CHALMERS

Per-core Power Estimation and Power Aware Scheduling Strategies for CMPs

Master of Science Thesis in Integrated Electronic System Design

BHAVISHYA GOEL

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, January 2011

Per-core Power Estimation and Power Aware Scheduling Strategies for CMPs

BHAVISHYA GOEL

Examiner: SALLY A. McKEE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden January 2011

Contents

List of Figures

List of Tables

List of Abbreviations

APIC	Advanced Programmable Interrupt Controller
ATX	Advanced Technology Extended
BTB	Branch Target Buffer
CDF	Cumulative Distribution Function
CMP	Chip Multiprocessor
DVFS	Dynamic Voltage Frequency Scaling
HPC	High-Performance Computing
MSR	Model Specific Register
NAS	NASA Advanced Supercomputing
OMP	OpenMP
PCB	Printed Circuit Board
PMC	Performance Monitoring Counter
PMU	Performance Monitoring Unit
ROB	Re-order Buffer
RS	Reservation Station
SIMD	Single Instruction Multiple Data
SPEC	Standard Performance Evaluation Corporation
SSE	Streaming SIMD Extensions
TDP	Thermal Design Power
UOPS	Micro-operations

Abstract

The problem of accurately estimating the processor power consumption has generated significant interest among computer architects in the last decade. With the focus on green computing intensifying, increasing number of task management applications have become power aware in last few years. Hence, the need for a fast and accurate power model is greater than ever. In addition, today's multi-core processors demand task schedulers to balance the performance requirements, power budget and thermal constraints. This thesis addresses this requirement by presenting a per-core power model based upon performance monitoring counters and temperature data. PMC based power models provide a straightforward and fast way of analyzing the activity of processor's underlying microarchitecture. The advantage of our model is that it is general enough to be ported and scaled across different platforms with ease, fast enough to be used online by task schedulers, and it requires no knowledge of individual applications. During this thesis work, we validated the model on three different (two- to eight-core) platforms. The model accurately estimates core power consumption, exhibiting 1.8%-4.8% per-suite median error on the NAS , SPEC OMP , and SPEC 2006 benchmarks (and 1.6%-4.4% overall).

Acknowledgements

First and foremost, I will like to thank my adviser Prof. Sally A. McKee. Her technical guidance and project management skills helped me at every step of my thesis. I hope that some of her skills had brushed off me during this period. I will also like to thank Karan A. Singh whose PhD thesis was extended into this work. He was very helpful in getting me started and resolving my queries about his work. I will like to thank Anders Widen, Chen Guancheng and Jacob Lidman for giving me valuable inputs for my work. Their company made my thesis period enjoyable. I also thank Major Bhadauria, Roberto Gioiosa and Marco Cesati for the extremely valuable work that contributed to the completion of this thesis. Last, but not the least, I will like to thank Magnus Själander for his insightful comments about my work.

Chapter 1

Introduction

This chapter provides a brief introduction to the problem addressed by this thesis, the previous work and the outline of this report.

1.1 Problem

In today's computing environment, power consumption is becoming an all-important metric that decides the design and performance specifications of a system [31]. A balance between power consumption and performance requirements is of highest importance to achieve the most judicious use of available resources in a system. To make power aware decisions, system resource managers require information about the power consumption and temperature in real time, for individual resources if available.

In a chip multiprocessor, at any given point of time, different cores may be executing different applications with a potentially wide range of power consumption figures. In the given scenario, an estimate of per core power consumption can prove to be a useful metric for consideration by the task scheduler to make power aware scheduling decisions. Such a power aware scheduler can be used to achieve a desired balance between power consumption and performance.

But as things stand now, the support from the chip manufacturers for providing accurate power consumption information for individual cores to the operating system is non-existent. Most current hardware lack the on-die infrastructure for sensing current consumption because of hardware costs and the intrusive nature of the current sensing techniques. And even when such hardware exists, the information is not available to the operating system. For example, the Core i7 [11] processor from Intel employs the power consumption monitoring hardware on-chip to enable it's Turbo Boost technology. But this information is available to and used only by the hardware for pushing chip performance without ever passing it on to the software.

The hardware resources like a power meter can be employed to measure total system power. Other current measuring equipments like ammeter and digital multimeter can be used for isolating the CPU power consumption from the system power. But hardware infrastructure that enables the user

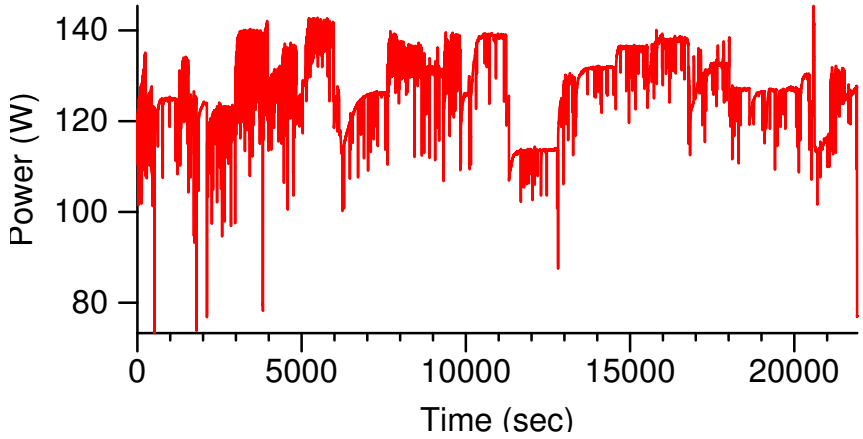


Figure 1.1: Core i7 System Power Consumption for NAS, SPEC2006 & SPEC-OMP

to measure per core power consumption is unavailable at present since the cores in current CMP designs share the power plane.

System simulators [10] can be used to obtain detailed and decomposable information about the power consumption of devices based on the processor activity for a given application. But the downside of these simulators is that they can be very time consuming to use, and obtaining the accurate power models for off-the-shelf commercial processors can be very difficult, if not impossible. Also, the architectural power models used in these simulators are prone to error [21]. But probably, the biggest disadvantage of these simulators is that they are of little use for online power estimation, especially when the running application and/or the input dataset is unknown.

A viable alternative to core level power monitoring hardware and system simulators is to use the core specific information available to the software dynamically and estimate the power consumption using this information.

Some previous works [7] have proposed simplistic power models. These works assume that all the instructions consume same amount of power. Hence, the power consumption can be estimated by dividing the CPU time between idle/stalled and active and assigning a fixed power value to the two sections of CPU time as in Equation 1.1.

$$CorePower = \eta * P_{active} + (1 - \eta) * P_{idle} \tag{1.1}$$

But if we take a look at CPU power consumption on a Core i7 machine across a run constituting different benchmark suites as shown in Figure 1.1, we can see that even when the processor is active all the time, there can be huge variations in the power consumption.

A modern processor has a very complex microarchitecture. Different applications at different times would exercise different sections of this microarchitecture. Hence a simplistic power model of 1.1 will prove to be insufficient. This is further established by the Figure 1.1 that shows the variations in the CPU power consumption when different microarchitectural sections are exercised on Core i7. The variations would be even larger when these microarchitectural components are

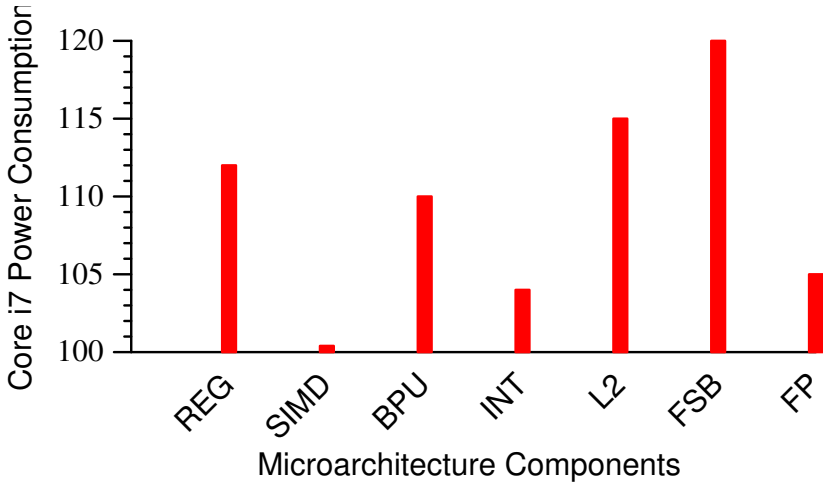


Figure 1.2: Variations in Core i7 Processor Power Consumption when different processor components are exercised

exercised in various combinations.

This enhances a case for a more detailed power model that can estimate the power consumption with reasonable accuracy even when application exercises different components of the microarchitecture in different phases of it's run.

This thesis presents a portable and scalable per-core power estimation model that enables the power aware resource managers to make scheduling decisions based upon core level power consumption information.

1.2 Previous Work and Contribution

This thesis builds upon the work done by Singh et al. [34, 35], which uses hardware performance monitoring counters and core temperature data to create a per core statistical power estimation model. They validated the model on three different platforms, namely Intel Q6600, AMD Phenom and Intel E5430 with median accuracy of less than 7% across all platforms. They have also demonstrated the usage of the power model in a live power management application to schedule processes under the constraint of a given system power budget.

For this thesis, we have validated the existing model on three more platforms, namely Intel Core Duo, AMD Opteron 8212 and Intel Core i7. We automated the data analysis process, analyzed the error sources, explored an alternate model for power estimation, and created a custom test benchmark to analyze the model performance in various corner conditions. We also augmented the scheduler code for scalability and modularity, introduced two new scheduling policies and implemented new scalable DVFS approach in the scheduler.

1.3 Report Outline

In chapter 2, we give a brief introduction to the theoretical aspects of our work. We explain the types of performance counters available, their reading mechanism, and introduce the tools available for reading performance counters. This chapter also introduces the statistical techniques that we use for this thesis work. We introduce the benchmark suites that we use for model evaluation at the end of this chapter.

In chapter 3, we explain the methodology that we use for making the power estimation model and present the evaluation results. We start with describing the setup of our experiments, our test machines, the software tools that we use for statistical analysis, and the microbenchmarks that we use as training set. We then explain procedure for counter selection, model formation and validation. In this chapter, we also discuss the effect of temperature, and various power reduction and performance boosting techniques on the accuracy of the model like DVFS, Turbo Boost, and Hyper-Threading.

In chapter 4, we present a live power management application that we use as proof-of-concept for exploring our model usage. We explain various scheduling policies that we use for power aware scheduling and present the results.

In chapter 5, we discuss the significant contributions made by research work related to our thesis work. We discuss the pros and cons of the research papers and compare them with our work.

In chapter 6, we discuss the research work that will be carried out related to thesis work in future.

In chapter 7, we conclude this thesis. We reiterate the contributions, discuss the results and state the significance of our research work.

Chapter 2

Background Information

This chapter briefly introduces some of the basic concepts that are used in this thesis.

2.1 Performance Monitoring Counters

In this thesis work, we use performance monitoring counters to estimate power consumption. This section gives a brief introduction to the performance monitoring counters for x86 architecture.

Most of the modern processors are equipped with a Performance Monitoring Unit that helps the software programmers in analyzing the performance of the processor and the interaction between the program and the microarchitecture. The PMUs provide a wide variety of performance events that can be sampled and mapped to limited set of Performance Monitoring Counters. The PMCs are accessible as Model Specific Registers by the operating system. The MSRs are control registers that are supported on a finite range of processor implementations. Hence, PMCs act as ideal hooks for peeking into the level of activity of certain microarchitectural components. The programmers can use this information to identify the performance bottlenecks and opportunities to speed up their applications. The PMCs can be used for sampling and investigating events like cache misses, micro-ops retired, stalls at various stages of out-of-order pipeline, floating point/memory/branch operations executed, and many more. Although, the PMC counts are not error-free [40, 41], if used correctly, the errors are small enough to make PMCs suitable candidates for estimating the power consumption. For the modern CMP systems, the PMCs are available individually for each core and hence can be used for making core specific models.

The number and variety of PMCs available for modern processors is increasing with each new architecture. For example, the number of PMCs available for use in Intel Core Duo processor was 34 while the Intel Core i7 processor has around 450 events [22] that can be counted using PMCs. This comprehensive coverage of event information increases the chances that the PMCs available for the performance and power analysis will be good representatives of overall microarchitectural activity.

2.1.1 Types of PMCs

There are two basic types of PMCs, architectural and non-architectural PMCs.

- **Architectural:** This category of PMCs behave consistently across processor models that belong to the same instruction set architecture. The architectural PMCs are usually small in numbers. The examples of this category of PMCs include Unhalted Core cycles, Instructions Retired, and Last Level Cache Misses.
- **Non-Architectural:** This category of PMCs are specific to a particular microarchitecture. The availability and behavior of these PMCs *may* change between different processor models. A larger variety of events can be monitored using non-architectural PMCs compared to architectural PMCs. Examples include Micro-operations Retired, Number of Loads Dispatched, and Resource Stalls.

Both these types of PMCs can be read through the same set of MSRs.

2.1.2 Reading mechanism

As mentioned above, PMCs are mapped to the MSRs of the processor for reading by the software. MSRs mapping PMCs can be classified as general purpose PMCs and fixed PMCs. The general purpose PMC registers are identified as IA32_PMCx, and they can be used for counting any of the events available for monitoring. The fixed PMC registers are identified as IA32_FIXED_CTRx and they are mapped to fixed events. Most of the processors have either two or four MSRs mapped as general-purpose PMCs. This limits the number of events that can be monitored using performance counters simultaneously. The PMCs can be written or read using the instructions used for general MSRs: WRMSR for writing MSR registers and RDMSR for reading. However these instructions are executable only from privilege level 0 of the operating system. PMCs can also be read using an instruction RDPMC. This instruction is available from privilege levels other than 0 when Performance Counter Enable (PCE) bit in Control Register 4 (CR4) is set.

The event selection and configuration of general purpose PMCs is done using PERFEVTSELx (performance event select) MSRs. The bit structure of PERFEVTSELx on processors supporting architectural performance monitoring version 3 is shown in figure 2.1.2.

The description of the fields shown in the layout for PERFEVTSELx register in figure 2.1.2 is as follows:

- **Event Select:** These bits select the logic unit of PMU
- **UMASK:** These bits select the event within the logic unit
- **CMASK:** These bits select the threshold for counting the event. When CMASK is non-zero, the counter is incremented by one during the cycles when the count of occurrences of event being monitored was greater than or equal to value of CMASK. When CMASK is zero, the counter is incremented each cycle by the count of event occurrences.

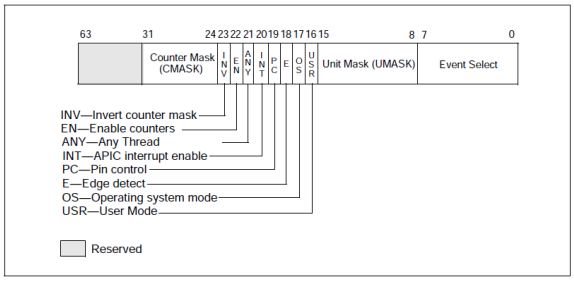


Figure 2.1: Layout of IA32_PERFEVTSELx MSR as appears in [22]

- **EN:** This is performance counter enable bit.
- **INT:** When set, the logical processor generates an exception through its local APIC on counter overflow.
- **PC:** When set, the logical processor toggles the external PM*i* pin and increments the counter on every occurrence of performance-monitoring events; when clear, the processor toggles the pin only when the counter overflows.
- **E:** This bit enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields. The mechanism does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).

For further reading, please refer to Intel's *System Programming Guide* for Intel 64 and IA-32 architectures.

2.1.3 Performance counter monitoring tools

Various tools are available for reading performance counter values. Some of them are listed below:

- **OProfile:** OProfile is a system profiler for Linux systems. It leverages the hardware performance counters for performance profiling of the applications. This tool calls perfmon library to read the performance counter values. OProfile is part of all the major Linux distributions and supports all major platforms.
- **VTune:** VTune is a commercial performance analyzer from Intel. It can do software performance analysis on x86 and x64 machines. It includes a performance counter monitor for identifying system level performance issues.

- **Pfmon:** Pfmon is a simple performance monitoring tool that can collect counter values for a particular thread or system-wide. Pfmon uses perfmon2 library for accessing the performance counters and exhibits low overhead. It requires a kernel patch.

We use Pfmon to access performance counters in our experiments because of its ease of use and small footprint.

2.2 Statistical Modeling

In this thesis work, we use statistical modeling for creating our power estimation model. Statistical modeling techniques are desirable because compared to the analytical modeling, the time and resources required to study and analyze the underlying processor microarchitecture is greatly reduced. Moreover, if the parameters used in the statistical model are generalized, like performance counters, the methodology can prove to be easily portable across various processor models. Statistical modeling can be used in the cases where large number of empirical data can be collected to study the relation between the predictor variables and the outcome, which is true in our case. This section gives a brief introduction to the statistical techniques we use for forming the model.

2.2.1 Rank correlation

Correlation can be defined as the measure of statistical dependence between two random variables. Correlations can be useful for selecting the best candidates among the available predictor variables for defining a predictive relationship between predictors and outcome. In our work, we select the performance counters that are most highly correlated with empirical power consumption. The degree of correlation is defined by the correlation coefficient. There are various types of correlation coefficients. The Pearson's product-moment correlation coefficient is only sensitive to the linear dependence among the random variables. On the other hand, the rank correlation coefficients is sensitive to non-linear relationships of the variables. The rank correlation coefficients measure the tendency of one variable to increase when the other variable increases. Unlike Pearson's correlation coefficient, the rank correlation coefficient's value does not get adversely affected if the relationship between the two variables is nonlinear. For example, consider the equations, 2.1 and 2.2. For the equation 2.1, the correlation coefficients would be perfect 1 for both Pearson's and rank correlation. But for the second equation, rank correlation coefficient would still be 1 but not Pearson's.

$$y = a * x \tag{2.1}$$

$$y = a^x \tag{2.2}$$

The rank correlation coefficient can be calculated by two methods: Spearman's rank correlation and Kendall Tau's rank correlation.

The rank correlation is preferable for our methodology because the relationship between the performance counter values and the power consumption is frequently nonlinear.

2.2.2 Multiple linear regression

In statistics, regression analysis is used to predict a continuous dependent variable from one or more independent variables. As an example, in Equation 2.3, y is the dependent variable and $x_i(i\epsilon[1,n])$ are independent variables. The regression model relates y to a function of $x_i(i\epsilon[1,n])$. Before we go ahead with the regression analysis, we need to specify the form of function f. For example, f can be linear, exponential, power, gaussian, or logarithmic. We chose linear regression analysis for our work since previous research works [12, 26, 27, 29] have shown good results by using linear models to relate architectural events to CPU power consumption and because of the simplicity of linear regression model.

$$y = f(x_1, x_2, \cdots, x_n) \tag{2.3}$$

In linear regression analysis, the dependent variable y is linear combination of independent variables x_i. So, for example, the Equation 2.3 would take the form as in Equation 2.4. Here, ϵ is the total residue collected for all x_i that deviate from the linear regression path. The linear regression analysis calculates the suitable weights for each independent variable such that the resulting ϵ is minimum.

$$y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \cdots + \beta_n * x_n + \epsilon \tag{2.4}$$

2.3 Test Benchmarks

This section gives a brief introduction of the test benchmarks used in this thesis work for model validation.

2.3.1 SPEC CPU2006

SPEC CPU2006 is CPU-intensive single-threaded benchmark suite. The applications included in this benchmark suite stress system processor both memory subsystem. The suite includes both floating point and integer applications. We use a total 26 applications from this suite, twelve of which are integer arithmetic intensive and the rest are floating point computations intensive.

2.3.2 SPEC OMP2001

SPEC OMP2001 is a multi-threaded benchmark suite consisting of OpenMP based applications. This benchmark suite is targeted towards measuring the performance of shared memory systems.

Like SPEC CPU2006, this suite also exercises the system processor and memory. We use a total 10 applications from this suite.

2.3.3 NAS

NAS Parallel Benchmarks are a set of applications designed to test the performance of parallel supercomputers, provided by the NASA Advanced Supercomputing (NAS) division. The benchmarks are derived from Computational Fluid Dynamics(CFD) applications. The applications come with three different size of input data sets, Class A, Class B and Class C, with Class C involving maximum amount of work. We use Class B flavor of benchmark applications for our work. We use a total of 9 applications from this suite.

Chapter 3

Power Estimation Model

This chapter describes our experimental setup, the methodology we use to select the counters, the process of model formation and presents the results from the validation of our power model against various test benchmark suites.

3.1 Overview

In this thesis work, we leverage the event driven performance monitoring counters of the modern processors to make a representative power estimation model. Since the performance monitoring counters represent the activity factor of architectural components, we can assign appropriate weights to appropriate counters using statistical analysis and estimate the processor power consumption. During the training phase of statistical analysis, we run application independent microbenchmarks on each core to exercise various microarchitectural components. During this run, we sample all the significant performance counters available for that processor model and the related empirical power consumption values for every core. We then select four counters that are most highly correlated with empirical power values and provide comprehensive and non-redundant information. We use the sampled data from these four counters along with the related core temperature values as predictors and empirical power values as response variable in multiple regression analysis to form a linear regression model. This model acts as our per-core power estimation model. We evaluate the accuracy of our model against different single- and multi-threaded benchmark suites.

3.2 Experimental Setup

This section describes our experimental setup, apparatus and software tools we use to successfully perform the experiments. As described in the overview section above, we need a mechanism to measure per-core empirical power measurement, read the performance counters and perform correlation and multiple regression analysis.

3.2.1 Empirical power measurement

For our experiments, we used an external power meter from Wattsup Pro [17] to measure the actual system power consumption. The idea behind using an external power meter is to be as non-intrusive as possible. This meter is accurate to within $\pm 1.5\%$, and has the maximum sampling rate of one sample per second. We are thus limited to validating the estimated power values with actual power consumption at the granularity of one second. Please note that this limitation is only applicable for the validation stage. The model itself can be used for power calculation at much higher granularity. The exploration of hardware apparatus that can sample actual power consumption at higher granularity and accuracy is part of future work.

We approximate the empirical per core power consumption by subtracting the uncore power from the CPU power and dividing the residual by number of cores. Since none of the benchmark applications we use are I/O bound, we assume that the uncore power sans power consumed by DRAM remains constant[1]. Hence, we can approximate the uncore power by subtracting the idle CPU power from the idle system power. The idle CPU power values are frequently available from tech sites such as *Anandtech* and *Tom's Hardware*. But when these values are not available, to calculate the value of idle CPU power, we first observe the idle system power and then remove the hardware components like graphics card, Ethernet controller, hard disk and RAM sticks to get an approximate value of CPU power. We ignore the fact that power value we get is the power being sourced from the supply line. Taking into account the efficiency of power supply unit of the motherboard and the on-board voltage regulators, the actual power being consumed by the CPU would be much less. Hence, the sum of the absolute per core values presented in this work may at times exceed the thermal limit of CPU. This however doesn't affect the relative accuracy of our model since the power values estimated target the supply power consumption values instead of actual. If one is interested in actual CPU power consumption values, he/she can do that easily if the efficiency figures of power supply unit of the system and on-board voltage regulators are known. The power consumption values obtained from the power meter are used only during the procedure of model formation and model validation. Once the model is formed, the power meter is no longer required for core power estimation.

3.2.2 Reading performance counters and core temperature

We use perfmon2 library [18] and *pfmon* tool to read the performance counter values. We use *sensors* utility provided by the *lm-sensors* driver to read the core temperature from on-die thermal sensors.

3.2.3 Statistical analysis

We use R package [32] for calculating correlation of individual counters and STAT package [19] for multiple regression analysis.

[1]Please note that any change in the DRAM power consumption is part of the model estimates

	Intel Core Duo	Intel Core i7	AMD Opteron 8212
Cores/Chips	2, dual-core	4, quad-core	8, quad dual-core
Frequency (GHz)	1.667	2.93	2.0
Process (nm)	65	45	90
L1 Instruction	32 KB 8-way	64 KB 2-way	64 KB 2-way
L1 Data	32 KB 8-way	64 KB 2-way	64 KB 2-way
L2 Cache	2 MB 8-way shared	2 MB 16-way shared	1024 KB 16-way exclusive
L3 Cache	N/A	8 MB 32-way shared	N/A
Memory Controller	off-chip, 2 channel	off-chip, 2 channel	on-chip, 2 channel
Main Memory	3 GB DDR2-667	16 GB DDR3-1333	16 GB DDR2-667
Bus (MHz)	667	1333	1000
Max TDP (W)	31	95	95
Linux Kernel	2.6.23	2.6.29	2.6.31

Table 3.1: Machine Configuration Parameters

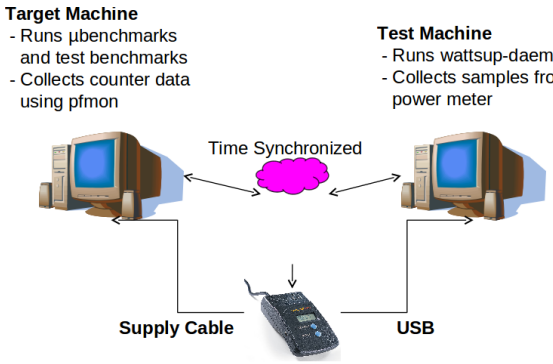


Figure 3.1: Experimental Setup

3.2.4 Test machines

For this master's thesis, we validated the power estimation model on three different platforms. The table 3.1 lists the platform used for this thesis work.

3.2.5 Hardware setup

As shown in the Figure 3.2.5, we use two machines for conducing the experiments. The target machine is one for which power estimation model is supposed to be formed. This target machine is connected to the supply line through the power meter. We sample the power values from the wattsup meter on the second (test) machine using a CPAN perl module *wattsup-daemon*. This is to make sure that this collection procedure doesn't pollute the power consumption values on the target machine. The two machines are time synchronized to make sure that correlation between the counter and power values is correct.

```
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 */
  .
  .
  .
}
}
```

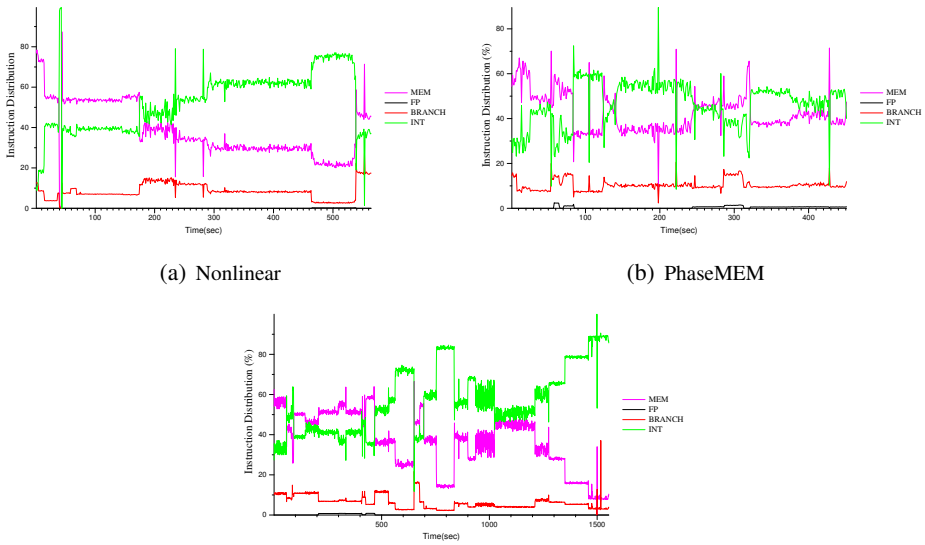
Figure 3.2: Microbenchmark Pseudo-Code

3.2.6 Microbenchmarks

We use the application independent microbenchmarks developed by Singh et al. [35] as training set for establishing the correlation between the normalized performance counter values and the measured power consumption value. These microbenchmarks exercise the performance counters from various categories, in isolation wherever possible, to establish the correlation between the counters and power consumption. The microbenchmarks consist of a mix of C and assembly code. The microbenchmarks run in different phases exercising counters from different categories in different phases. The Figure 3.2 shows the pseudo code for the microbenchmarks.

Apart from establishing the correlation with the power consumption, the microbenchmarks also establish the correlation between the counters themselves. This helps us in finding out the counters that are highly correlated with each other and hence redundant. The Figure 3.3 shows the instruction distribution for the three microbenchmarks used as training set in our work.

As shown in the instruction distribution, the microbenchmarks have large variations in instruction mix with different phases exercising different architectural components. Moreover, microbenchmarks change computational intensity in terms of total instructions retired per cycle over the period of run. Although, all the stated microbenchmarks exercise all the counter categories at various phases, they differ in terms of how they mix their phases. For example, nonlinear benchmark in-



(a) Nonlinear

(b) PhaseMEM

(c) PhaseNewMIX

Figure 3.3: Instruction Distribution During across Microbenchmarks

creases it's memory accesses gradually, phaseMEM accesses memory in separate but continuous phases while phaseNewMIX spreads out its memory accesses. These variations help in covering the corner cases for regression analysis.

We compile the microbenchmarks with no optimization to prevent redundant code removal, and run them simultaneously on each core. We assume that behavior of all programs falls within the space defined by our categories, making the approach applicable to both current and future applications.

Apart from the microbenchmarks, we also perform the training on the counter samples collected when the processor is idle.

3.3 Counter Selection

The selection of appropriate performance counters to be used in the power model is an extremely important step for the accuracy of the model. The methodology is aimed at choosing the counters that are most highly correlated with actual power consumption. At the same time, the chosen counters should preferably cover larger set of events to make sure that they are exercised by a general set of applications. If the chosen counters do not meet these criteria, the model will be error-prone.

The counter selection process involves collecting the activity factor of all the available counters on the processor using microbenchmarks, categorizing the counters and calculating the correlation of

each counter with actual power consumption, and then evaluating the top few counters to pick the ones best suited for the power estimation model.

3.3.1 Counter categories

To choose appropriate PMCs, divide the available counters into four categories and then choose one counter from each category. We do this to make sure that the chosen counters are comprehensive representations of entire microarchitecture and are not biased towards any particular section. We chose four categories because most of the processors have only two or four MSRs to which the performance counters can be mapped. As a result, we are limited in the number of counters that we can use in an online model without sacrificing the accuracy of sampled counter values. To chose the four categories, we take inspiration from the real estate usage of a typical modern processor. The caches and floating point execution units form large part of chip real estate and the performance counters which keep a tab on their activity factors would be very useful additions to the total power consumption information. There are various counters available in both these categories. For example, we can count the events that accessed all the different levels of cache hierarchy, and the total number of cache references as well as the number of cache misses. For the floating point operations, depending upon the processor model, one can count separately or in combination, the number of multiply, addition, or division operations. Apart from these two components, we expect out-of-order logic to consume a significant chunk of power consumption because of deep pipelining of modern processors. The stalls caused by the pipeline front end due to branch mispredictions or empty instruction decoder may reduce power. But the pipeline stalls caused because of full reservation stations and reorder buffers will be positively correlated with power. This is because full RS/ROB indicates that processor is working hard to extract instruction-level parallelism and keeping the execution units busy. Hence, pipeline stalls indicate not just the power usage of out-of-order logic but executions units as well. Therefore, we add the number of pipeline stalls as the third category for our counter selection. For the fourth category, we prefer choosing a generalized counter that can cover all the microarchitectural components that are not covered by the above three categories. This includes for example integer execution units, branch prediction unit, and SIMD units. Hence, we choose total instructions retired/executed as our fourth category to get an overall picture of CPU usage. Thus we categorized the counters as: *FP Units*, *Memory*, *Stalls*, and *Instructions Retired*.

3.3.2 Correlation

We collect the data for all the counters available for the particular platform by multiple runs of the benchmark. While running the microbenchmarks, we also collect the actual power consumption values from the power meter. Once we have collected all the data, we divide the counters in categories as described in 3.3.1. Then to select the counters for model formation, we use Spearman's rank correlation [36] to measure the relationship between each counter and power. We use the rank correlation method because later on, while forming the model, we intend to apply nonlinear transformations to the counter input values. Using the rank correlation, in comparison to correlation methods like Pearson's, makes sure that the non-linear relationship between the counter and

(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 3.2: Core i7 Counter Correlation

the power values does not affect the correlation coefficient.

Table 3.2 shows the top counters in each category as per the correlation coefficients obtained on Core i7 platform. As per table 3.1(a), only FP_COMP_OPS_EXE:X87 is a suitable candidate from the FP operations category. One thing to note here is that, ideally, to get total FP operations executed on the processor, we should sample both FP_COMP_OPS_EXE:X87 and FP_COMP_OPS_EXE:SSE_FP. For our experiments on Core i7, we use binaries compiled earlier on Intel Q6600 machine. The compiler used for compilation of these binaries didn't use SIMD floating point operations and hence we see high correlation for X87 counter but not for SSE counter. Compiling binaries with gcc 4.4 on Core i7, makes the microbenchmarks SSE intensive with minimal usage of X87. We stick with older binaries here because we were not able to compile all the test benchmarks on Core i7. Ideally, we will like chip manufacturers to provide a counter that counts the sum of X87 and SSE FP instructions so that we don't have to choose one of the two. As per table 3.1(b), the correlation values in total instructions category is too close to call and will need further analysis for selection. Same is the case for top three counters in stalls category as shown in 3.1(d), but we need to remember that we are looking for counters that provide us insight about out-of-order logic usage and hence, RESOURCE_STALLS:ANY counter is our best bet. As for counters for memory operations, choosing one of MEM_INST_RETIRED:LOADS or MEM_INST_RETIRED:STORES will bias the model towards load or store intensive applications. Similarly, choosing one of UOPS_EXECUTED:PORT1 or UOPS_EXECUTED:PORT0 counter from total instructions category will bias the counter towards add or multiply intensive applications. Hence we do not choose these counters for further analysis.

3.3.3 Inter-Counter correlation

As shown in the table 3.2, correlation analysis may throw up counters from the same category with very similar correlation numbers. If the correlation coefficients are too close to call, further analysis is required to choose an appropriate counter. Since, our aim is to make a comprehensive power model using only four counters, we need to make sure that the counters chosen are

(a) MEM vs 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 vs 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 vs 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 3.3: Counter-Counter Correlation

Category	Intel Core Duo	Intel Core i7	AMD Opteron 8212
Memory	L2_LINES_IN	LAST_LEVEL_CACHE_MISSES	DATA_CACHE_ACCESSES
Instructions Executed	BRANCH_INSTRUCTIONS_RETIRED	UOPS_ISSUED	RETIRED_INSTRUCTIONS
Floating Point	FP_COMP_OPS_EXE	FP_COMP_OPS_EXE:X87	DISPATCHED_FPU:OPS_MULTIPLY
Stalls	DBUS_BUSY:THIS_CORE	RESOURCE_STALLS:ANY	DECODER_EMPTY

Table 3.4: PMCs Selected for Each Platform

not providing redundant information. We do this by analyzing the correlation of counters from one category with the counters of other categories. To select one of the counters from memory operations category, we analyze the correlation of UOPS_EXECUTED:PORT234_CORE and LAST_LEVEL_CACHE_MISSES with the counters from total instructions category as shown in the table 3.2(a). From this table, it is evident that UOPS_EXECUTED:PORT234_CORE is highly correlated with instructions counter and hence providing redundant information. We thus choose LAST_LEVEL_CACHE_MISSES to represent memory operations. Now to choose one of the counters from total instructions category, we analyze the correlation of these counters with FP and Stalls counter as in 3.2(b) and 3.2(c) respectively. The results that we get are too close to select one of the four options. In such a case, we can either choose one counter at random or choose a counter intuitively. UOPS_EXECUTED:PORT015 is not so preferable since it does not cover the memory operations that were executed but were satisfied by L1/L2 cache instead of going to memory. The UOPS_RETIRED:ANY and INSTRUCTIONS_RETIRED cover only those instructions that were retired but do not cover the instructions that were executed but not retired due to branch misprediction (wasted work). A UOPS_EXECUTED:ANY counter would have been good but since we don't have it, the next best thing we have is UOPS_ISSUED:ANY. This counter covers all the instructions issued, so it also cover the instructions that were issued but not executed (and not retired obviously), because of flushing of RS/ROB after branch misprediction. To validate this theory, we have tested our model with both UOPS_ISSUED:ANY and UOPS_RETIRED:ANY, and the first one gives better results, albeit only marginally.

Table 3.4 shows the counters selected for each of the test machines that were used during this thesis.

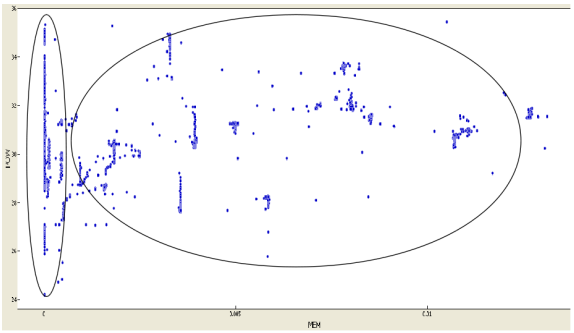


Figure 3.4: Piece-wise Relationship of Power and Memory

3.4 Model Formation

We use multiple regression analysis to form a linear regression model for predicting the power consumption using sampled counter values and temperature readings. Once the desired counters have been identified, we run the microbenchmarks and sample the chosen counters, temperature sensor readings and the empirical power consumption. The sampled PMC values e_i are normalized to the elapsed cycle count to generate an event rate r_i, so that it can be used in an equation involving rise in core temperature T and the rise in power consumption P_{core}, which are instantaneous values. This makes sure that changing the sampling period of the readings doesn't affect the weight of the respective predictors.

3.4.1 Piecewise modeling

We get better fit, when we break the collected samples into two parts based upon the value of either memory counter or FP counter. Breaking the data using memory counter value helps us in separating the memory bound phases from the CPU bound phases. Using FP counter instead of memory to break the data helps in separating the floating point intensive phases from non-floating point intensive phases. The selection of candidate for breaking the counter is machine specific and depends on what gives better fit. For Intel Core Duo we break the data using FP counter while on AMD 8212 and Intel Core i7 we use memory counter. The value of the breakpoint is decided by eyeballing the counter vs power data. We use the eureqa tool [28] to analyze this data and decide the breakpoint value. The relation between memory and empirical power for Core i7 is shown in the Figure 3.4.1. As shown, the power doesn't show any correlation with memory for very low values ($< 1e-6$).

3.4.2 Nonlinear transformations

The relationship between the counters and the power consumption is not always linear. The empirical observation suggests that many a times non-linear correlation exists between the two. To

accommodate such a behavior in the model, we can explore nonlinear regression analysis techniques. But a simpler approach is to apply non-linear transformations on the suspected counters in a linear regression model. This approach ensures that we get a better fit in the face of existing non-linear relationships while at the same time retain the simplicity of linear regression techniques. In most of the cases, we use logarithmic transformation to get a better fit.

3.4.3 Regression model

We formulate a piece-wise linear regression model using STAT Package's *regress* tool based on Kerlinger and Pedhazur's algorithm [25], as depicted in Equation 3.1. We apply transformations g_i to the event rates as shown in Equation 3.2 to accommodate the non-linearity in the relationship between r_i and power, wherever applicable.

$$\hat{P}_{core} = \begin{cases} F_1(g_1(r_1), \cdots, g_n(r_n), T), & if \text{ condition} \\ F_2(g_1(r_1), \cdots, g_n(r_n), T), & else \end{cases} \tag{3.1}$$

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

$$F_n = p_0 + p_1 * g_1(r_1) + ... + p_n * g_n(r_n) + p_{n+1} * T \tag{3.2}$$

The sample linear regression model for Core i7 is shown in Equation 3.3. For the first part of the piece-wise model, the coefficient for memory counter is zero. This is expected as per the data observed in Figure 3.4.1. Another thing to note is that some of the coefficients are negative even though all the counters are positively correlated with power. This shows that some amount of colinearity exists in our model. Although not desirable, this is acceptable as long as our model exhibits low error rates.

$$\hat{P}_{core} = \begin{cases} 10.9246 + 0 * r_{MEM} + 5.8097 * r_{INSTR} + 0.0529 * r_{FP} + \\ \qquad 6.6041 * r_{STALL} + 0.1580 * T, & if\ r_{MEM} < 1e-6 \\ 19.9097 + 556.6985 * r_{MEM} + 1.5040 * r_{INSTR} + 0.1089 * r_{FP} + \\ \qquad -2.9897 * r_{STALL} + 0.2802 * T, & if\ r_{MEM} \geq 1e-6 \end{cases} \tag{3.3}$$

3.4.4 Significance testing

We need to check the results of the significance testing performed by *regress* tool during model formation to verify the significance of chosen counters as predictors of power consumption. The results of the significance test for the model formation on Core i7 are shown in the Figure 3.5. The R-squared value in the *regress* output shows the overall significance of the regression. It is a measure of the fraction of response variable (core power) that is predicted by the predictor variables. As per the results obtained, we can see that our regression model shows very high overall significance.

The significance test of the individual predictors tell us the importance of each counter chosen in the model formation. The p values in the last column of significance test shows the probability of null hypothesis for the chosen counter. In other words, it signifies the affect the chosen counter

```
Significance test for prediction of POW
    Mult-R  R-Squared      SEest   F(5,245)   prob (F)
    0.9994     0.9988     0.3416 41496.6729     0.0000

Significance test(s) for predictor(s) of POW
Predictor        beta          b        Rsq         se     t(245)          p
X1            -0.0219     0.0000     0.2279     0.0000    99.9950     0.0000
X2             0.5482     5.8097     0.9791     0.1610    36.0908     0.0000
X3             0.0147     0.0529     0.0630     0.0082     6.4814     0.0000
X4             0.1747     6.6042     0.8971     0.2586    25.5400     0.0000
X5             0.3277     0.1580     0.9875     0.0095    16.6771     0.0000
```

(a) *regress* output piece 1

```
Significance test for prediction of POW
    Mult-R  R-Squared      SEest  F(5,2465)   prob (F)
    0.9653     0.9318     0.9637  6733.9879     0.0000

Significance test(s) for predictor(s) of POW
Predictor        beta          b        Rsq         se    t(2465)          p
Y1             0.5285   556.6986     0.7668    11.4759    48.5104     0.0000
Y2             0.4510     1.5040     0.9112     0.0589    25.5480     0.0000
Y3             0.1215     0.1089     0.0303     0.0048    22.7455     0.0000
Y4            -0.1767    -2.9897     0.6157     0.1436    20.8172     0.0000
Y5             0.5557     0.2802     0.8973     0.0083    33.8460     0.0000
```

(b) *regress* output piece 2

Figure 3.5: Significance Test Results for Core i7

is having on the prediction of the power and lower is better. As can be seen in the results, all our chosen counters show zero probability of being insignificant.

We mentioned in 3.2.6 that we use idle processor samples for training apart from microbenchmarks. We can see the change in significance tests when we remove these idle samples from our training set in Figure 3.6. As seen, there is sharp decrease in values of R-squared when we remove idle set.

3.5 Temperature Effects

The power consumption of processor is composed of dynamic and static power consumption elements. Among these, the static power consumption is dependent on temperature of the core. As shown in Equation 3.5, the static power consumption of processor is a function of leakage current and supply voltage. The processor leakage current, is in turn, affected by process technology, supply voltage and temperature. With the increase in the power consumption, the processor temperature increases. This increase in temperature increases the leakage current which in turn

```
Regression Equation for POW:
POW  =  0 X1  +  3.923 X2  +  0.0306 X3  +  3.359 X4  +  0.1881 X5  +  14.0756

Significance test for prediction of POW
    Mult-R  R-Squared      SEest   F(5,150)   prob (F)
    0.9497     0.9020     0.4321   275.9820     0.0000

Significance test(s) for predictor(s) of POW
Predictor        beta          b        Rsq         se     t(150)          p
X1             0.0525     0.0000     0.0907     0.0000    99.9950     0.0000
X2             1.0066     3.9228     0.8669     0.2731    14.3649     0.0000
X3             0.0781     0.0306     0.0761     0.0104     2.9375     0.0038
X4             0.4711     3.3595     0.8503     0.4712     7.1299     0.0000
X5             0.4918     0.1881     0.2799     0.0115    16.3237     0.0000
```

(a) *regress* output piece 1

```
Significance test for prediction of POW
    Mult-R  R-Squared      SEest  F(5,2393)   prob (F)
    0.8693     0.7558     0.8783  1480.9011     0.0000

Significance test(s) for predictor(s) of POW
Predictor        beta          b        Rsq         se    t(2393)          p
Y1             1.3698   692.9299     0.8200    12.0456    57.5256     0.0000
Y2             1.0108     2.7653     0.8933     0.0846    32.6866     0.0000
Y3             0.2846     0.1214     0.0409     0.0044    27.5892     0.0000
Y4            -0.2035    -1.8659     0.7921     0.2031     9.1851     0.0000
Y5             0.2929     0.1993     0.5357     0.0101    19.7582     0.0000
```

(b) *regress* output piece 2

Figure 3.6: Significance Test Results(without idle training set)

increases the processor static power consumption. This effect is shown in the Figure 3.7(a). To study the affect of temperature on power consumption, we run a multi-threaded program that executes *MOV* operations in an infinite while loop. Hence, the behavior of the program over its entire run is almost constant. This should entail that the dynamic power consumption of the processor doesn't change over the run of the stated program as per Equation 3.6. Thus, the gradual increase in power consumption during the run of the program can be attributed to the gradual increase in temperature as shown in the graph. As we can see, the total power consumption increases by almost 10% due to change in static power consumption. To account for this increase in static power, it is necessary to include temperature in the power model.

As per Equation 3.5, the static power consumption increases exponentially with temperature. We can confirm this empirically by plotting the increase in power consumption once the program starts execution with the increase in temperature as shown in the Figure 3.7(b). The exponential regression gives us the curve as shown in the figure, which closely follows the empirical data points with determination coefficient $R^2 = 0.995$. Plotting this estimate of increment in static power consumption as in 3.7(c) explains the gradual rise in total power consumption when the dynamic behavior of program is constant. But this equation and curve fitting is valid only for $V_{core} = 1.09V$. For estimating static power consumption accurately for benchmarks across different DVFS points, we will need an equation that takes the core voltage into account. We will explore that option as part of future work. We use the given equation for calculating static power consumption in an alternate model mentioned in Appendix A.

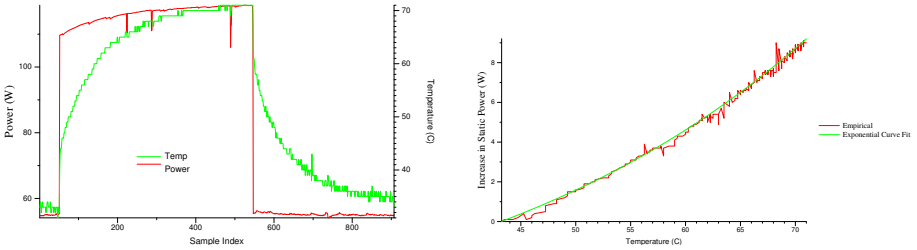
$$P_{staticInc} = 1.4356 \times 1.034^T, \quad \text{when } V_{core} = 1.09V \tag{3.4}$$

3.6 Validation

We evaluate the accuracy of our model by estimating per-core power for the SPEC 2006 [38], SPEC-OMP [2, 37], and NAS [4] benchmark suites. We sample the empirical core power consumption as described in 3.2.1 and performance counters and core temperature during the run of benchmark applications. We then compare the power value estimated by the regression model against the empirical power values to calculate estimation error. We use gcc 4.2 to compile our benchmarks for 64-bit architecture (Intel Core i7 and AMD 8212) and 32-bit architecture (Intel Core Duo) with optimization flags enabled, and run all benchmarks to completion on top of Linux kernel version 2.6.29 (Intel Core i7), 2.6.31 (AMD 8212) and 2.6.23 (Intel Core Duo).

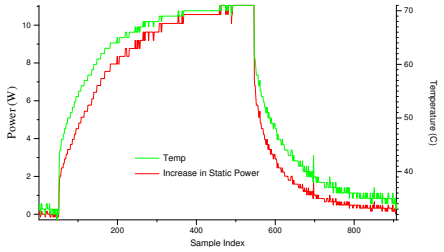
Idle processor temperature is 33°C for both the Intel Core i7 and AMD 8212 platforms. Idle system power is 54.0W for the Intel Core i7, and 302.1W for the AMD 8212. We subtract 44.0W idle processor power to obtain an uncore power of 10W for the Core i7, and subtract 212.1W idle processor power to obtain an uncore power of 90.0W for the 8212. Idle processor temperature for the Intel Core Duo system is 18°C, and uncore power is 13.2W.

We test our models for both multi-threaded and single-threaded applications on our three platforms. We assess model error at the granularity of one second (limitation of power meter).



(a) Temperature vs Power on Core i7

(b) Static Power Curve



(c) Temperature vs Static Power on Core i7

Figure 3.7: Temperature Effects on Power Consumption

3.6.1 Estimation error

In our experiments, we run multi-threaded benchmarks with one thread per core, and single-threaded benchmarks with an instance on each core. Data are calculated per core, and error is reported across all cores. Figure 3.8, Figure 3.9, and Figure 3.10 show percentage median error for the NAS, SPEC-OMP, and SPEC 2006 applications on all systems. Figure 3.11, Figure 3.12, and Figure 3.13 show standard deviation of error for each benchmark suite on the Intel Core Duo, Intel Core i7, and the AMD 8212 platforms. The occasional high standard deviations illustrate one problem with our current infrastructure: instantaneous power measurements once per second do not tell us what has been happening to the performance counters since the last meter reading.

Estimation error ranges from 0.3% (*leslie3d*) to 7.1% (*bzip2*) for the Intel Core Duo system, from 0.3% (*ua*) to 7.0% (*hmmer*) for the Intel Core i7 system, and from 1.0% (*bt.B*) to 10.7% (*soplex*) for the AMD Opteron 8212 system. Our model exhibits median error of 4.36%, 4.01%, and 3.73% for SPEC-OMP, SPEC 2006, and NAS, respectively, on the Core Duo. Median errors per suite are 4.14%, 1.61%, and 3.11% on the Core i7, and 3.4%, 4.8%, and 2.5% on the 8212. On the Intel Core Duo, 17 (out of 45) applications exhibit median error exceeding 5%; and on the Intel Core i7, only five exhibit error exceeding 5%. On the AMD Opteron 8212, thirteen exhibit error over 5%.

The table 3.5 summarizes the median estimation error observed on all the test machines.

The model estimation error can be attributed to following sources:

1. Even though the microbenchmarks try to cover all scenarios of power consumption, the

Benchmark	AMD Opteron 8212	Intel Core i7	Intel Core Duo
SPEC 2006	4.80	2.22	4.01
NAS	2.55	3.11	3.45
SPEC OMP	3.35	2.02	4.36
Total	4.38	2.07	4.18

Table 3.5: Estimation Error Summary

resulting regression model will represent a generalized case. This is especially true for a model like ours which tries to estimate power for a complex microarchitecture using only four counters. For example, the floating point operations can consist of add, multiply or divide operations which use different execution units and hence consume a different amount of power. If the test benchmark is close the instruction mix used for microbenchmark, estimation error will be low, and vice-versa.

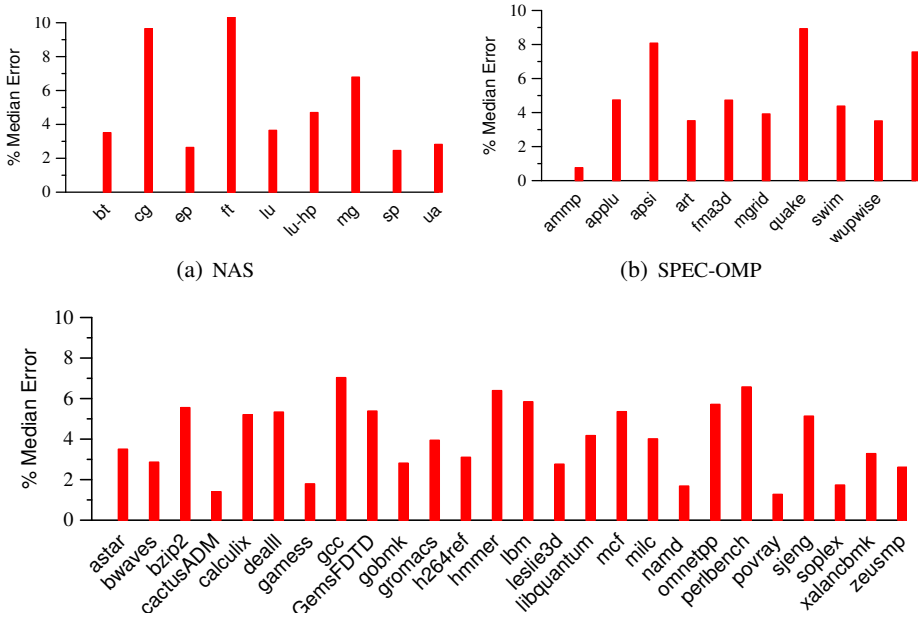
2. The temperature components plays a large part in model formation. The *lm-sensors* driver gets the temperature readings from the on-die thermal diodes. But these thermal diodes are not so accurate for some processor models, like Opteron 8212 [1]. This inaccuracy of thermal sensors adversely affects the model estimations.

3. Model accuracy also depends on the PMCs available on a given platform. If available PMCs do not sufficiently represent the microarchitecture, model accuracy will suffer. For example, the AMD Opteron 8212 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. Same is true for stalls counters available in Intel Core Duo.

4. Our model suffers with high error peaks because of limitation of the sampling rate of power meter. A maximum sampling rate support of one sample per second entails that we need to accumulate performance counter values for one second and normalize them using the cycles elapsed during one second. This in effect averages the counter activity during the one second accumulation period. As a result, whenever there is a rapid change in counter activity, the power estimated for that sample is significantly lower (for a positive surge) or higher (for a negative surge) compared to the actual value. This effect is more easily understood by looking at the figure showing error peaks in Appendix C.

5. As stated earlier, some processor models support only two MSRs for mapping the performance counters. For these processors, to capture the activity of four counters required for model formation, the counting of performance counter related events needs to be multiplexed. In effect, the events are counted for only half the second (or half the total sampling period), and are multiplied by two to estimate the value over the entire period. This approximation can introduce inaccuracies when the program behavior is changing rapidly.

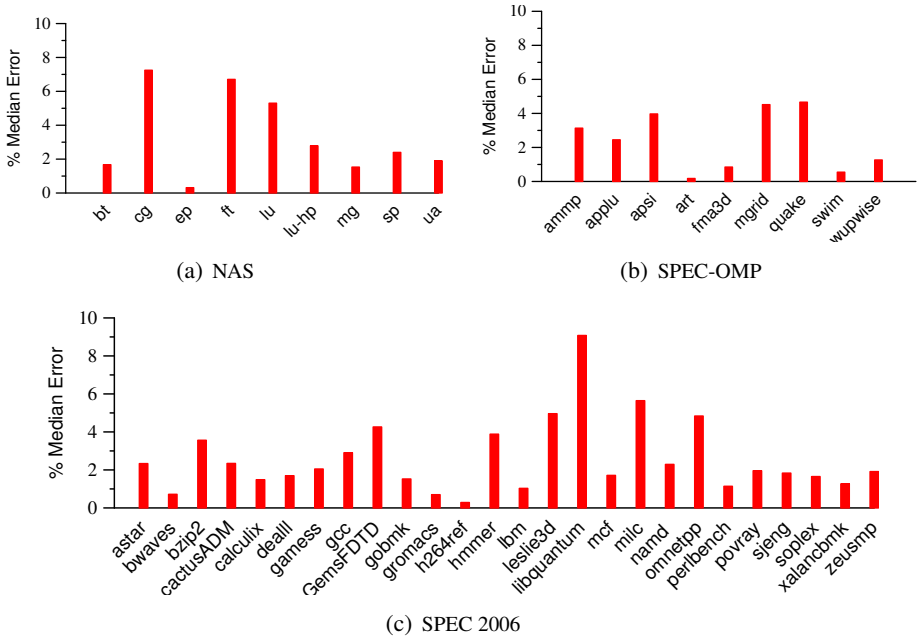
Figure 3.14 shows model coverage via CDF plots for the suites. On the Core Duo, 62% of estimates have less than 5% error, and 96% have less than 10%. On the Core i7, 82% have less than



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

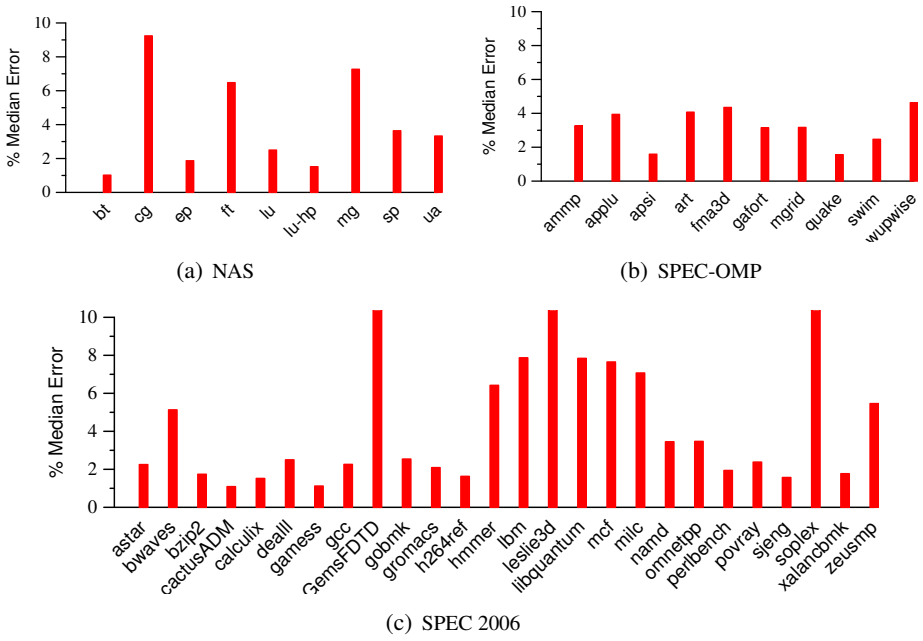
Figure 3.8: Median Estimation Error for Intel Core Duo



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

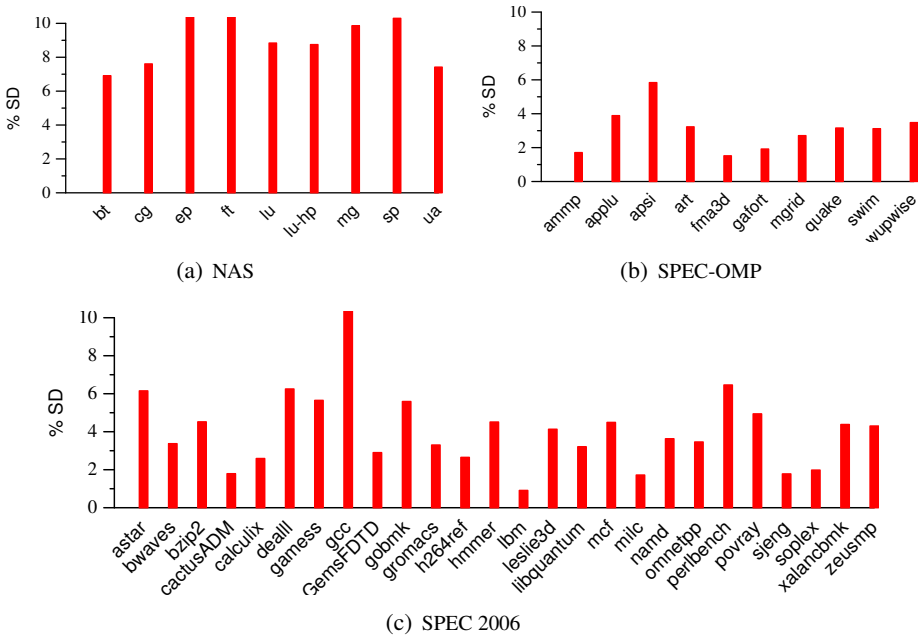
Figure 3.9: Median Estimation Error for Intel Core i7



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

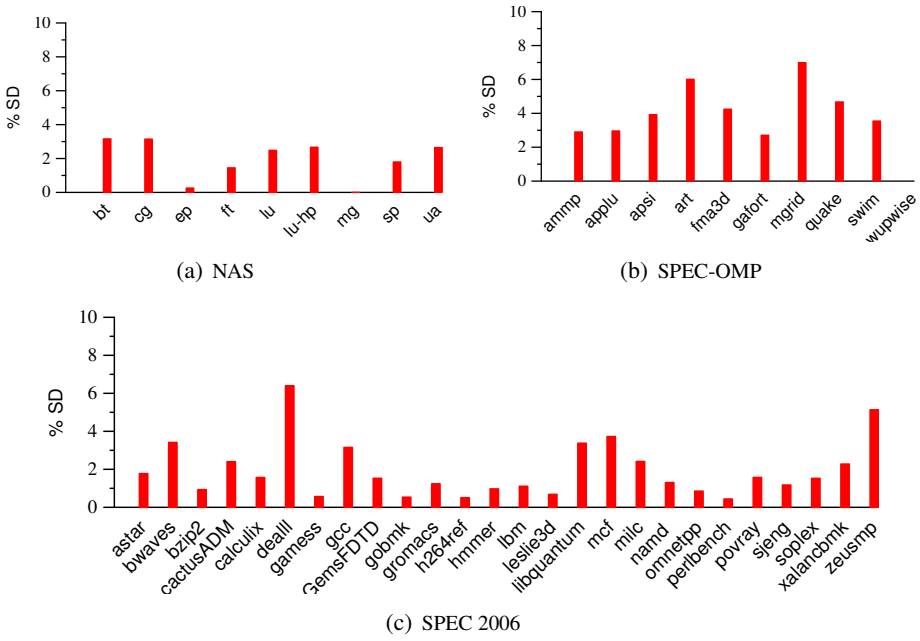
Figure 3.10: Median Estimation Error for the AMD Opteron 8212



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

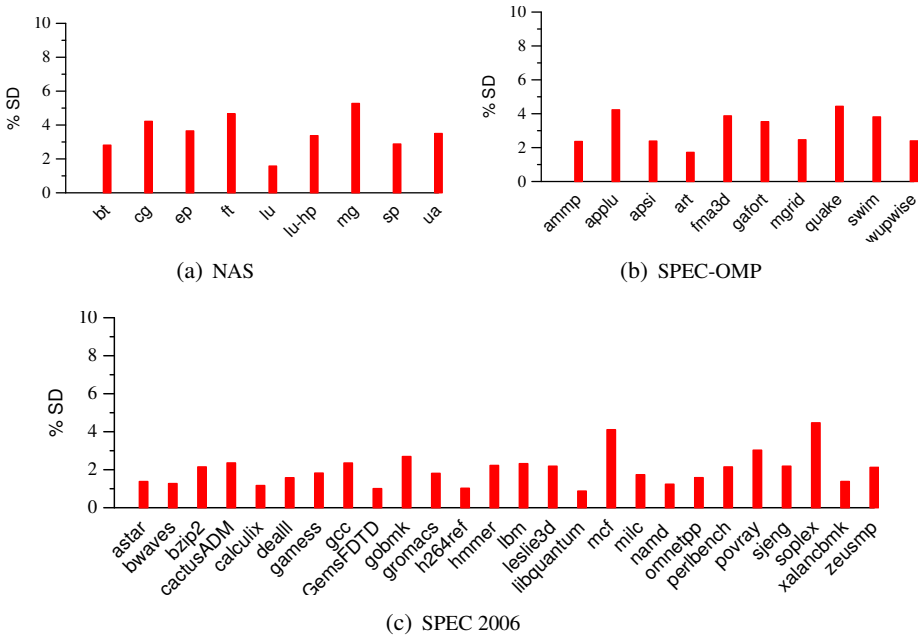
Figure 3.11: Standard Deviation of Error for Intel Core Duo



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

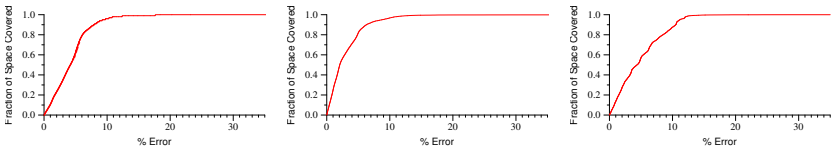
Figure 3.12: Standard Deviation of Error for the Intel Core i7



(a) NAS (b) SPEC-OMP

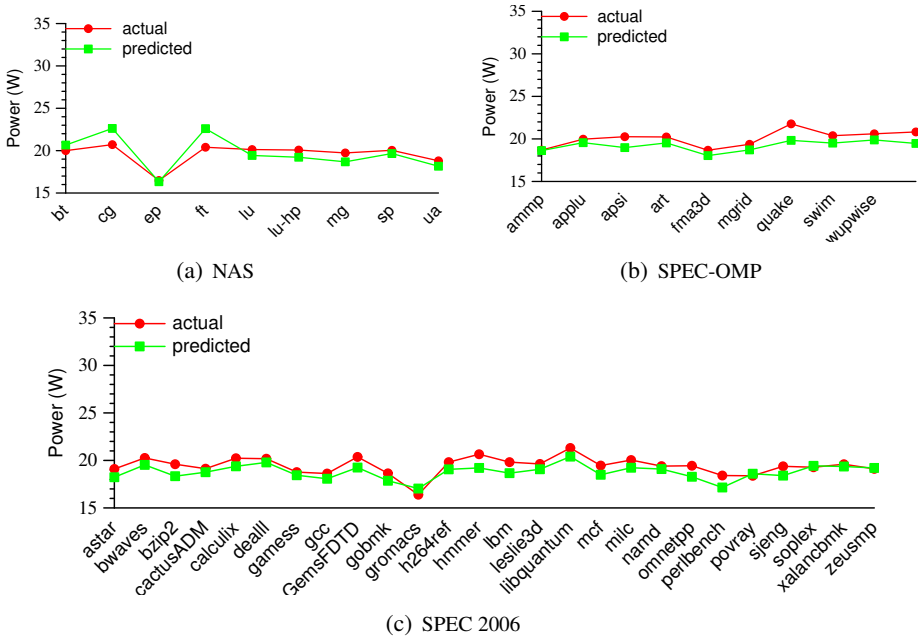
(c) SPEC 2006

Figure 3.13: Standard Deviation of Error for the AMD Opteron 8212



(a) Intel Core Duo (b) Intel Core i7 (c) AMD 8212

Figure 3.14: 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

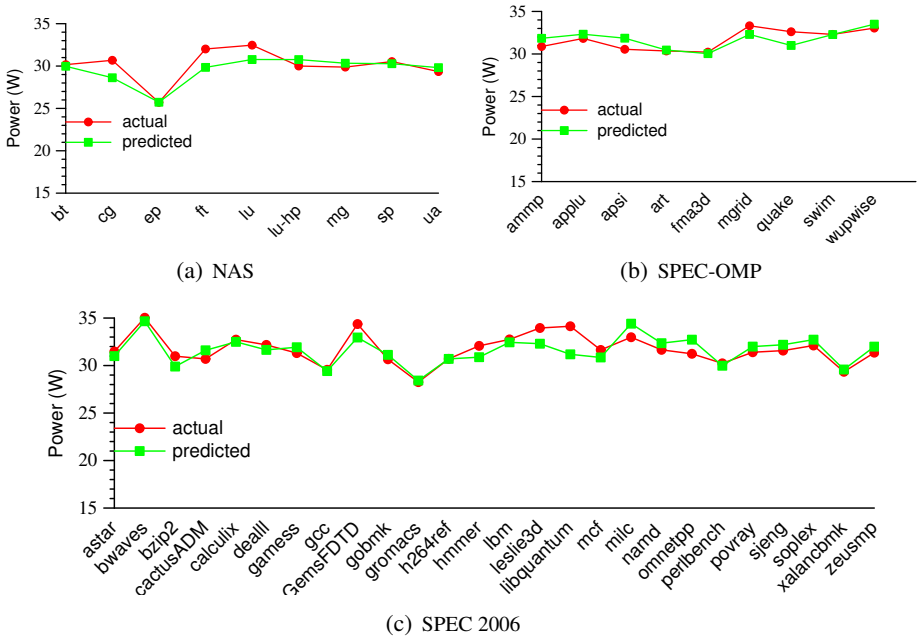
Figure 3.15: Estimated vs. Measured Error for Intel Core Duo

5% error, and 97% less than 10%. On the 8212, 37% have less than 5% error, and 76% have less than 10%. The vast majority of estimates exhibit very small error.

Figure 3.15, Figure 3.16, and Figure 3.17 compare measured to estimated power for our experimental platforms. Values are calculated as the mean of power over an application's execution. These data show that our models do not consistently under- or over-estimate power, tracking it well for each suite.

3.6.2 Effects of Turbo Boost

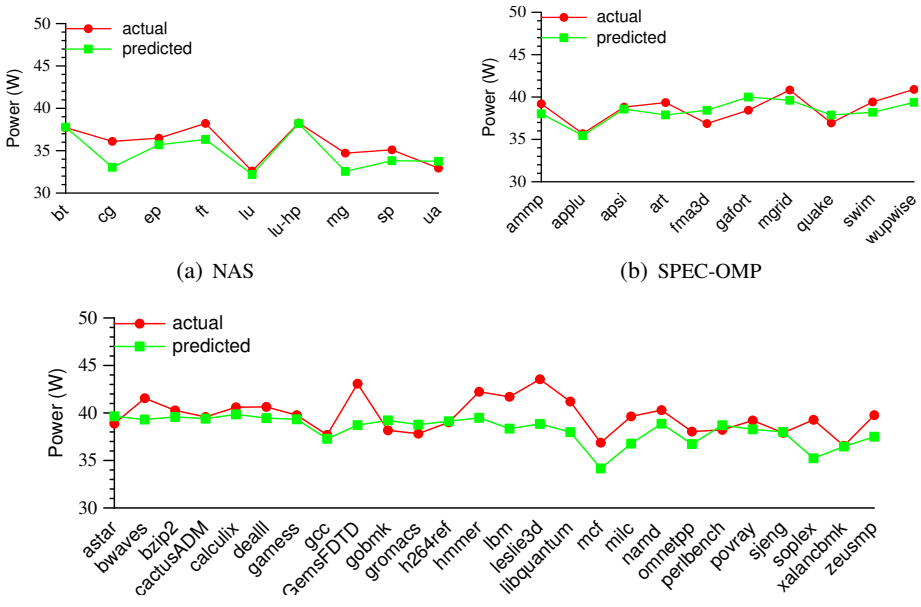
Intel Core i7 platform employs a performance boosting technique called Turbo Boost [11]. This technique allows the active processor cores to run at a higher frequency than the base operating frequency (2.93 GHz in our case) if there is headroom in the temperature, power and current



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

Figure 3.16: Estimated vs. Measured Error for Intel Core i7



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

Figure 3.17: Estimated vs. Measured Error for the AMD Opteron 8212

Policy	Description	Affected Microarchitecture
Replicated	Duplicate logic per thread	Register State Renamed RSB Large Page ITLB
Partitioned	Statically allocated to threads	Load Buffer Store Buffer Reorder Buffer Small Page ITLB
Competitively shared	Dynamically allocated to threads	Reservation Station Caches Data TLB 2^{nd} level TLB
Unaware	No impact	Execution Units

Table 3.6: Hyper-Threading Partitioning on Core i7

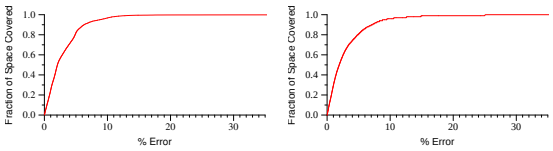
specification limits. The Turbo Boost upper limit is defined by the number of active cores. When all the four cores are active and operating system demands the higher performance state (P0), the core frequency can go up to 3.2 GHz. This change in frequency can adversely affect the accuracy of our model since with change in frequency, the core voltage changes and hence the static and dynamic power consumption of the core changes. This will render the model formed at 2.93 GHz frequency error prone. But during our experiments, it is observed that since while running the test suites, all four CPU cores are already stretched to limit and since all the four cores are running identical loads, the Turbo Boost is never triggered. Nevertheless, we run our experiments with Turbo Boost disabled to be sure.

3.6.3 Effects of Hyper-Threading

Intel Core i7 platform supports Hyper-Threading [11]. This technology allows the processor to divide the physical core into two logical cores, effectively providing eight logical cores on Core i7. The partitioning example of the physical core resources among two logical cores is shown in the Figure 3.6. The results presented above are for case when Hyper-Threading is disabled in the system BIOS. But our methodology can work in the case when Hyper-Threading is enabled. The reason being that all performance counters, barring UOPS_EXECUTED:PORT234_CORE, are available separately for each core. We tried evaluating our methodology for the case when Hyper-Threading is enabled with median errors of 3.48% for SPEC-OMP, 2.15% for NAS, 2.46% for SPEC 2006 and 2.56% overall. The detailed results are presented in Appendix D.

3.6.4 Model formation using eight counters

During our tests, we have validated that for most of the processors, four performance counters bring the predicted value within 5% of the actual power consumption. More performance counters would definitely be beneficial to improve the accuracy, but only if they can be read accurately. Unfortunately, most of the platforms allow simultaneous reading of only two or four counters. Hence, to use more counters, the software would need to multiplex the counter reading and scale up the sampled values to estimate the total count. This process would introduce errors to PMC values. Moreover, as per collected data, the model accuracy does not improve significantly when



(a) Intel Core i7 - 4 PMCs (b) Intel Core i7 - 8 PMCs

Figure 3.18: CDF Plots Comparison of 4- and 8-counter Model

more than four counters are used.

For example, on Core i7, using eight counters gives us a median error for all benchmarks of 1.92% while using four counters gives a median error of 2.06%. The Figure 3.18 compares the CDF plots of the errors encountered during the entire run of test benchmark suites on Core i7 processor for two models, one using four counters as before and the other using eight counters. As we can see, the CDF plots are almost similar. This proves, that for most of the cases, four counters are enough to bring the model accuracy within acceptable range.

3.6.5 Effects of SIMD operations

To capture all the floating point operations being executed on Core i7, ideally we would need to sum up two performance counters: FP_COMP_OPS_EXE:X87 and FP_COMP_OPS_EXE:SSE_FP. But since we don't have the privilege of using more than one counter for floating point operations, we can use only one of the two. If the benchmark application has been compiled on an older x86-64 architecture, chances are that it would be using the x87 coprocessor more heavily. But if the application has been compiled locally on the Core i7 machine, it would be more SSE intensive. Since we were not able to compile all the benchmark applications locally, we have used the benchmark applications compiled on AMD phenom and used FP_COMP_OPS_EXE:X87 in our model.

3.6.6 Effects of DVFS

The Core i7-870 processor can operate at 14 different P(performance) states. The range of frequency across the P states range from maximum of 2.93 GHz (without Turbo Boost enabled) to minimum of 1.197 GHz. Although this wide range and fine control of the scaling frequency proves to be an excellent knob for power consumption control, the enabling of DVFS increases the complexity of experiments tremendously. This is because when the voltage-frequency operational point is shifted, apart from the change in dynamic power consumption, there is a significant change in the static power consumption. The equation for the static and dynamic power consumption is as per equations 3.5 and 3.6 [39]. To create a single power model that can correctly estimate power consumption across all the P-states, we need to separate static power consumption estimation from dynamic power consumption estimation and then scale the two power components separately for different frequency and core supply voltage points. Hence, to estimate power

consumption across different frequencies, we have to generate individual power models for each P-state. A generalized single power model that can calculate the power consumption values across different frequency-voltage points is part of future work.

$$\begin{aligned} P_{static} &= \sum I_{leakage} * V_{core} \\ &= \sum I_s(e^{qV_d/kT} - 1) * V_{core} \end{aligned} \tag{3.5}$$

where $I_s = reverse\ saturation\ current$
$V_d = diode\ voltage$
$k = Boltzmann's\ constant$
$q = electronic\ charge$
$T = Temperature$
$V_{core} = Core\ supply\ voltage$

$$P_{dynamic} = N_{SW} * C_{pd} * V_{CC}^2 * f_I \tag{3.6}$$

where $N_{SW} = Number\ of\ bits\ switching$
$C_{pd} = dynamic\ power\ dissipation\ capacitance$
$V_{CC} = supply\ voltage$
$f_I = CPU\ frequency$

Chapter 4

Power-Aware Scheduling

In this chapter, we present a live power management application as a proof-of-concept for the power estimation model presented in chapter 3.

4.1 Introduction

The live power management application schedules the given tasks under the constraint of maintaining a user-defined system power envelope. The application uses the power model to compute the core power consumption in real time. The application works like a user-level meta scheduler that spawns one process on each core. The scheduler uses *pfmon* to sample performance counter values and feeds the sampled PMC values to the power model for estimating core power consumption.

The meta-scheduler binds the affinity of the processes to a particular core to make the task management and power estimation process simpler. The scheduler dynamically calculates the core power consumption values at a set interval (one second in our case), and compares the sum of all cores and the uncore power consumption for comparison with system power envelope. When the breach in the envelope is detected by the scheduler, the scheduler takes steps to force the power consumption down. The scheduler employs two knobs to control the 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 voltage-frequency. If the voltage-frequency has been scaled down to the maximum extent and the target power consumption is still below the estimated value, the scheduler starts suspending processes to meet the envelope demand. When the current power consumption is less than the target power envelope, the scheduler first checks if any suspended process can be resumed. If the gap between the current and 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 frequency. The Figure 4.1 shows the flow diagram of the meta-scheduler.

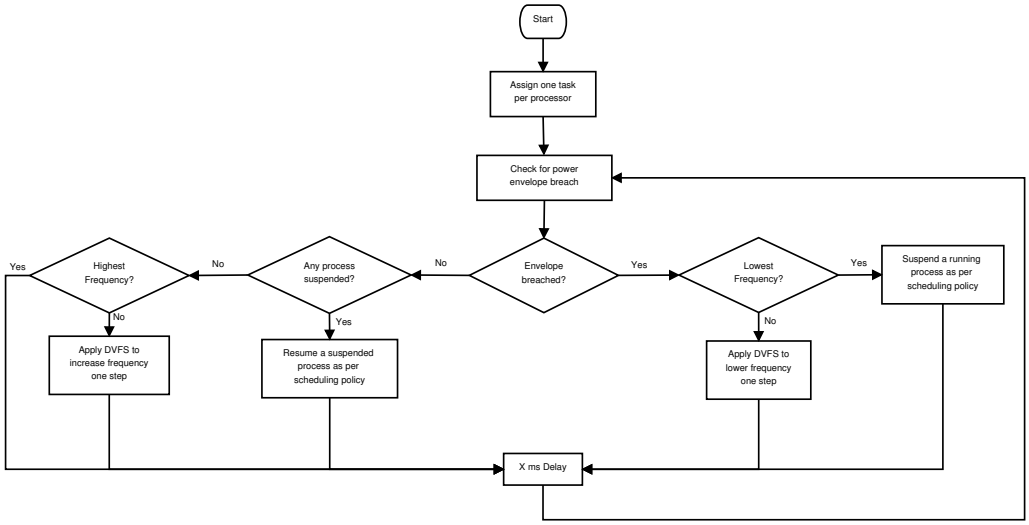


Figure 4.1: Flow diagram for Meta-scheduler

4.2 Sample Policies

When the scheduler suspends a process, it needs to choose the process for suspension that will have the least impact on overall completion time of all the processes. We have explored four sample policies for process suspension as described below.

4.2.1 Max instruction/watt policy

This policy targets maximum power efficiency under given power envelope. When the envelope is breached, the scheduler calculates the ratio of instructions/UOPS retired to the power consumed for each core and suspends the process which has committed least instructions per watt consumed. When resuming a process, it selects the process (if there are more than one suspended processes) which had committed maximum instructions/watt at the time of suspension. This policy favors the processes which are stalled less often waiting for load operations to complete. Hence, this policy favors the CPU bound applications

4.2.2 Per-core fair policy

This policy is aimed at dividing the available power budget fairly among all the processes. The scheduler when applying this policy maintains a running average of the power consumed by each core. At the time of process selection for suspension, the scheduler suspends the process which has consumed maximum average power. For resumption, the scheduler resumes the process which had consumed least average power at the time of suspension. This policy can be used to regulate the core temperature since it makes sure that cores that are consuming maximum average power are throttled. Since there is high correlation between the core power consumption and core tem-

perature, this makes sure that the core with highest temperature gets time to cool down while the cores with lower temperature continue working. Since the memory bound applications are stalled more often and hence consume less average power, this policy favors such applications.

4.2.3 Critical process running policy

This policy is useful when the scheduler has prior knowledge of the relative execution times of the tasks. This policy attempts to achieve the best possible total execution time with given power envelope. The scheduler selects the process with lowest execution time for suspension. This makes sure that the process with longest execution time which is critical to the overall execution time of the group of tasks is always running. This policy is preferable when there is enough difference between the execution times of the given tasks. This policy is essentially a modification of the user-based priority policy proposed by Singh et al. [34].

4.2.4 Round-robin policy

As the name suggests, this policy suspends the processes in round-robin fashion to be fair to all the processes regardless of their computational intensity. The scheduler in this policy maintains two lists: a list of running processes and a list of suspended processes. When the power envelope is breached, the next process to be suspended is always taken the head of the running processes list and added to the end of suspended processes list. When the system power is below the envelope, the scheduler searches through the list of suspended processes, starting from the head of the list, to find a process whose resumption will not breach the envelope. If none of the suspended processes can be resumed without breaching the envelope, the scheduler suspends the process at the head of list of running processes and resumes the process at the head of suspended processes list.

4.3 Experimental Setup

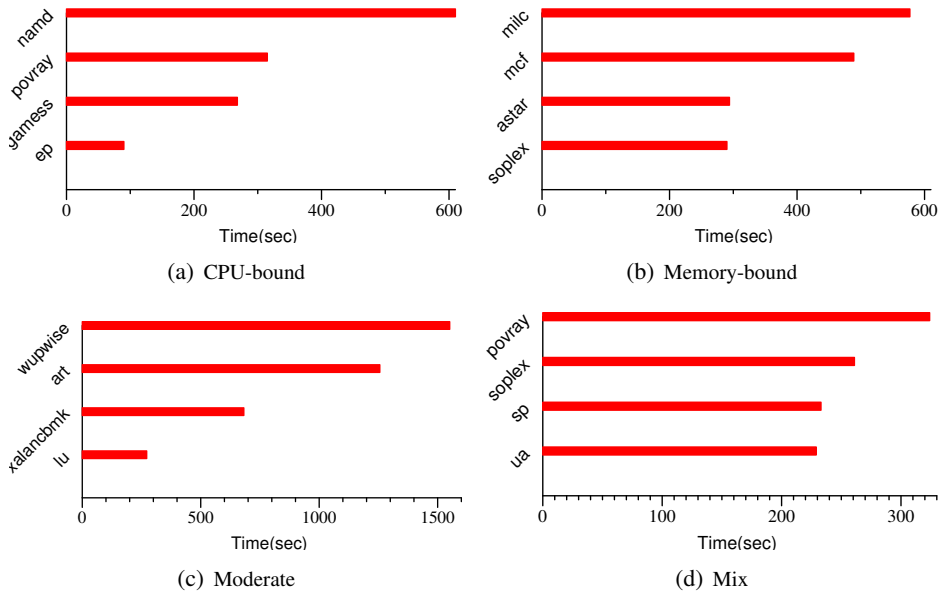
We have conducted all our scheduler experiments on Core i7 processor. We conduct the meta-scheduler experiments by dividing the workloads into three sets based upon the *CPU intensity*. We define CPU intensity as ratio of instructions retired to last level cache misses. The three sets are called CPU-bound, Moderate and Memory-bound workloads in the decreasing order of CPU intensity. Apart from these three sets, we also experiment with a mix set of workloads with focus on similar execution times. The workloads categorized in these sets is listed in the table 4.1.

The benchmark applications used in the workload sets have very different execution times. The execution times of different applications is shown in Figure 4.2. As we will see in the results section 4.4, this affects the total execution times of different scheduling policies significantly. This is the reason we included a fourth workload set 'Mix' with emphasis on similar execution times.

We conduct the experiments by setting the power envelope to 90%, 80% and 70% of the peak power usage and calculating the total execution time to run all the applications in the workload

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
Mix	ua, sp, soplex, povray	145

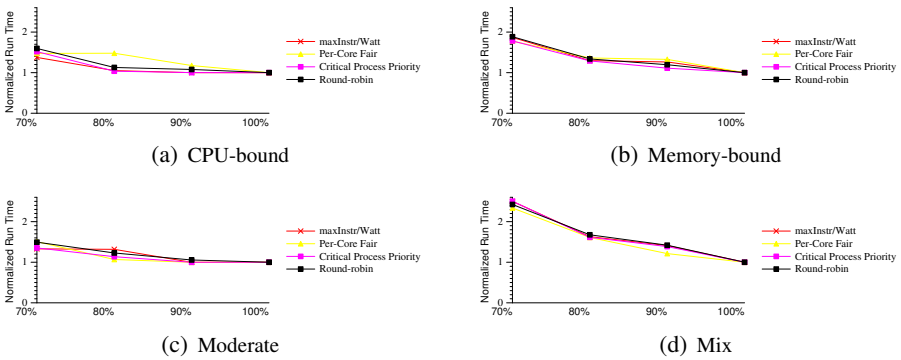
Table 4.1: Workloads for Scheduler Evaluation



(a) CPU-bound (b) Memory-bound

(c) Moderate (d) Mix

Figure 4.2: Absolute Runtimes for Unconstrained Workloads on the Core i7



(a) CPU-bound (b) Memory-bound

(c) Moderate (d) Mix

Figure 4.3: Runtimes for Workloads on the Core i7 (without DVFS)

under a given scheduler policy. We chose the stated power envelope settings, because the difference in the performance was not significant for the power envelope settings at the difference of 5% (95%, 90% and 85%) and running the experiments was not feasible for the envelope settings at the difference of 20% (80%, 60% and 40%).

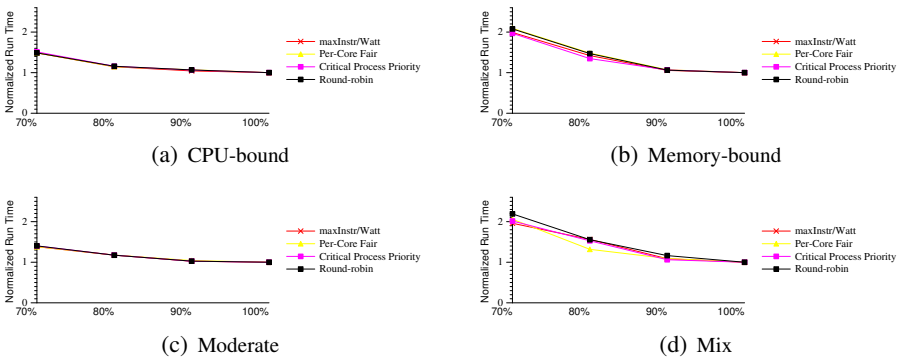
4.4 Results

4.4.1 No DVFS

This section discusses the experimental results achieved for scheduling policies on Intel Core i7 when only process suspension knob is used by the scheduler to maintain power envelope. Figure 4.3 shows the normalized runtimes for all the sets of workloads on Core i7 for the first two policies, max instruction/watt and per-core fair. As per the results obtained by Singh et al. [34], the max instruction/watt policy should favor the CPU bound workloads while the per-core fair policy should favor the memory bound workloads. But this distinction is not so clearly visible here. This is because of the difference between the runtimes of various workloads. To achieve the best possible runtime to complete all workloads, the scheduler should always select the shorter workloads for suspension. This will ensure, the longest workload which is critical to the total runtime is never throttled and hence the impact on the runtimes is minimal. This is clearly visible for the execution times of CPU bound benchmarks when using Max Instruction/Watt policy. The CPU bound applications *ep* and *gamess* have lowest computation intensity as well as execution times. As a result, these two applications are suspended most frequently, which doesn't affect the total execution time even when power envelope is set to 80% of peak usage. As can be seen from the results, the round-robin policy gives average performance while being fair to all the processes.

4.4.2 DVFS + Process suspension

This section discusses the results obtained on Core i7 when the scheduler uses both DVFS and process suspension to maintain the power envelope. As mentioned earlier, the scheduler uses



(a) CPU-bound (b) Memory-bound

(c) Moderate (d) Mix

Figure 4.4: Runtimes for Workloads on the Core i7 (with DVFS)

DVFS as a fine knob and process suspension as a coarse knob to maintain the envelope. The Intel Core i7 processor that we use for our experiments, *Intel Core i7 - 870*, supports fourteen different voltage-frequency points, also called P-states. These frequency points range from 2.926 GHz to 1.197 GHz. Also, the said Intel processor supports only chip wide DVFS capability. Hence, individual cores cannot be operated at different frequencies [11]. For our experiments, we have made models for seven frequency points (2.926, 2.66, 2.394, 2.128, 1.862, 1.596 and 1.33 GHz) and adjust the processor frequency across these points. The results of our experiments are shown in Figure 4.4.

The experiment results show that for CPU bound and Moderate benchmarks, there is hardly any difference in the execution times of different suspension policies. This result suggests that for these applications, the scheduler hardly needs to suspend the processes and that regulating DVFS points proves to be enough to maintain the power envelope. This is in contrast to the results obtained by on AMD Phenom and Intel Q6600 by Singh et al. [20]. This is because Singh et al. use only two frequency points for their experiments while we use seven frequency points and across a wider range. Another result in contrast to Singh et al. is that the performance for DVFS degrades compared to the case where no DVFS is used, except for the mix workload set. The explanation for this result lies in the difference between runtimes of different applications within the workoad sets. In the case when no DVFS is used, all the processes are running at full speed. And even when one of the process 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 given higher priority over process suspension, when the envelope is breached, all the processes are slowed down and this affects the total execution time. This is further proven by the results of mix workload set. Since the difference in execution times among individual applications of this set is not so high, it shows improvement in performance over non-DVFS case.

Chapter 5

Related Work

Many researchers have explored the usage of performance counters for power estimation in last decade. This chapter discusses the significant contributions made by previous research work in this area and compares them to our work.

5.1 Run-time Power Estimation in High-Performance Microprocessors

Joseph and Martonosi [24] use the performance counters to generate power estimates for individual components of microarchitecture. They perform the power estimation for a 600 MHz Alpha 21264 model in simulator (SimpleScalar [3]) and a 200 MHz Pentium Pro hardware. For the power estimation on simulator, they rely on per usage power statistics from Wattch for each microarchitecture component and provide their own heuristic approximations for the usage of each component. On the Pentium Pro hardware, they use a shunt resistor and a multimeter to measure real power consumption of the processor chip. They use twelve performance events in their model, which requires them to rotate the sampling of events, two at a time. Their model shows low average error rates on simulator but higher error on hardware. They attribute this to the lack of better Pentium Pro power model, in the absence of which, they made conservative power consumption estimates for certain resources. Their paper doesn't detail the procedure to calculate weights for individual resource usage for making the model for Pentium Pro.

5.2 Application-Aware Power Management

Rajamani et al. [33] develop online power and performance models on a Pentium M system. They base their power model on the single event counter (*Decoded Instructions per cycle*). They don't mention the error rates obtained for their power model but as per our experience, such a simplistic model will exhibit low accuracy. As we showed in Figure 1.1, different types of instructions with similar CPI can exhibit widely varying power consumption figures. They develop their model

using small training set, which targets exercising memory hierarchy levels and stability across the runs instead of covering all possible scenarios like ours. They present two power-management solutions (PowerSave and PerformanceMaximizer) that dynamically controls the p-state of processor. Their methodology shows good results for power and performance management.

5.3 Event-driven Energy Accounting for Dynamic Thermal Management

Bellosa et al. [6] use the performance counters for online energy estimation. They use sense resistors employed on the power supply lines and A/D converter to measure the actual power consumption. They chose the performance events for the power model based on the empirical correlation. They use the *netlib* FORTRAN routine *dqed* to calculate the weights of their performance events from the set of linear equations relating the events to the power consumption. They use nine events in their power model. Their model shows less than 1% error on 11 of the 25 test programs but the outliers show error rates of up to 30%.

5.4 Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction

Lee and Brooks [26] propose performance and power prediction for large microarchitectural design space through statistical inference models formed using regression modeling. They obtain a small set of sample observations drawn uniformly at random from a large design space on Turandot processor simulator. They vary a total of twelve architectural parameters to cover large microarchitectural design space. They use this subset of design space to formulate regression model for performance and power prediction. They explore both application specific and regional models. They use cubic splines for accommodating the nonlinear relationship of certain architectural parameters with performance and power. They present their prediction results for the same applications which they used as training set. Unlike our work, their model targets the simulation frameworks and aims to reduce the simulation time required for performance and power prediction. Their model shows median and CDF errors that are higher than our model.

5.5 Decomposable and Responsive Power Models for Multicore Processors using Performance Counters

Bertran et al. [8] present a decomposable power model for Intel Core 2 Duo processor. They use a total of thirteen counters for estimating the power consumption of different micro-architectural components. They use microbenchmarks and incremental linear regression technique to come up with individual power consumption figures for components like Front end, Integer execution units, FP units, BPU, SIMD unit, etc. They do not use temperature in their model and hence fail

to account for the increase in static power consumption with temperature. Also, since their model uses thirteen different PMCs, they have to multiplex the sampling of counters and hence the model would be error prone in the face of highly transient applications. Their model shows low average error like ours.

5.6 Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events

Contreras and Martonosi [12] use five performance counters to implement their power model. They perform power estimation experiments on an Xscale system at various frequencies.They gather counter data from multiple benchmark runs assuming that the application behavior remains consistent across various runs. This also renders their methodology unsuitable for online usage. They derive power weights for frequency-voltage pairs, and form a parametrized linear model. Their model exhibits high accuracy but their validation is limited to seven applications on a single platform while we perform validation on 45 different applications spanning three benchmark sets and six platforms.

5.7 Full-System Power Analysis and Modeling for Server Environments

Economou et al. [16] use PMCs to predict power on a blade and itanium server. Their work aims to use the power estimation for dynamic control of server systems. Like us, they use application-independent microbenchmarks to profile the system under test. They use data acquisition unit to measure actual power consumption at the sample rate of one per second. Their model uses CPU utilization rate, memory access count, hard disk I/O rate and network I/O rate to estimate power for the CPU, memory, hard drive and network controller with 10% average error. Their model also suffers with the simplistic assumption that CPU utilization rate alone is sufficient to reasonably estimate processor power consumption.

5.8 An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget

Isci et al. [23] analyze power management policies to enforce a given power budget and to minimize the power consumption for the given performance target. They conduct their experiments on the Turandot [30] simulator. They get their power estimates from IBM PowerTimer instead of developing their own power model. They have developed a global power manager that leverages power-performance data available from locally available monitors per core to enforce the DVFS policies individually for each core. Unlike us, they do not explore process suspension policies to enforce power budget.

5.9 PAM: A Novel Performance/Power Aware Meta-scheduler for Multi-core Systems

Banikazemi et al. [5] present a power-aware meta-scheduler (PAM). PAM monitors the performance, power and energy of the system by using performance counters and in-built power monitoring hardware. It then uses this information to dynamically remap the software threads on multi-core servers for higher performance and lower energy usage. Like our scheduler, PAM runs in user space and hence does not require kernel changes. Their framework is flexible enough to substitute the hardware power monitor with performance counter based power model.

Chapter 6

Future Work

6.1 Current Measurement PCB

One of the major limitations of the current experimental setup is the max sampling frequency of one sample per second of the power meter. Also, since the power meter is plugged between the mains supply and the power supply unit of the test machine, we can only measure the power consumption of entire machine instead of just the processor chip. To overcome this limitation in the future work, we will make use of a custom made current measurement PCB that can be placed between the ATX power supply and the ATX connections on the motherboard. This PCB will use the current transducers ([13]) to sense the current consumed by processor and other components of motherboard. The voltage output of the current transducers will be measured using multi-channel USB based Data Acquisition device ([14]).

6.2 Implementation on Tilera Processor

For this thesis, we have tested our model only on Intel and AMD processors. For the future work, we would like to extend the implementation of the model on 64-core Tilera processor *TILEPro64* [15]. As per the currently available information, this processor has limited and very different set of performance counters compared to Intel and AMD processors used in our thesis, and hence, will stretch the implementation of our model.

Chapter 7

Conclusion

In this thesis, we have validated the power estimation model presented by Singh et al. [34, 35] on three different microarchitectures (Intel Core Duo, AMD Opteron 8212 and Intel Core i7). We have demonstrated the scalability and portability of the model by showing that it can estimate per core power estimation with high accuracy ($< 5\%$ overall median error) on the processors ranging from dual core to quad core. The model showed high accuracy across both single-threaded (SPEC 2006) and multi-threaded (SPEC-OMP and NAS) benchmarks, and for both integer arithmetic and floating point arithmetic intensive applications. We extended our methodology to work with Hyper-Threading enabled processor and created power estimation model for each logical core instead of each physical core. The estimation accuracy for Hyper-Threading enabled processor was within the range (2.63% overall error) of other results. We demonstrated that our methodology is simple enough to work with both physical and logical cores. We showed the cumulative distribution of errors for each collected sample and showed that overwhelming majority of samples showed low error rate.

We gained insights into the working and usage of performance counters and how to analyze the correlation results to choose the best counters available for model formation. We concluded that high correlation alone cannot be chosen as metric for individual counter selection. We analyzed the correlation between the counters themselves to choose the counters that provide non-redundant information.

We analyzed the performance of our model to study the sources of error. Even though, our model shows low median error, we observed high intermittent error peaks. This is because of the limitation of our power meter that can sample power values at the maximum sample rate of one second, which is not enough to capture behavior of fast transient phases of benchmark applications. The accuracy of our model is highly dependent on availability of counters that are relevant to power consumption and cover more comprehensive events.

We use the live power management application written by Singh et al. [34, 35] to schedule tasks on Core i7 machine maintaining a strict power envelope. We augmented the application to make it scalable across various DVFS performance points, and introduced two new process suspension policies. We showed that applications with unequal execution times can be scheduled under the

given power envelope without using DVFS and with minimal performance loss. We showed that our scheduler can use both DVFS and process suspension judiciously to schedule tasks under power budget.

Appendix A

Alternate Power Model

This appendix details the methodology and results of an alternate power estimation model that was explored during this thesis.

A disadvantage of a statistical model based on multiple linear regression is that the model provides little insight into the power consumption of individual components of microarchitecture. The weights associated with counters in the multiple regression power model don't necessarily represent the individual contribution of the corresponding component to the total power. This is because a finite degree of colinearity exists between the counters as discussed in Chapter 3. As a result, the model sometimes contain negative coefficients for the counter that should have been positively correlated with the power consumption. This makes the model structure counter-intuitive. To overcome this limitation, we explored making an alternate model based upon the work published by Bertran et al. [8].

The idea behind making the alternate model is to identify individual microarchitectural components that add significant contribution to the overall power consumption of core and calculate weights for the activity factor of these components that correspond to their contribution. Also, we separate the static and dynamic power usage of the core. As shown in Equation A.1, we calculate the total power as the sum of idle power, and increment in static and dynamic power. The increase in static power is calculated using the core temperature as per Equation 3.4 in Chapter 3. The increase in dynamic power consumption is calculated by the linear sum of weighted activity factor of relevant performance events as stated earlier and depicted in Equation A.2. Since, the activity factor of various events is negligible during the processor idle state, we neglect event activity at idle and use the absolute values to calculate the increment in dynamic power consumption.

$$P_{core} = P_{Idle} + P(Static)_{inc} + P(Dynamic)_{inc} \tag{A.1}$$

where $P_{Idle} = Idle\ Power$
$P(Static)_{inc} = Increase\ in\ static\ power\ consumption$
$P(Dynamic)_{inc} = Increase\ in\ dynamic\ power\ consumption$

$$P(Dynamic)_{inc} = \sum_i r_i * W_i \tag{A.2}$$

where $P(Dynamic)_{inc} = Increase\ in\ dynamic\ power\ consumption$
$r_i = Event\ activity\ ratio\ of\ event\ i$
$W_i = Calculated\ weight\ of\ event\ i$

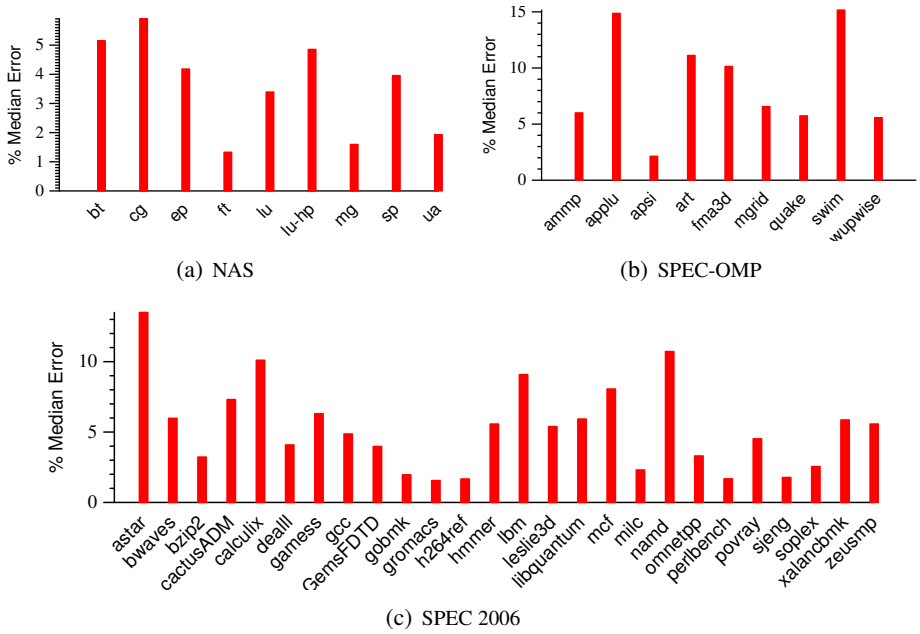
We validated this power model only on Core i7 machine and the results shown here are very preliminary. As suggested by Bertran et al., we target eight power components. These power components and their related performance counters and calculated weights are listed in Table A.1. As shown in the table, we were able to cover all the eight components using only eight counters. To calculate the weights of power components individually, we used the modified version of our existing microbenchmarks. We created one microbenchmark targeting each power component. We designed the microbenchmarks such that we calculate the weights of the components incrementally from the top of the table to the bottom. The first microbenchmark exercises only the processor front end, the second microbenchmark exercises both front end and branch prediction unit and so on. The weight obtained from the first microbenchmark is plugged in the results of second microbenchmark to calculate the weight of branch prediction unit. Then the weights of both front end and BPU are used to calculate weight of Integer execution units. We continue this process until we calculate all the weights. The weights received using this procedure were used to form the model and validated against few test benchmarks initially. During the initial validation, it became evident that we were overestimating the weights of some of the components like front end and x87 floating point units. This can be attributed to the insufficiency of diversity in our training set. We calibrated our microbenchmarks to increase the accuracy of the model. We will develop more diverse microbenchmarks as part of future work.

The estimation error for the model on Core i7 machine in shown in Figure A.1 for SPEC2K6, SPEC OMP and NAS benchmark sets. The model exhibits median error of 4.59% on SPEC2K6, 8.09% on SPEC OMP and 3.78% on NAS, with an overall error of 4.87%. The comparison of average actual and estimated values of core power consumption is shown in Figure A.2. Although

Power Component	**Related Performance Counters**	**Calculated Weight**
Front End	UOPS_RETIRED:ANY	0.823
Branch Prediction Unit	BR_INST_EXEC:ANY	1.524
Integer Execution Units	UOPS_EXECUTED:PORT015-BR_INST_EXEC:ANY-FP_COMP_OPS_EXE:X87-FP_COMP_OPS_EXE:SSE_FP	1.723
L1 Cache Accesses	L1D_ALL_REF:ANY	2.403
L2 Cache Accesses	L2_RQSTS:REFERENCES	34.184
DRAM Accesses	MEM_UNCORE_RETIRED:LOCAL_DRAM	1293.394
x87 Floating Point Units	FP_COMP_OPS_EXE:X87	2.261
SSE Floating Point Units	FP_COMP_OPS_EXE:SSE_FP	27.532

Table A.1: Power components and related performance counters

the model accuracy is worse compared to the statistical model, the model shows potential and at least part of the error can be attributed to lack of proper microbenchmarks. We will improve the model implementation as part of future work.

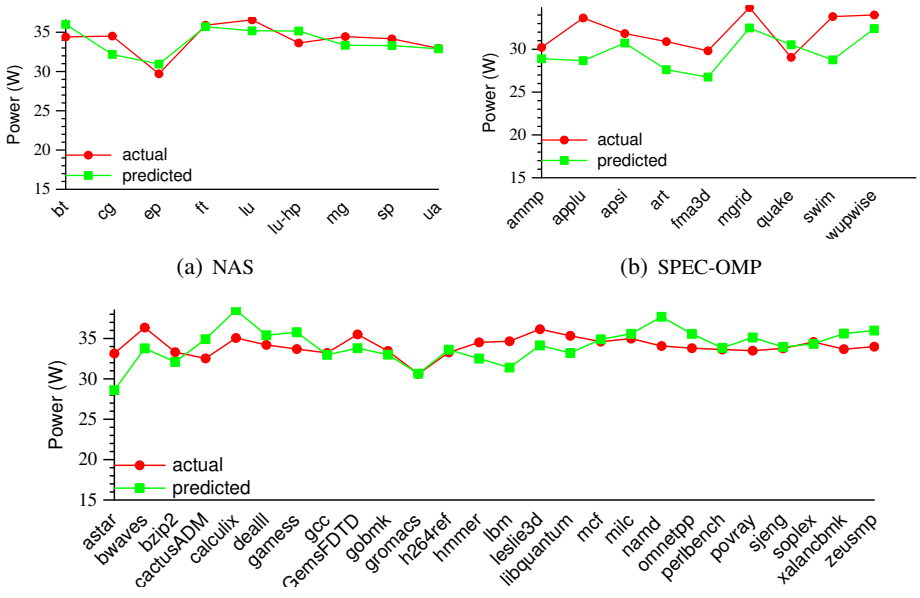


(a) NAS

(b) SPEC-OMP

(c) SPEC 2006

Figure A.1: Median Estimation Error for Intel Core i7



(a) NAS

(b) SPEC-OMP

(c) SPEC 2006

Figure A.2: Estimated vs. Measured Error for Intel Core i7

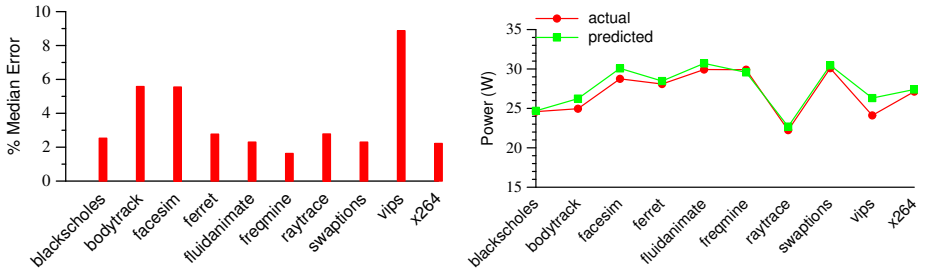
Appendix B

PARSEC Results on Core i7

The Princeton Application Repository for Shared-Memory Computers (PARSEC) [9] is a multi-threaded benchmark suite that focuses on emerging workloads. PARSEC aims to represent the shared memory programs for future multi-core processors. PARSEC workloads differ from SPEC-OMP in that unlike the later, they are not HPC focused. The PARSEC suite includes programs from desktop and server domains. One of the major differences in the characteristic of PARSEC applications to the multi-threaded benchmark suites that we have used (SPEC-OMP and NAS) is that the former constitutes of applications that have significant portion of the program single-threaded. Hence, evaluating our model against the PARSEC suite helped us in validating the methodology in a scenario where the program behavior changes from single-threaded to multi-threaded dynamically.

Figure B.1(a) details the median errors obtained on the various applications of PARSEC. As shown, all the applications show less than 10% median error while most of the applications show less than 5% median error. The median error across all the samples collected for the PARSEC benchmark suite was 2.69%.

Figure B.1(b) compares the actual to estimated mean power consumption values across the run of



(a) Median Estimation Error for PARSEC on Intel Core i7

(b) Estimated vs. Measured Error for PARSEC on Intel Core i7

Figure B.1: PARSEC Estimation Results

PARSEC applications. The figure verifies that our model's estimated power values closely follow the empirical values. These results further adds to our confidence that our model is application independent.

Appendix C

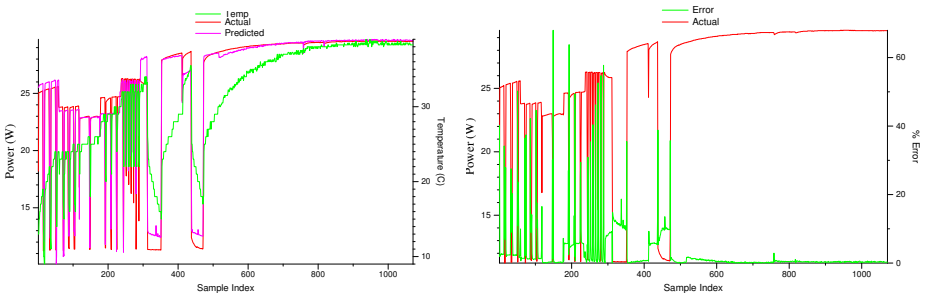
Custom Test Benchmark

Tbench is a custom benchmark developed for following purposes:

- Analyze the performance of model when the benchmark is exercising, in isolation wherever possible, different micro-architectural features of the core, namely: FP, ALU, register file, external memory, branch predictor and BTB, and RS and ROB.
- Analyze the error peaks observed in other benchmarks when a sharp change in power consumption occurs.
- Analyze the relationship between temperature and power consumption.

Tbench benchmark is a collection of small code snippets that have been written in assembly to exercise the above mentioned micro-architectures. The benchmarks runs for 1200 seconds and is distributed as follows:

- 0-60: Register file
- 61-120: FP + low stalls
- 121-180: FP + high stalls
- 181-240: Register file
- 241-300: Branch + high stalls
- 300-360: Sleep
- 360-480: External memory
- 480:540: Branch + high stalls
- 540-1200: External memory + FP + ALU + high stalls



(a) Estimated vs. Measured Power vs Temperature during the run of Tbench

(b) Error peaks during the run of Tbench

Figure C.1: Tbench power estimation results

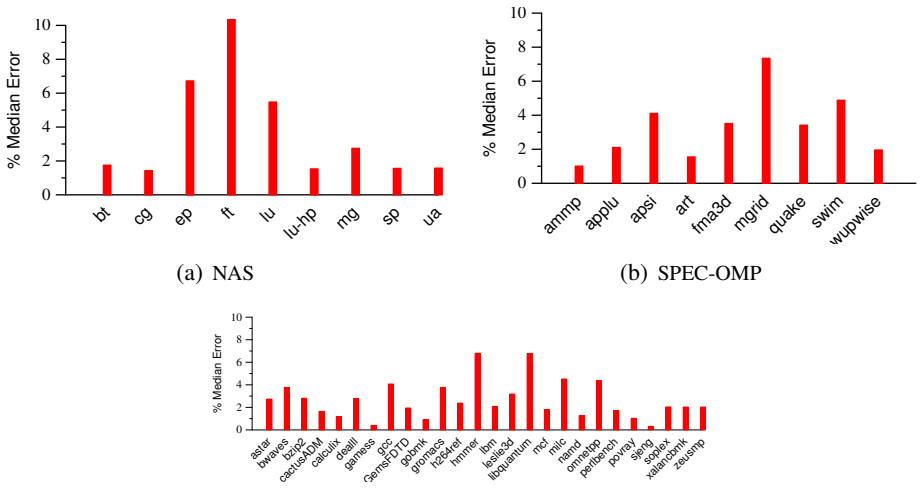
As the Figure C.1(a) shows, a capacitive relationship is observed between the temperature and power. Including the temperature in the model helps the predicted curve follow the non-linear growth of power consumption. The graph also shows that power consumption can be used to approximate the temperature of the sensor dynamically, but it would be extremely difficult to predict power consumption by observing just the core temperature since small approximation error in temperature would lead to highly erroneous power consumption figures.

As the error graph C.1(b) for the Tbench depicts, our model suffers with high error peaks whenever there is a sharp change in power consumption of the core. This is because of the limitation of our validation methodology and not because of limitation of the statistical linear model. Our power meter can sample the power consumption only once every 1 second. Hence we accumulate the performance counters over one second and normalize them per cycle to compare with the power consumption which is an instantaneous metric. This works well when the power consumption curve is not sharp over the period of one second. But when the power consumption changes significantly over the period of one second, the model tends to predict the power consumption value averaged over the period of one second. Hence, we see error peaks with sharp changes in power. We believe that when our model will be used at a higher sampling rate, every 10 ms or less, the error peaks would be less severe.

Appendix D

Core i7 Results with Hyper-Threading Enabled

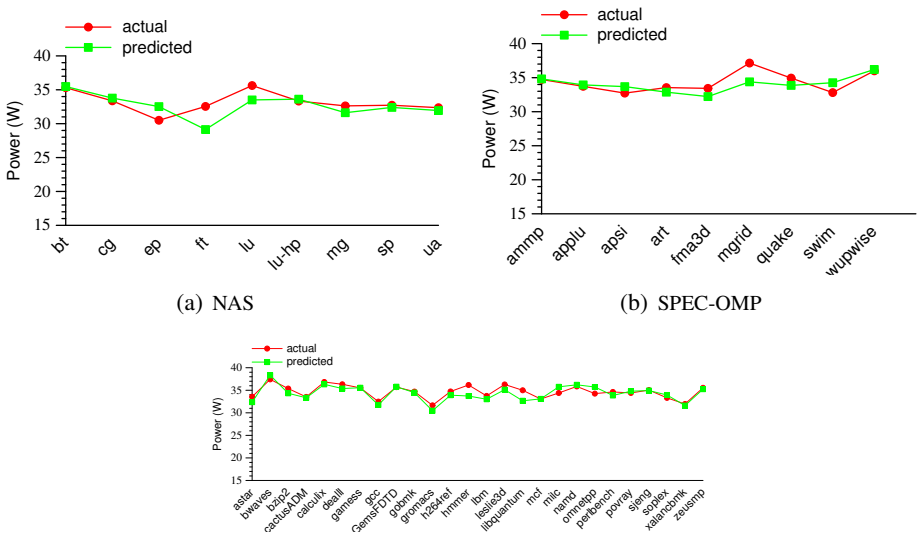
This appendix gives the detailed estimation results when our methodology is used for estimating power consumption for each logical core on Intel Core i7 when Hyper-Threading is enabled. For this experiment, we estimate the power consumption for each logical core instead of physical core. This is possible because Intel Core i7 provides individual set of performance counters for each logical core. For the resources that are shared dynamically between the cores, or the resources that are Hyper-Threading agnostic, the performance counters count only those events that are associated with respective logical core.



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

Figure D.1: Median Estimation Error for Intel Core i7



(a) NAS (b) SPEC-OMP

(c) SPEC 2006

Figure D.2: Estimated vs. Measured Error for Intel Core i7

Bibliography

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

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

[3] T. Austin. SimpleScalar 4.0 release note. http://www.simplescalar.com/.

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

[5] 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*, number 39, Nov. 2008.

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

[7] Y. Ben-Itzhak, I. Cidon, and A. Kolodny. Performance and power aware cmp thread allocation modeling. In *HiPEAC*, pages 232–246, Jan. 2010.

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

[9] C. Bienia, S. Kumar, J. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proc. IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81, Oct. 2008.

[10] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. 27th IEEE/ACM International Symposium on Computer Architecture*, pages 83–94, June 2000.

[11] J. Casazza. Intel core i7-800 processor series and the intel core i5-700 processor series. White Paper, Intel Corporation, 2009.

[12] 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*, pages 221–226, Aug. 2005.

[13] L. Corporation. Intel current transducer lts 25-np. Datasheet, LEM, Nov. 2009.

[14] N. Corporation. NI bus-powered m series multifunction daq for usb. http://sine.ni.com/ds/app/doc/p/id/ds-9/lang/en, 2009.

[15] T. Corporation. Tilera tilepro64 processor product brief. http://www.tilera.com/sites/default/files/productbriefs/PB019_TILEPro64_Processor_A_v3.pdf.

[16] 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.

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

[18] S. Eranian. Perfmon2: a flexible performance monitoring interface for Linux. In *Proc. 2006 Ottawa Linux Symposium*, pages 269–288, July 2006.

[19] Gary Perlman. *STAT Statistical Data Analysis: Free Data Analysis Programs for UNIX and DOS*, 2001.

[20] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. pages 135 –146, aug. 2010.

[21] M. Govindan, S. Keckler, and D. Burger. End-to-end validation of architectural power models. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 383–388, July 2009.

[22] Intel. *Intel Architecture Software Developer's Manual: System Programming Guide*, Mar. 2010.

[23] C. Isci, A. Buyuktosunoglu, C. 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*, pages 347–358, Dec. 2006.

[24] 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*, pages 135–140, Aug. 2001.

[25] F. Kerlinger and E. Pedhazur. *Multiple regression in behavioral research*. Holt, Rinehart and Winston (New York), 1973.

[26] B. Lee and D. 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*, pages 185–194, Oct. 2006.

[27] T. Li and L. K. John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31(1):160–171, 2003.

[28] S. M. and L. H. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

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

[30] 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*, pages 452–457, Feb. 1999.

[31] T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer*, 34:52–57, 2001.

[32] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[33] 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*, pages 39–48, Oct. 2006.

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

[35] K. Singh, M. Bhadauria, and S. McKee. Real time power estimation and thread scheduling via performance counters. *Proc. Workshop on Design, Architecture and Simulation of Chip Multi-Processors*, Nov. 2008.

[36] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, jan 1904.

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

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

[39] Texas Instruments. Cmos power consumption and cpd calculation. `http://focus.ti.com/lit/an/scaa035b/scaa035b.pdf`, June 1997.

[40] V. Weaver and S. McKee. Can hardware performance counters be trusted? Technical Report CSL-TR-2008-1051, Cornell University, Aug. 2008.

[41] D. Zaparanuks, M. Jovic, and M. Hauswirth. Accuracy of performance counter measurements. Technical Report USI-TR-2008-05, UniversitÃ della Svizzera italiana, Sept. 2008.