. In contrast, our multi-threaded experiments reveal that RTM exhibits roughly 30% and 50% lower overhead than locks in low and high contention, respectively, while CAS is in both cases around 35% better than locks. Note that transactions avoid hold-and-wait behavior, which seems to give RTM an advantage in our study. When comparing locks and CAS, the higher lock overhead is likely due in part to the ping-pong coherence behavior of the cache line containing the lock and to cache-to-cache transfers of the line holding the queue head.

4.3.2 Eigenbench Characterization

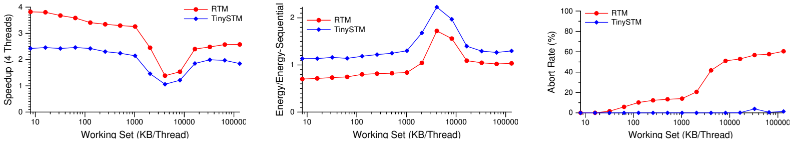
To compare RTM and STM in detail, we next study the behaviors of Hong et al.'s Eigenbench [96]. This parameterizable microbenchmark attempts to characterize the design space of TM systems by orthogonally exploring different transactional application behaviors. Table 4.2 defines the seven characteristics we use to compare performance and energy expenditure of the Haswell RTM implementation and the TinySTM [97] software transactional memory system. Hong et al. [96] provide a detailed explanation of these characteristics and the equations used to quantify them.

Unless otherwise specified, we use the following parameters in our experiments, results for which we average over 10 runs. Transactions are 100 memory references (90 reads and 10 writes) in length. We use one small (16KB) and one medium (256KB) working set size to demonstrate the differences in RTM performance. Since L1 size has no influence on TinySTM's abort rates, we only show TinySTM results for the smaller working set size. To prevent L1 cache interference, we run four threads with hyperthreading disabled as our default, and we fix the CPU affinity to prevent thread migration.

Characteristic	Definition
Concurrency	Number of concurrently running threads
Working-set size[a]	Size of frequently used memory
Transaction length	Number of memory accesses per transaction
Pollution	Fraction of writes to total memory accesses inside transaction
Temporal locality	Probability of repeated address inside transaction
Contention	Probability of transaction conflict
Predominance	Fraction of transactional cycles to total application cycles

[a] Working-set size for Eigenbench is defined per-thread.

Table 4.2: *Eigenbench TM Characteristics*

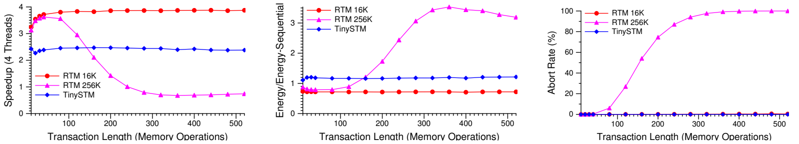


(a) Speedup (b) Relative Energy (c) Abort Rate

Figure 4.3: *Eigenbench Working-Set Size*

For each characteristic, we compare RTM and TinySTM performance and energy (versus sequential runs of the same code) and transaction-abort rates. For the graphs in which we plot two working-set sizes for RTM, the speedups and energy efficiency given are relative to the sequential run of the same size working set.

Working-Set Size. Fig. 4.3 shows Eigenbench results over a logarithmic scale as we increase each thread's working set from 8KB to 128MB. RTM performs best with the smallest working set, and its performance gradually degrades as working-set size increases. The performance of both RTM and TinySTM drops once the combined working sets of all threads exceed the 8MB L3 cache. RTM performance suffers more because events like L3 evictions, page faults, and interrupts trigger a transaction abort, which is not the case for TinySTM. The speedups of both RTM and TinySTM are lowest at working sets of 4MB: at this point, the parallelized code's working sets (16MB in total) exceed L3, but the working set of the sequential version (4MB) still fits. For working sets above 4MB, the sequential version starts encountering L3 misses, and thus the relative performances of both transactional memory implementations begins to improve. Parallelizing the transactional code using RTM is energy-efficient compared to sequential version when the combined working sets of all threads fits inside the cache.



(a) Speedup (b) Relative Energy (c) Abort Rate

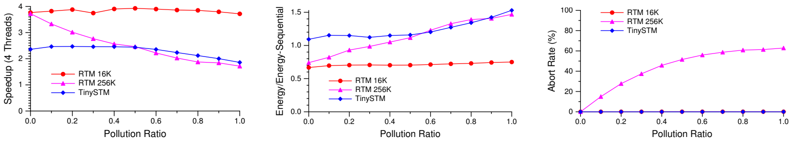
Figure 4.4: *Eigenbench Transaction Length*

Transaction Length. Fig. 4.4 shows Eigenbench results as we increase the transaction length from 10 to 520 memory operations. When the working set (16KB) fits within L1, RTM outperforms TinySTM for all transaction lengths. For 256KB working sets, RTM performance drops sharply when the transaction length exceeds 100 accesses. Recall that evicting write-set data from the L1 triggers transaction aborts, but when the working set fits within L1, such evictions are few. As the working set grows, the randomly chosen addresses accessed inside the transactions have a higher probability of occupying more L1 cache blocks, and hence they are more likely to be evicted. In contrast, TinySTM shows no performance dependence on working-set size. The overhead of starting the hardware transaction affects RTM performance for very small transactions. As observed in the working-set analysis above, RTM is more energy efficient than both the sequential run and TinySTM for all transaction lengths when using the smaller working set. When using the larger working set, RTM expends more energy for transactions exceeding 120 accesses.

Pollution. Fig. 4.5 shows results when we test symmetry (with respect to handling read-sets and write-sets) by gradually increasing the fraction of writes. The pollution level is zero when all memory operations in the transaction are reads and one when all are writes. When the working set fits within L1, RTM shows almost no asymmetry. But for the larger working-set size, RTM speedup suffers as the level of pollution increases. TinySTM outperforms RTM when the pollution level increases beyond 0.4.

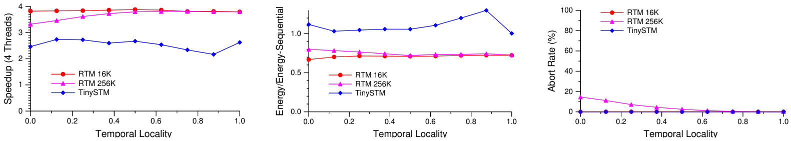
Temporal Locality. We next study the effects of temporal locality on TM performance (where temporal locality is defined as the probability of repeatedly accessing the same memory address within a transaction). The results in Fig. 4.6 reveal that RTM shows no dependence on temporal locality for the 16KB working set, but performance degrades for the 256KB working set (where low temporal locality increases the number of aborts due to L1 write-set evictions). In contrast, TinySTM performance degrades as temporal locality increases, indicating that it favors unique addresses unless only one address is being accessed inside the transaction (locality = 1.0).

Contention. This analysis studies the behavior of TM systems when the level of contention is varied from low to high. We set the working-set size to 2MB for both



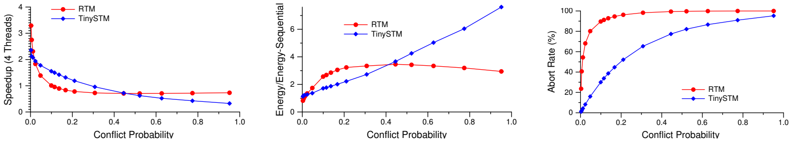
(a) Speedup (b) Relative Energy (c) Abort Rate

Figure 4.5: *Eigenbench Pollution*



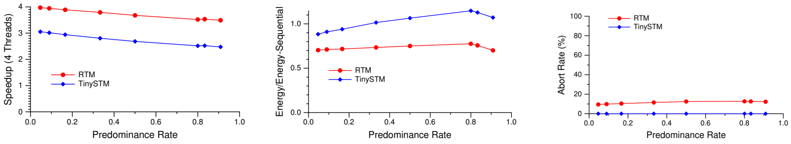
(a) Speedup (b) Relative Energy (c) Abort Rate

Figure 4.6: *Eigenbench Temporal Locality*



(a) Speedup (b) Relative Energy (c) Abort Rate

Figure 4.7: *Eigenbench Contention*



(a) Speedup (b) Relative Energy (c) Abort Rate

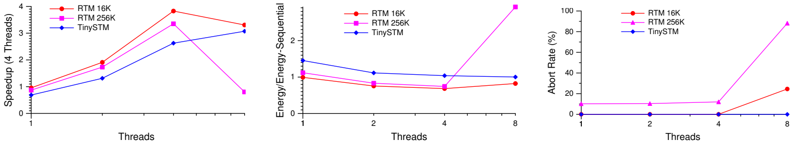
Figure 4.8: *Eigenbench Predominance*

RTM and TinySTM. The level of contention is calculated as an approximate value representing the probability of a transaction causing a conflict (as per the probability formula given by Hong et al. [96]). The conflict probability figures shown in Fig. 4.7 are calculated at word granularity and hence are specific to TinySTM. Since RTM detects conflicts at the granularity of cache line (64 bytes), the contention level is actually higher for RTM with the same workload configuration. When the degree of contention among competing threads is very low, RTM performs better than TinySTM. For low to medium contention, TinySTM considerably outperforms RTM. However, for high contention workloads, TinySTM performance degrades while RTM performance remains almost the same.

Predominance. We study the behavior of the TM systems when varying the fraction of application cycles executed within transactions to the total number of application cycles. For this analysis, we set working-set size to 256KB for both TM systems, we set contention to zero, and we vary the predominance ratio from 0.125 to 0.875. Fig. 4.8 shows that performance for both RTM and TinySTM suffers as the ratio of transactional cycles to non-transactional cycles grows. This can be attributed to the overheads associated with the TM systems: for the same level of predominance, TinySTM introduces more overhead because it must instrument the program memory accesses.

Concurrency. Next we study how the performance and energy of RTM and TinySTM scale when concurrency is increased from one thread to eight. Fig. 4.9 shows that RTM scales well up to four threads. At eight threads, the L1 cache is shared between two threads running on the same core. This cache sharing degrades performance for the larger working set more than for the smaller working set because hyper-threading effectively halves the write-set capacity of RTM. In contrast, TinySTM scales well up to eight threads. For the small working set, RTM proves to be more energy-efficient than either TinySTM or the sequential runs.

The results from the Eigenbench analysis help us in identifying a range of workload characteristics for which either RTM or TinySTM is better performing or more energy efficient. We next apply the insights gained from our microbenchmark studies to analyze the performance and energy numbers we see for the STAMP benchmark suite.



(a) Speedup (b) Relative Energy (c) Abort Rate

Figure 4.9: *Eigenbench Concurrency*

4.4 HTM versus STM using STAMP

Next we use the STAMP transactional memory benchmark suite [95] to compare the performance and energy efficiency of RTM and TinySTM. We use the lock-based fallback mechanism explained in Section 4.2 and run the applications with input sizes that create large working sets and high contention. We average all results over 10 runs. Fig. 4.10 shows STAMP execution times for RTM and TinySTM normalized to the average execution time of sequential (non-TM) runs. Fig. 4.11 shows the corresponding energy expenditures, again normalized to the average energy of the sequential runs. Results for single-threaded TM versions of the benchmarks illustrate the TM system overheads.

`bayes` has a large working set and long transactions, and thus RTM performs worse than TinySTM. This corresponds to our findings in the Eigenbench transaction-length analysis in Fig. 4.4. As expected, RTM does not improve the performance of `bayes` as the number of threads scales, and TinySTM performs better overall. Since the time the `bayes`'s algorithm takes to learn the network dependencies depends on the computation order, we see significant deviations in learning times for multi-threaded runs.

`genome` has medium transactions, a medium working-set size, and low contention. Most transactions have fewer than 100 accesses. Recall that in the working-set analysis shown in Fig. 4.3(a) (for transaction length 100), RTM slightly outperforms TinySTM for working-set sizes up to 4MB. On the other hand, TinySTM outperforms RTM when contention is low (Fig. 4.7(a)). The confluence of these two factors within `genome` yields similar performances for RTM and TinySTM up to four threads. For eight threads, as expected, TinySTM's performance continues to improve, whereas RTM's suffers from increased resource sharing among hyper-threads.

`intruder` is also a high-contention benchmark. As with `genome`, RTM performance scales well from one to four threads. Since `intruder` executes very short transactions, scaling to eight threads does not cause as much resource contention as for `genome`, and thus RTM and TinySTM perform similarly. Even though this application has a small to medium working set — which might otherwise give RTM an advantage — its performance is dominated by very short transaction lengths.

kmeans is a clustering algorithm that groups data items in N-dimensional space into K clusters. As with bayes, our 10 runtimes deviate significantly for the multi-threaded versions. On average, RTM performs better than TinySTM. The short transactions experience low contention, and the small working set has high locality, all of which give RTM a performance advantage over TinySTM. Even though both TM systems show speedups over the sequential runs, synchronizing the kmeans algorithm in TinySTM expends more energy at all thread counts.

labyrinth routes a path in a three-dimensional maze, where each thread grabs a start and an end point and connects them through adjacent grid points. Fig. 4.10 shows that labyrinth does not scale in RTM. This is because each thread makes a copy of the global grid inside the transaction, triggering capacity aborts that eventually cause the fallback to using a lock. Energy expenditure increases for the RTM multi-threaded runs because the threads try to execute the transaction in parallel but eventually fail, wasting many instructions while increasing cache and bus activity.

ssca2 has short transactions, a small read-write set, and low contention, and thus even though it has a large working set, it scales well to higher thread counts. Performance for eight threads is good for both RTM and TinySTM. In general, RTM performs better (with respect to both execution time and energy expenditure) but not by much, as is to be expected for very short transactions.

vacation has low to medium contention among threads and a medium working set size. The transactions are of medium length, locality is medium, and contention is low. Like genome, vacation scales well up to four threads, but performance degrades for eight threads because its read-write set size is large enough that cache sharing causes resource limitation issues.

yada has big working set, medium transaction length, large read-write set, and medium contention. All these conditions give TinySTM a consistent performance advantage over RTM at all thread counts.

Our results in Fig. 4.11 indicate that the energy trends of applications do not always follow their performance trends. Applications like bayes, labyrinth, and yada expend more energy as they are scaled up, even when performance changes little (or even improves, in the case of yada). Only intruder, kmeans, and ssca2 benefit from hyper-threading under RTM. In contrast, most STAMP applications benefit from hyper-threading under TinySTM, and those that do not suffer only small degradations.

Fig. 4.12 shows the overall abort rates for all benchmarks, including the contributions of different abort types. Based on our observations of hardware counter values, the current RTM implementation does not seem to distinguish between data-conflict aborts and aborts caused by read-set evictions from L3 cache, and thus both phenomena are reported as conflict aborts. When a thread incurs the maximum number of failed

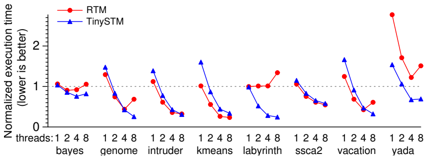


Figure 4.10: *RTM versus TinySTM Performance for STAMP Benchmarks*

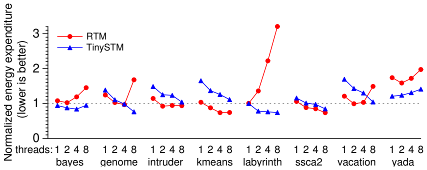


Figure 4.11: *RTM versus TinySTM Energy Expenditure for STAMP Benchmarks*

transactions and acquires the lock in the fallback path, it forces all currently running transactions to abort. We term this a *lock* abort. These aborts are reported either as conflict aborts, *explicit* aborts (i.e., deliberately triggered by the application code), or both (i.e., the machine increments multiple counters). Lock aborts are specific to the fallback mechanism we use in our experiments. Other fallback mechanisms that do not use serialization locks within transactions (they can be employed in non-transactional code) do not incur such aborts. Note that avoiding lock aborts does not necessarily result in better performance since the lock aborts mask other type of aborts (i.e., that would have occurred subsequently). This can be seen in abort contributions shown in the figure. As applications are scaled, the fraction of aborts caused by locks increases because every acquisition potentially triggers N-1 lock aborts (where N is the number of threads).

The `RTM_RETIRED:ABORTED_MISC3` performance counter reports aborts due to events like issuing unsupported instructions, page faults, and page table modifications. The `RTM_RETIRED:ABORTED_MISC5` counter includes miscellaneous aborts not categorized elsewhere, such as aborts caused by interrupts. Table 4.3 gives an overview of these abort types. In addition to these counters, three more performance counters represent categorized abort numbers: `RTM_RETIRED:ABORTED_MISC1` counts aborts due to memory events like data conflicts and capacity overflows; `RTM_RETIRED:ABORTED_MISC2` counts aborts due to uncommon conditions; and `RTM_RETIRED:ABORTED_MISC4` counts aborts due to incompatible memory types (e.g., due to cache bypassing or I/O accesses). In our experiments, `RTM_RETIRED:ABORTED_MISC4` counts are always less than 20, which we attribute to hardware error (as per the Intel specification update [98]). In all our experiments, `RTM_RETIRED:ABORTED_MISC2` is zero.

Abort Type	Description
Data-conflict/ Read-capacity	Conflict aborts and read-set capacity aborts
Write-capacity	Write-set capacity aborts
Lock	Conflict and explicit aborts caused by serialization locks
Misc3	Unsupported instruction aborts[a]
Misc5	Aborts due to none of the previous categories[b]

[a] includes explicit aborts and aborts due to page fault/page table modification

[b] interrupts, etc.

Table 4.3: *Intel RTM Abort Types*

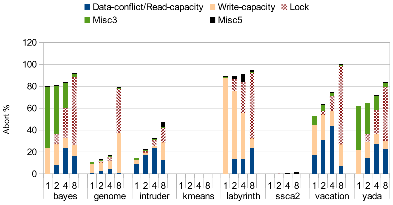


Figure 4.12: *RTM Abort Distributions for STAMP Benchmarks*

4.5 Related Work

Hardware transactional memory systems must track memory updates within transactions and detect conflicts (read-write, write-read, or write-write conflicts across concurrent transactions or non-transactional writes to active locations within transactions) at the time of access. The choice of where to buffer speculative memory modifications has microarchitectural ramifications, and commercial implementations naturally strive to minimize modifications to the the cores and on-chip memory hierarchies on which they are based. For instance, Blue Gene/Q [92] tracks updates in the 32MB L2 cache, and the IBM System z [93] series and the canceled Sun Rock [90] track updates in their store queues. Like the Haswell RTM implementation that we study here, the Vega Azul Java compute appliance [91] uses the L1 cache to record speculative writes. The size of transactions that can benefit from such hardware TM support depends on the capacity of the chosen buffering scheme. Like us, others have found that rewriting software to be more transaction-friendly improves hardware TM effectiveness [91].

Previous studies investigate the characteristics of hardware transactional memory systems. Wang et al. [92] use the STAMP benchmarks to evaluate hardware transactional memory support on Blue Gene/Q, finding that the largest source of TM overhead is loss of cache locality from bypassing or flushing the L1 cache.

Yoo et al. [94] use the STAMP benchmarks to compare Haswell RTM with the TL2 software transactional memory system [99], finding significant performance differences between TL2 and RTM. We perform a similar study and find that TinySTM consistently outperforms TL2, and thus we choose the former as our STM point of comparison. Our RTM scaling results for STAMP benchmark concur with their results.

Wang et al. [100] evaluate RTM performance for concurrent skip list scalability, comparing against competing synchronization mechanisms like fine-grained locking and lock-free linked-lists. They use the Intel RTM emulator to model up to 40 cores, corroborating results for one to eight cores with Haswell hardware experiments. Like us, they highlight RTM performance limitations due to capacity and conflict misses and propose programmer actions that can improve RTM performance.

Others have also studied power/performance trade-offs for TM systems. For instance, Gaona et al. [101] perform a simulation-based energy characterization study of two HTM systems: the LogTM-SE *Eager-Eager* system [86] and the Scalable TCC *Lazy-Lazy* system [102]. Ferri et al. [103] estimate the performance and energy implications of using TM in an embedded multiprocessor system-on-chips (MPSoCs), providing detailed energy distribution figures from their energy models.

In contrast to the work presented here, none of these studies analyzes energy expenditure for a commercial hardware implementation.

4.6 Conclusions

The Restricted Transactional Memory support available in the Intel Haswell microarchitecture makes programming with transactions more accessible to parallel computing researchers and practitioners. In this study, we compare RTM and TinySTM, a software transactional memory implementation, in terms of performance and energy. We highlight RTM's hardware limitations and quantify their effects on application behavior, finding that performance degrades for workloads with large working sets and long transactions. Enabling hyper-threading worsens RTM performance due to resource sharing at the L1 level. We give details about the sources of aborts in a TM application and a way to quantiy these aborts. Using the knowledge presented in this thesis, parallel programmers can optimize TM applications to better utilize the Haswell support for RTM.

5 Conclusion

The significance of power-aware computing technologies is rapidly increasing because of the expanding carbon footprint of ICT sector. Analyzing the power consumption of a system is a fundamental step in formulating power-aware techniques. This thesis proposes methodologies to better enable power-aware research through power measurement, power modeling, and power characterization.

5.1 Contributions

This thesis makes following contributions:

Power Measurement (Chapter 2). We develop an infrastructure to measure power consumption at three points in voltage supply path of the processor: at the wall outlet, at the ATX power rails and at the CPU voltage regulator. We do a qualitative comparison for the measurements sampled from the three points for accuracy and sensitivity. We discuss the advantages and disadvantages of each sampling point. We show that sampling power at the wall outlet is easiest but also least accurate. We test Intel's digital power model (available to the software through Running Average Power Limit (RAPL) interface) for accuracy, overhead and temporal granularity. We demonstrate that the RAPL model estimates power with good mean accuracy but suffers with infrequent high deviations from the mean.

Power Modeling (Chapter 3). We build upon the power model developed by Singh et al. [16, 63] that estimates power consumption of individual cores in chip multiprocessors. We port and validate the model across many platforms, improve model accuracy by augmenting microbenchmark and counter selection methodology, provide analysis for model estimation errors, and show the effectiveness of the model for the meta-scheduler that uses multiple DVFS operating points and process suspension to enforce power budget.

Characterization of Intel's Restricted Transactional Memory (Chapter 4). We present a detailed evaluation of Intel's Restricted Transactional Memory (RTM) performance and energy expenditure. We compare RTM behavior to that of the TinySTM software transactional memory system, first by running microbenchmarks, and then by running the STAMP benchmark suite. We quantify the RTM hardware limitations concerning its read/write-set size, duration and overhead. We find that RTM performs well when transaction working-set fits inside cache. RTM also handles high contention workloads better than TinySTM.

5.2 Future Work

The methodologies presented in this thesis give rise to many future research directions.

The RTM characterization we presented in Chapter 4 is an initial study. We plan to expand this study by exploring techniques to perform energy profiling for transactional benchmarks to help the programmers who will like to do energy-aware code optimizations. The orthogonal characterization used in our work is useful in visualizing the effects of an individual transactional property on the energy expenditure of the system. However, hybrid transactional memory schemes that use both hardware and software transactional memory would need to evaluate all the transactional properties simultaneously, which will be difficult because of huge number of operating points involved. We plan to explore machine learning techniques for helping such hybrid TM schemes in making an HTM versus STM decision.

The STAMP results in Chapter 4 show that energy consumption of benchmarks with large working sets increases even when the performance remains the same or decreases. We intend to explore this further and devise remedial measures, if possible.

Datacenters are major power consumers in the ICT sector and present many opportunities in power-aware computing. We plan to expand our power modeling and characterization approaches to datacenter workloads and explore how datacenter specific characteristics like data replication, data placement, and data volumes affect overall energy expenditure.

Bibliography

[1] The Climate Group, “SMART 2020 : Enabling the Low Carbon Economy in the Information Age,” *GeSI’s Activity Report, The Climate Group on behalf of the Global eSustainability Initiative (GeSI)*, June 2008.

[2] Ryan Heath and Linda Cain, “Digital Agenda: global tech sector measures its carbon footprint,” `http://europa.eu/rapid/press-release_IP-13-231_en.htm`, 2013.

[3] S. Murugesan, “Harnessing green it: Principles and practices,” *IEEE IT Professional*, vol. 10, no. 1, pp. 24–33, 2008.

[4] Gary Cook and Jodie Van Horn, “How Dirty is your Data,” *A Look at the Energy Choices That Power Cloud Computing*, 2011.

[5] Parthasarathy Ranganathan, “Recipe for efficiency: principles of power-aware computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 60–67, 2010.

[6] E. Rotem, A. Naveh, D. Rajwan, A. Ananthadrishnan, and E Weissmann, “Power management architecture of the 2nd generation Intel® Core™ microarchitecture, formerly codenamed Sandy Bridge,” in *Proc. 23rd HotChips Symposium on High Performance Chips*, August 2011.

[7] Advanced Micro Devices, “AMD Opteron™ 6200 series processors linux tuning guide,” 2012, `http://developer.amd.com/wordpress/media/2012/10/51803A_OpteronLinuxTuningGuide_SCREEN.pdf`.

[8] NVIDIA, *Nvidia NVML API Reference Manual*, 2012.

[9] Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang, “Fine-grained power modeling for smartphones using system call tracing,” in *Proc. the sixth conference on Computer systems*. ACM, 2011, pp. 153–168.

[10] R. Joseph and M. Martonosi, “Run-time power estimation in high-performance microprocessors,” in *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, August 2001, pp. 135–140.

[11] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson, "Application-aware power management," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, October 2006, pp. 39–48.

[12] F. Bellosa, S. Kellner, M. Waitz, and A. Weissel, "Event-driven energy accounting for dynamic thermal management," in *Proc. of the Workshop on Compilers and Operating Systems for Low Power (COLP'03)*, Sept. 2003.

[13] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proc. 24th ACM International Conference on Supercomputing*, June 2010, pp. 147–158.

[14] G. Contreras and M. Martonosi, "Power prediction for Intel XScale processors using performance monitoring unit events," in *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, August 2005, pp. 221–226.

[15] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *Proc. Workshop on Modeling, Benchmarking, and Simulation*, June 2006.

[16] K. Singh, M. Bhadauria, and S.A. McKee, "Real time power estimation and thread scheduling via performance counters," November 2008.

[17] R Ren, E Juarez, C Sanz, Mickael Raulet, and F Pescador, "System-level PMC-driven energy estimation models in RVC-CAL video codec specifications," in *Proc. Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE, 2013, pp. 55–62.

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

[19] Krishna Rangan, Gu-Yeon Wei, and David Brooks, "Thread motion: Fine-grained power management for multi-core systems," in *Proc. 36th IEEE/ACM International Symposium on Computer Architecture*, June 2009.

[20] M. Banikazemi, D. Poff, and B. Abali, "PAM: A novel performance/power aware meta-scheduler for multi-core systems," in *Proc. IEEE/ACM Supercomputing International Conference on High Performance Computing, Networking, Storage and Analysis*, November 2008, number 39.

[21] C. Isci, A. Buyuktosunoglu, C.Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. IEEE/ACM 40th Annual International Symposium on Microarchitecture*, December 2006, pp. 347–358.

[22] Ke Meng, Russ Joseph, Robert P. Dick, and Li Shang, "Multi-optimization power management for chip multiprocessors," in *Proc. the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 177–186.

[23] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: A framework for continuously adaptive dvfs," in *Proc. Green Computing Conference and Workshops (IGCC), 2011 International*, July 2011, pp. 1–8.

[24] Kyong Hoon Kim, Rajkumar Buyya, and Jong Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled Clusters," in *CCGRID*, 2007, vol. 7, pp. 541–548.

[25] Josep Ll Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres, "Towards energy-aware scheduling in data centers using machine learning," in *Proc. the 1st International Conference on energy-Efficient Computing and Networking*. ACM, 2010, pp. 215–224.

[26] Gregor Von Laszewski, Lizhe Wang, Andrew J Younge, and Xi He, "Power-aware scheduling of virtual machines in dvfs-enabled clusters," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[27] Tridib Mukherjee, Ayan Banerjee, Georgios Varsamopoulos, Sandeep KS Gupta, and Sanjay Rungta, "Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers," *Computer Networks*, vol. 53, no. 17, pp. 2888–2904, 2009.

[28] Qinghui Tang, Sandeep KS Gupta, and Georgios Varsamopoulos, "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1458–1472, 2008.

[29] Xin Zhan and Sherief Reda, "Techniques for energy-efficient power budgeting in data centers," in *Proc. the 50th Annual Design Automation Conference*, 2013, pp. 176:1–176:7.

[30] Victor Jimenez, Francisco Cazorla, Roberto Gioiosa, Eren Kursun, Canturk Isci, Alper Buyuktosunoglu, Pradip Bose, and Mateo Valero, "Energy-aware accounting and billing in large-scale computing facilities," *IEEE Micro*, vol. 31, no. 3, pp. 60–71, 2011.

[31] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Henrdrik Hamann, Alan Weger, and Pradip Bose, "Thermal-aware task scheduling at the system software level," in *Proc. the 2007 international symposium on Low power electronics and design*. ACM, 2007, pp. 213–218.

[32] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Keith Whisnant, "Temperature aware task scheduling in mpsocs," in *Proc. the conference on Design, automation and test in Europe*. EDA Consortium, 2007, pp. 1659–1664.

[33] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof," in *Proc. the 7th ACM European Conference on Computer Systems*, 2012, EuroSys '12, pp. 29–42.

[34] K.S. Banerjee and E. Agu, "PowerSpy: fine-grained software energy profiling for mobile devices," in *Proc. 2005 International Conference on Wireless Networks, Communications and Mobile Computing*, 2005, vol. 2, pp. 1136–1141 vol.2.

[35] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications*, February 1999, pp. 2–10.

[36] Y.S. Shao and D. Brooks, "Energy characterization and instruction-level energy model of intel's xeon phi processor," in *Proc. 2013 IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 389–394.

[37] Ramon Bertran, Alper Buyuktosunoglu, Meeta S. Gupta, Marc Gonzalez, and Pradip Bose, "Systematic energy characterization of CMP/SMT processor systems via automated micro-benchmarks," in *Proc. the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2012, MICRO-45, pp. 199–211, IEEE Computer Society.

[38] T. Mukherjee, G. Varsamopoulos, S. K S Gupta, and S. Rungta, "Measurement-based power profiling of data center equipment," in *2007 IEEE International Conference on Cluster Computing*, 2007, pp. 476–477.

[39] Epifanio Gaona, Rubén Titos-Gil, Juan Fernandez, and Manuel E. Acacio, "Characterizing energy consumption in hardware transactional memory systems," in *Proc. 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2010, pp. 9–16.

[40] Y. Guo, S. Chheda, I. Koren, C. M. Krishna, and C. A. Moritz, "Energy Characterization of Hardware-Based Data Prefetching," in *Proc. IEEE International Conference on Computer Design*, 2004, pp. 518–523.

[41] Marc Gamell, Ivan Rodero, Manish Parashar, Janine C. Bennett, Hemanth Kolla, Jacqueline Chen, Peer-Timo Bremer, Aaditya G. Landge, Attila Gyulassy, Patrick McCormick, Scott Pakin, Valerio Pascucci, and Scott Klasky, "Exploring power behaviors and trade-offs of in-situ data analytics," in *Proc. SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2013, SC '13, pp. 77:1–77:12.

[42] John C. McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C. Snoeren, and Rajesh K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proc. the 2011 USENIX Conference on USENIX Annual Technical Conference*, 2011, pp. 12–12.

[43] B. Goel, S.A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati, "Portable, scalable, per-core power estimation for intelligent resource management," in *Proc. 1st International Green Computing Conference*, August 2010, pp. 135–146.

[44] C. Sun, L. Shang, and R.P. Dick, "Three-dimensional multiprocessor system-on-chip thermal optimization," in *Proc. 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2007, pp. 117–122.

[45] E. Stahl, "Power Benchmarking: A new methodology for analyzing performance by applying energy efficiency metrics," White Paper, IBM, 2006.

[46] Standard Performance Evaluation Corporation, "SPECpower_ssj2008 benchmark suite," `http://www.spec.org/power_ssj2008/`, 2008.

[47] B. Goel, R. Titos-Gil, A. Negi, S.A. McKee, and P. Stenstrom, "Performance and energy analysis of the restricted transactional memory implementation on haswell," in *Proceedings of the Proc. International Parallel and Distributed Processing Symposium*, 2014, To appear.

[48] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig, "Measuring energy consumption for short code paths using RAPL," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, pp. 13–17, January 2012.

[49] Electronic Educational Devices, "Watts Up PRO," `http://www.wattsupmeters.com/`, May 2009.

[50] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *Proc. 14th IEEE International Symposium on High Performance Computer Architecture*, February 2008, pp. 101–110.

[51] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "Powermon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proc. IEEE SoutheastCon 2010 (SoutheastCon)*, Mar. 2010, pp. 479–484.

[52] Server System Infrastructure Forum, *EPS12V Power Supply Design Guide*, Dell, HP, SGI, and IBM, 2.92 edition, 2006.

[53] LEM Corporation, "Intel current transducer LTS 25-NP," Datasheet, LEM, November 2009.

[54] National Instruments Corporation, "NI bus-powered m series multifunction daq for usb," `http://sine.ni.com/ds/app/doc/p/id/ds-9/lang/en`, Apr. 2009.

[55] Intel Corporation, "Voltage regulator-down (VRD) 11.1," Design Guidelines, Intel Corporation, Sept. 2009.

[56] Intel, *Intel Architecture Software Developer's Manual: System Programming Guide*, June 2013.

[57] D. Hackenberg, T. Ilsche, R. Schone, D. Molka, M. Schmidt, and W.E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," in *Proc. 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 194–204.

[58] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. IEEE/ACM 37th Annual International Symposium on Microarchitecture*, 2003, pp. 93–104.

[59] Z. Cui, Y. Zhu, Y. Bao, and M. Chen, "A fine-grained component-level power measurement method," in *Proc. 2nd International Green Computing Conference*, July 2011, pp. 1–6.

[60] Intel Corporation, "Intel Core$^{\text{TM}}$ i7 Processor," `http://www.intel.com/products/processor/corei7/`, December 2008.

[61] M.S. Floyd, S. Ghiasi, T.W. Keller, K. Rajamani, F.L. Rawson, J.C. Rubio, and M.S. Ware, "System power management support in the IBM POWER6 microprocessor," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 733–746, 2007.

[62] H.-Y. McCreary, M.A. Broyles, M.S. Floyd, A.J. Geissler, S.P. Hartman, F.L. Rawson, T.J. Rosedahl, J.C. Rubio, and M.S. Ware, "Energyscale for IBM POWER6 microprocessor-based systems," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 775–786, 2007.

[63] K. Singh, *Prediction Strategies for Power-Aware Computing on Multicore Processors*, Ph.D. thesis, Cornell University, 2009.

[64] V.M. Weaver and S.A. McKee, "Can hardware performance counters be trusted?," in *Proc. IEEE International Symposium on Workload Characterization*, Sept. 2008, pp. 141–150.

[65] D. Zaparanuks, M. Jovic, and M. Hauswirth, "Accuracy of performance counter measurements," Tech. Rep. USI-TR-2008-05, Università della Svizzera italiana, Sept. 2008.

[66] V.M. Weaver and J. Dongarra, "Can hardware performance counters produce expected, deterministic results," in *In Proc. 3rd Workshop on Functionality of Hardware Performance Monitoring*, December 2010.

[67] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual*, May 2013.

[68] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *IEEE Computer*, vol. 36, no. 12, pp. 68–75, December 2003.

[69] M. Berktold and T. Tian, "CPU monitoring with DTS/PECI," White Paper, Intel Corporation. `http://download.intel.com/design/intarch/papers/322683.pdf`, Sept. 2010.

[70] C. Spearman, "The proof and measurement of association between two things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, January 1904.

[71] Standard Performance Evaluation Corporation, "SPEC CPU benchmark suite," `http://www.specbench.org/osg/cpu2006/`, 2006.

[72] Standard Performance Evaluation Corporation, "SPEC OMP benchmark suite," `http://www.specbench.org/hpg/omp2001/`, 2001.

[73] V. Aslot and R. Eigenmann, "Performance characteristics of the SPEC OMP2001 benchmarks," in *Proc. European Workshop on OpenMP*, Sept. 2001.

[74] D.H. Bailey, T. Harris, W.C. Saphir, R.F. Van der Wijngaart, A.C. Woo, and M. Yarrow, "The NAS parallel benchmarks 2.0," Report NAS-95-020, NASA Ames Research Center, December 1995.

[75] S. Eranian, "Perfmon2: A flexible performance monitoring interface for Linux," in *Proc. 2006 Ottawa Linux Symposium*, July 2006, pp. 269–288.

[76] C. Boneti, R. Gioiosa, F.J. Cazorla, and M. Valero, "A dynamic scheduler for balancing HPC applications," in *Proc. IEEE/ACM Supercomputing International Conference on High Performance Computing, Networking, Storage and Analysis*, November 2008, number 41.

[77] E. Betti, M. Cesati, R. Gioiosa, and F. Piermaria, "A global operating system for HPC clusters," in *Proc. IEEE International Conference on Cluster Computing*, August 2009, p. 6.

[78] Advanced Micro Devices, *AMD Athlon Processor Model 6 Revision Guide*, 2003.

[79] T. Austin, "SimpleScalar 4.0 release note," `http://www.simplescalar.com/`.

[80] W. Wu, L. Jin, and J. Yang, "A systematic method for functional unit power estimation in microprocessors," in *Proc. 43rd ACM/IEEE Design Automation Conference*, July 2006, pp. 554–557.

[81] A. Merkel and F. Bellosa, "Balancing power consumption in multicore processors," in *Proc. ACM SIGOPS/EuroSys European Conference on Computer Systems*, Apr. 2006, pp. 403–414.

[82] E. Grochowski, R. Ronen, J. Shen, and H. Wang, "Best of both latency and throughput," in *Proc. IEEE International Conference on Computer Design*, October 2004, pp. 236–243.

[83] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's Law through EPI throttling," in *Proc. 32nd IEEE/ACM International Symposium on Computer Architecture*, June 2005, pp. 298–309.

[84] M. Moudgill, P. Bose, and J. Moreno, "Validation of Turandot, a fast processor model for microarchitecture exploration," in *Proc. International Performance, Computing, and Communications Conference*, February 1999, pp. 452–457.

[85] Maurice Herlihy and J. Eliot B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proceedings of the 20th International Symposium on Computer Architecture*, 1993, pp. 289–300.

[86] Luke Yen, Jayaram Bobba, Michael R. Marty, Kevin E. Moore, Haris Volos, Mark D. Hill, Michael M. Swift, and David A. Wood, "LogTM-SE: Decoupling hardware transactional memory from caches," in *Proceedings of the 13th Symposium on High-Performance Computer Architecture*, 2007, pp. 261–272.

[87] Lance Hammond, Vicky Wong, Mike Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun, "Transactional memory coherence and consistency," in *Proceedings of the 31st International Symposium on Computer Architecture*, 2004, pp. 102–113.

[88] Marc Lupon, Grigorios Magklis, and Antonio González, "A dynamically adaptable hardware transactional memory," in *Proceedings of the 43rd International Symposium on Microarchitecture*, 2010, pp. 27–38.

[89] Rubén Titos-Gil, Anurag Negi, Manuel E. Acacio, Jose M. Garcia, and Per Stenstrom, "Zebra : A data-centric, hybrid-policy hardware transactional memory design," in *Proceedings of the 25th International Conference of Supercomputing*, 2011, pp. 53–62.

[90] Dave Dice, Yossi Lev, Mark Moir, and Daniel Nussbaum, "Early experience with a commercial hardware transactional memory implementation," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 157–168, Mar. 2009.

[91] Cliff Click, "Azul's experiences with hardware transactional memory," 2009, http://sss.cs.purdue.edu/projects/tm/tmw2010/talks/Click-2010_TMW.pdf.

[92] Amy Wang, Matthew Gaudet, Peng Wu, Jose Nelson Amaral, Martin Ohmacht, Christopher Barton, Raul Silvera, and Maged Michael, "Evaluation of Blue Gene/Q hardware support for transactional memories," in *Proc. the 21st International Conference on Parallel Architectures and Compilation Techniques*, 2012, pp. 127–136.

[93] Christian Jacobi, Timothy Slegel, and Dan Greiner, "Transactional memory architecture and implementation for IBM System z," in *Proc. the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 25–36.

[94] Richard M. Yoo, Christopher J. Hughes, Konrad Lai, and Ravi Rajwar, "Performance evaluation of Intel transactional synchronization extensions for high-performance computing," in *Proc. SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 19:1–19:11.

[95] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun, "STAMP: Stanford transactional applications for multi-processing," in *Proceedings of the IEEE International Symposium on Workload Characterization*, 2008, pp. 35–46.

[96] Sungpack Hong, Tayo Oguntebi, Jared Casper, Nathan Bronson, Christos Kozyrakis, and Kunle Olukotun, "Eigenbench: A simple exploration tool for orthogonal TM characteristics," in *Proc. the IEEE International Symposium on Workload Characterization*, 2010, pp. 1–11.

[97] Pascal Felber, Christof Fetzer, Patrick Marlier, and Torvald Riegel, "Time-based software transactional memory," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1793–1807, 2010.

[98] Intel, *Desktop 4th Generation Intel Core^{TM}Processor Family Specification Update*, August 2013.

[99] David Dice, Ori Shalev, and Nir Shavit, "Transactional Locking II," in *Proc. the 19th International Symposium on Distributed Computing*, 2006.

[100] Zhaoguo Wang, Hao Qian, Haibo Chen, and Jinyang Li, "Opportunities and pitfalls of multi-core scaling using hardware transaction memory," in *Proc. the 4th Asia-Pacific Workshop on Systems*, 2013, pp. 3:1–3:7.

[101] E. Gaona-Ramirez, R. Titos-Gil, J. Fernandez, and M.E. Acacio, "Characterizing energy consumption in hardware transactional memory systems," in *Proc. 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2010, pp. 9–16.

[102] Hassan Chafi, Jared Casper, Brian D. Carlstrom, Austen McDonald, Chi Cao Minh, Woongki Baek, Christos Kozyrakis, and Kunle Olukotun, "A scalable, non-blocking approach to transactional memory," in *Proceedings of the 13th Symposium on High-Performance Computer Architecture*, 2007, pp. 97–108.

[103] C. Ferri, R.I. Bahar, A. Marongiu, L. Benini, M. Herlihy, B. Lipton, and T. Moreshet, "SoC-TM: Integrated HW/SW support for transactional memory programming on embedded MPSoCs," in *Proc. the 9th International Conference on Hardware/Software Codesign and System Synthesis*, 2011, pp. 39–48.