PMMP: A POWER MODEL FOR MULTI-CORE PROCESSOR SYSTEMS

by

Nasser Ghazali-Beiklar
B.Sc., Ferdowsi University, 1994

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

In the School
Of
Computing Sciences
Faculty of Applied Science

SIMON FRASER UNIVERSITY

Summer 2011

Approval

Name: **Nasser Ghazali-Beiklar**

Degree: **Master of Science**

Title of Project: **PMMP: A Power Model for Multi-core Processor Systems**

Examining Committee:

Dr. Janice Regan,
Chair
Computing Science
Simon Fraser University

Dr. Alexandra Fedorova,
Senior Supervisor
Computing Science
Simon Fraser University

Dr. Lou Hafer,
Supervisor
Computing Science
Simon Fraser University

Dr. Anthony Dixon,
SFU Examiner
Computing Science
Simon Fraser University

Date Approved: May 24, 2011

Declaration of
Partial Copyright Licence

Abstract

Today, power is increasingly becoming the key limitation in CPU performance; therefore, power consumption is an important issue for computer manufacturers. While manufacturers main design challenge is reducing the power to decrease heat and energy consumption, they are also concerned with designing smaller devices which will work longer while consuming the same amount of energy without losing the performance.

In this research we have focused on power consumption optimization. The goal is to answer the following question: Can we find a method to save energy and CPU expenses, based on scheduling, without a significant loss in performance?

We create a power model for multiple cores as opposed to a single core. Through extensive validation using SPECCPU2006 benchmarks, we show that the model, developed by the proposed methodology, improves energy delay by 16%.

Dedication

To All My Loved Ones

Acknowledgements

I would like to thank Sasha Fedorova, without whose help this project would ever have materialized.

I would like to give a special thanks to Dr. Anthony Dixon for his great effort for editing this report.

I would also like to thank Graham and all my colleagues in the systems lab at SFU.

Table of Contents

Approval ..... **ii**
Abstract ..... **iii**
Dedication ..... **iv**
Acknowledgements ..... **v**
Table of Contents ..... **vi**
List of Figures ..... **viii**
List of Tables ..... **ix**
Glossary ..... **x**
Introduction **1**
Motivation **3**
Background **4**
Experimental Environment **6**
Project Experiments: **8**
Baseline Idle Power 8
All Benchmark Singles 8
Core Jumping 9
Analysis **11**
Metrics 11
Energy Prediction Function 11
Results: **15**
Base-line idle power 15
Solo measurements 15
Co-scheduling measurements 16
Prediction Function 23
Conclusion **26**
Appendices **28**
Appendix A 29
Power Analyzer Datalogger Model 380803 29
Communicating with the Power Analyzer 29
Data Structure 32
Appendix B 33
MCF 33

MILC 33
GCC 34
Sphinx 34
Gobmk 34
Povray 35
Gamess 35
Appendix C 36
Bibliography 38

List of Figures

Figure 1: Architect of an Intel Multi-core processor.. 2
Figure 2- Experimental environment topology 6
Figure 3: Experiment Setup 9
Figure 4: Application properties.. 16
Figure 5: Energy consumption change when running with different applications in scenario 1..... 19
Figure 6: Energy consumption change when running with another application in scenario 2. 19
Figure 8: Energy Delay changes in different scenarios.. 21
Figure 9: Energy Delay changes in different scenarios.. 22
Figure 10: Success rate to predict the optimum scenario for two co-scheduled applications..... 26

List of Tables

Table 1: List of selected applications from SPEC2006 benchmarks 8
Table 2: Runtime and power consumption (solo measurement)..... 15
Table 4: Computed weight vector for scenario 1 and scenario 2.. 23
Table 5 : Prediction results in different strategies..... 25
Table 6: Function code table..... 30
Table 7: Optimum solution based on experiments..... 37

Glossary

NUMA	Non-Uniform Memory Access
IPC	Instructions Per Cycle
DVFS	Dynamic Voltage and Frequency Scaling
PMMP	Power Model for Multi-core Processor

Introduction

Saving power is a primary goal in today's computer design. The key concerns are to extend the battery life and reduce thermal dissipation and energy consumption in desktop and server systems. When we talk about system power budget, processors consume a significant portion of the overall budget. The emergence of multicore and multithreaded processor designs has introduced new complexities into platform and operating system support for processor power management (PPM). As we know, the CPU frequency used to increase by a factor of two every 18 months, but today this trend has ceased due to excessive power consumption and heat dissipation. However, still we have more powerful computers each year. This performance improvement is not only because of increased CPU frequency but because of other factors as well, such as increasing the number of cores, improved memory technology, and improved I/O techniques. All manufacturers now use multi-core processors in their new products and face unavoidable challenges, like resource contention, when the cores use the same resources.

Computer manufacturers and operating system designers wish to know as much as possible about system power consumption so they can design more efficient computers and develop power-aware scheduling algorithms and dynamic power management policies. In a multi-core CPU system, applications consume less power when clustered on the same memory domain, but they may run longer because of shared resource contention. Therefore, it is important to know which scheduling decision results in a lower energy delay: scheduling applications on the same domains, or scheduling them on different domains. In the first case we get worse performance but lower power consumption while in the second case we get better performance but higher power consumption; this is the problem. In this study we design a power model to help scheduler algorithms make online decisions that will resolve this problem. It is done by finding a model with an optimum balance of power and performance to gain the best energy delay. Energy delay is defined to be the product of energy and time and we use to find a balance between energy consumption and performance.

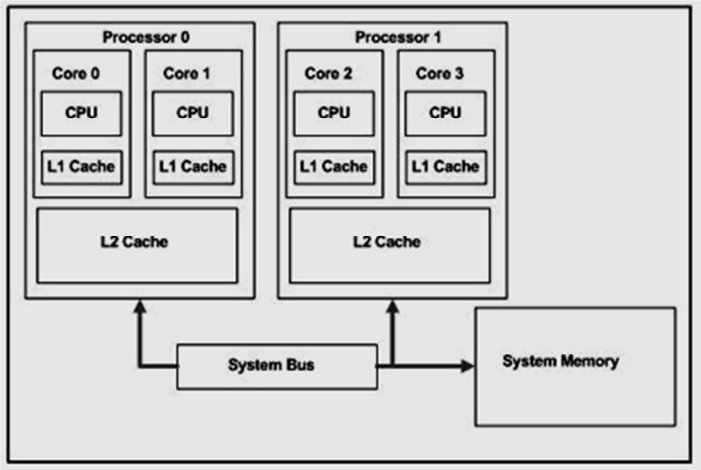


Figure 1: Architect of an Intel Multi-core processor. Core 0 and core 1 are located in the same memory domain. They share L2 cache. Core 2 and core 3 are in the same memory domain as well. Core 0 and core 2are in a different memory domain because each core uses a different L2 cache.

Because of a lack of power sensors in the majority of modern systems, temperature sensors are often used as an alternative way to measure power consumption. The problem with this approach is that there is a delay in reading power fluctuations; therefore, hardware performance event counters were used to estimate the CPU power consumption. This is an efficient method because it has a very low overhead and online monitoring is easily accomplished by the scheduler.

In this study, we use four hardware performance event counters and measure the power consumption using a power analyzer device. This helps us to be more realistic about our results because we can validate our proposed model using these data and consequently we will have an accurate model. After gathering the results and finding a prediction function by observation, we finally propose a power model based on hardware performance counters. By applying this model we are able to predict energy delays in different scenarios in terms of co-scheduling the threads on a multicore processor system, which can be used to implement an online power-aware scheduler.

Through extensive validation using SPECCPU2006 benchmarks, we show that the model which we develop by the proposed methodology predicts the power consumption in about 84% of cases successfully and it results in a 16% improvement in energy delay.

Motivation

The goal of this study is to determine how the space sharing of a machine among threads affects power consumption. More specifically, we would like to measure on real hardware how power consumption changes for memory/compute intense threads when they are scheduled in a variety of configurations. By conducting an exhaustive power consumption survey we develop a comprehensive power model that can be used to predict the expected energy delay of a given scheduling configuration. Such a power model can be applied in scheduler algorithms to make them power-aware. As the number of cores is expected to rapidly grow in the near future, the situation where there are fewer threads than cores becomes a reality. Those situations offer the potential for power savings through thread consolidation on fewer memory domains. However, thread consolidation can also lead to performance degradation and hence the scheduler needs to be able to accurately model both performance and power consequences of scheduling decisions in order to make the best possible choices.

Although the goal of this project is to gain a comprehensive understanding of power consumption under a variety of conditions and on a variety of hardware, in order to make the study tractable we need to start small. We need to begin in such a way as to acquire basic knowledge and develop techniques with simpler experiments where the number of unknown variables is minimized. We then build on these simpler results to explore progressively more sophisticated effects.

Background

The idea of using performance events to predict the power consumption is not a new one. Bellosa et al. [3] used this idea to develop a CPU power model for thermal management with 8 predictors in a Pentium 4 system. He also [1] developed a power model for each subsystem (CPU, Memory, IO, Disk) of an Intel XScale System. Gilberto Contreras et al. [2] developed different models for different CPU frequencies in an Intel PXA255 system. Stoess et al. [4] and Lee et al. [5] developed power models for hypervisor-based virtual machines and for run-time temperature sensing in high-performance processors respectively.

Snowdon et al. [6] developed 4-predictor power models for an Intel PXA255 system. Canturk Isci et al. [7] developed techniques to characterize power phases using performance event counters and showed that techniques using performance counters provide better power behaviour than code-oriented techniques.

Pusukuri et al. [8] demonstrated the use of a statistical methodology for developing simple power models across different CPU frequencies.

Our work also uses performance event counters to predict the power consumption, but idea is different from the previous research and studies. We use a power analyzer to measure the real power consumption regardless of thermal sensors or other power measurement methods. Our model predicts how the power consumption would change if the workload is running on cores within a single domain vs. separate domains.

We built a power model based on energy delay prediction using CPU counters and application behaviour. A scheduler algorithm could improve the power consumption using this model.

The development of the model consists of two steps:

1. We select the appropriate hardware performance counters by backtracking method, measure and rank. Then we run different benchmarks and monitor the selected hardware performance counters and power consumption. We observe which counters are correlated with the power and also which counters are correlated with other counters. After this stage we define our energy delay prediction function which is the fundamental factor in our model.

2. Next, we study the behaviour of the cores individually in a multi-core system. We measure CPU power consumption in different scenarios. From this information we propose an appropriate power model that saves energy while having a minimal impact on performance. The challenge in designing this model is to find the optimum scenario based on space sharing when we co-schedule two applications in a multi-core processor system. The power model that we have proposed will help overcome this challenge.

Experimental Environment

An Intel Xeon X5365 system (Octavia) is used for this study which has the following specification:

- 4 chips, 2 cores per chip
- 4MB shared L2 cache per chip
- 32KB L1 instruction and data caches per core
- Frequency: 2GHz - 3GHz DVFS per chip

To measure the power consumption an EXTECH Power Analyzer Datalogger 380803 is used. The sample rate of the Power Analyzer is 2.5 samples per second, and it is able to measure power between 0.1W and 2000W with an error rate ±0.5% and current between 0.001A and 20A with an error rate ±0.5%.

To avoid overhead in Octavia due to the power monitoring application, we use another computer to run the power analyzer communication software while Octavia runs the experiments as shown in Figure 2.

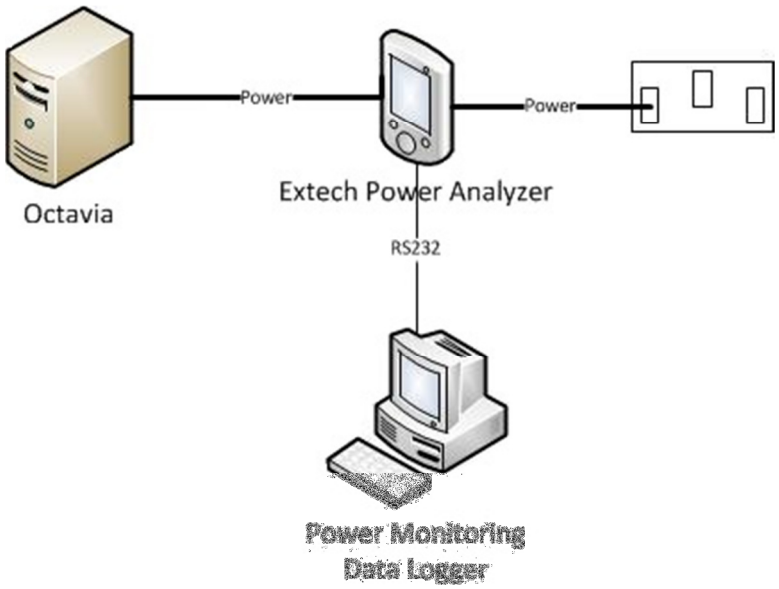


Figure 2- Experimental environment topology

Technical details for the monitoring software are described in the Appendix A.

All experiments were run at least three times to make sure that there were no obvious errors in the resulting data. The error could be caused by some factors such as cold cache (measuring performance event counters) or error in measurement device (monitoring the power consumption). We expect the result of each run are fairly the same; therefore, the assumption is variances of data sets are equal. We used Levene's test, a homogeneity variances test, to assess our assumption.

Project Experiments:

Baseline Idle Power

In order to be able to compare any measured values we need a baseline. How much power does the machine consume in the background when running only the OS? Keeping the system on for several hours (running only the OS) and constantly measuring power will give us a baseline as well as tell us how stable the power readings are. Will we get a nearly constant power the whole time or will there be constant spikes? Knowing the baseline behavior will be very helpful in analyzing future results as well as in perfecting the measurement techniques.

Application name	Application Area	Memory Intensity
mcf	Combinatorial Optimization	High
milc	Physics / Quantum Chromo Dynamics	
Gcc	C Compiler	
sphinx	Speech Recognition	Medium
gobmk	Artificial Intelligence(game of Go)	
povray	Image Ray-tracing	Low
gamess	Quantum Chemistry	

Table 1: List of selected applications from SPEC2006 benchmarks

All Benchmark Singles

Seven applications were chosen from SPEC2006 CPU and are shown in *Table 1*. These benchmarks are well known applications in terms of their memory access patterns and CPU intensity. It makes the most sense, therefore, to begin our exploration of power with these familiar applications as they will likely offer the fewest surprises. More information about the selected benchmarks can be found in Appendix B.

We ran each application solo on the machine 3 to 5 times to measure power consumption throughout its run. We observed the total power, the average power, how stable the power readings were throughout execution, and the occurrence of any phase behavior.

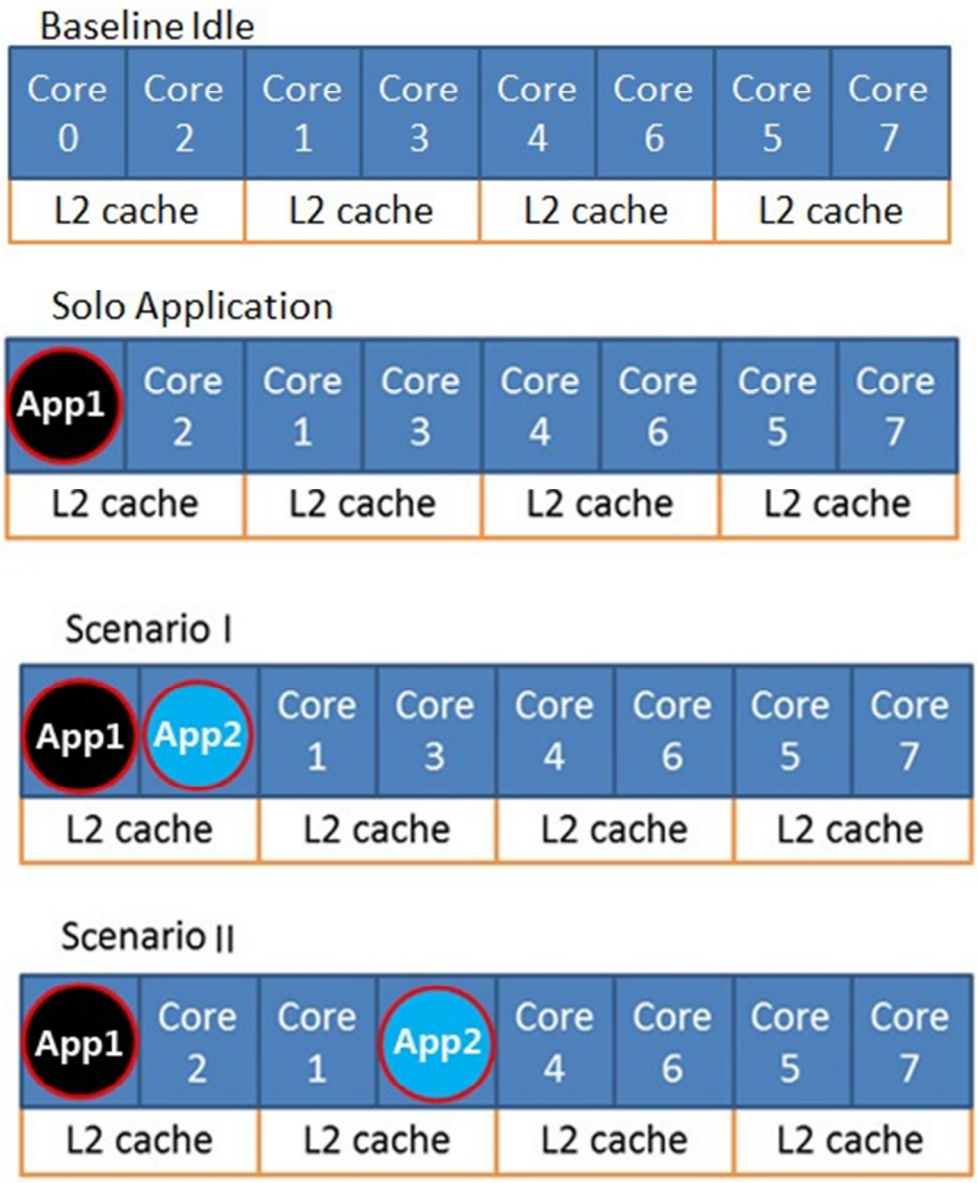


Figure 3: Experiment Setup - Baseline Idle: no application runs on the cores and all cores are idle. Solo Application: Just one core is utilized and other cores are idle. Scenario 1: two applications run on the two cores with the same memory domain. Scenario 2: Two applications run on the cores with different memory domains.

Core Jumping

Since we are interested in exploring how power consumption is affected by a scheduling policy we need to measure if core migration has any effect on power consumption.. From

the last experiment we picked the most and least power-hungry applications and ran them solo on the machine. Afterward we ran the same applications in different cores on the same machine simultaneously. There were two kinds of migration studies.

Scenario 1: Migration to core on same memory domain

Scenario 2: Migration to core on a different memory domain

Likewise the duration of runtime was also varied. The results for both the power hungry and non-power hungry benchmarks will be reported in a matrix. Figure 3 shows the different configurations of our experiments.

Analysis

Metrics

The goal is to find a model that will help improve energy consumption efficiency in a multi-core processor system. Hardware performance event counters were used to predict the energy consumption and predictions used to co-schedule different threads in the system so as to optimize the energy consumption. The factors that we considered were are as follows:

- IPC (Instruction Per Cycle)
- Cache Miss Rate
- Memory References
- Number of Pre-fetch requests

Because the number of hardware performance counters is limited in most of CPU architectures, four factors have been selected that illustrate memory access pattern and performance of running application. As described previously, hardware performance counters of Octavia were measured while combinations of selected benchmarks (Table 1) in different scenarios were running on the machine. At the same time, the real power consumption was monitored by the power analyzer. To analyze the results and find a pattern to achieve the goal of the project we determined the following metrics for each scenario:

- Run Time
- Power Consumption
- Energy Delay

Energy Prediction Function

After running all the experiments we obtained a data table containing the measured hardware performance counters, run time, and power consumption.

The next step was to find the energy delay for each application on different scenarios. In this stage, we computed the energy delay for each application in both scenarios:

- ***Scenario 1:*** running two applications on the same domain.
- ***Scenario 2:*** running two applications on different domains.

At this point, measured power consumption comes from the power analyzer. For computing the energy delay to determine a balance between power consumption and performance we need to find out the energy.

$$Energy(Joule) = RunTime\ (\text{sec}) \times Power(watt) \qquad (1)$$

"Power" is the amount of energy that is consumed by the system in one second.

Applying equation 1 gives us the Energy consumption during runtime.

Now, by applying the Formula:

$$Energy\ Delay = RunTime \times Energy = RunTime^2 \times Power \qquad (2)$$

energy delay can be computed based on the data we obtained. We are then able to determine optimal energy consumption scenario which is a balance between performance and power consumption to achieve the best energy delay.

We need to find a method to predict the optimal scenario using measured hardware counters and other information such as the real power consumption and runtime which come from our experiments.

For this purpose we start by classifying our results. We have a set of applications and each application has some properties such as hardware performance counters. Also, there are two different scenarios and we have energy delays for each scenario from our experiments. Our classification maps the co-scheduled application properties to energy delays for each scenario.

Formally, the problem can be stated as follows: given training data

$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, produce a classifier $\boldsymbol{h}: \boldsymbol{\chi} \rightarrow \boldsymbol{\gamma}$ that maps any object $x \in \boldsymbol{\chi}$ to its true classification label $y \in \boldsymbol{\gamma}$ defined by some unknown mapping $\boldsymbol{g}: \boldsymbol{\chi} \rightarrow \boldsymbol{\gamma}$ (ground truth). In our case, the problem is choosing a scheduling scenario, where x_i represents two co-scheduled applications properties and *y* is either "Scenario I" or "Scenario II".

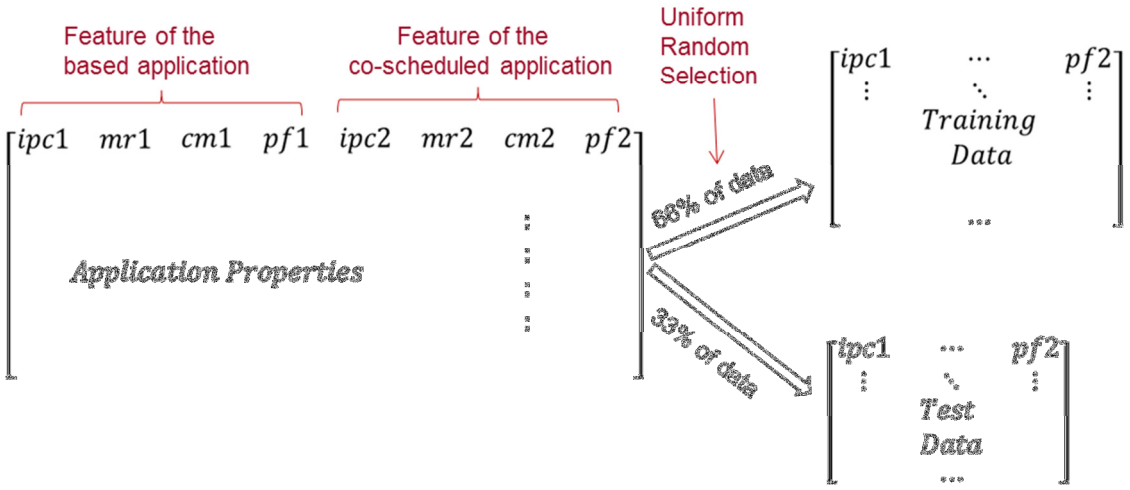
The second problem is to consider classification as an estimation problem, where the goal is to estimate a function of the form

$$E(class\ |x) = f(x; \theta) \qquad (3)$$

where the feature vector input is x, and the function f is typically parameterized by some parameters θ. In the Bayesian approach to this problem, instead of choosing a single parameter vector θ, the result is integrated over all possible thetas, with the thetas weighted by how likely they are given the training data D:

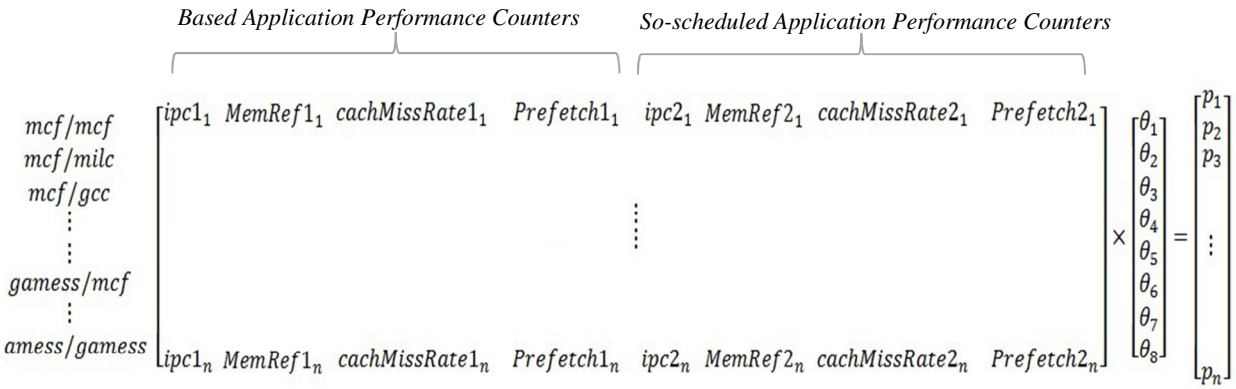
$$P(class\ |x) = \int f(x;\theta)E(\theta|D)d\theta \qquad (4)$$

The approach that we used in our project is finding a linear function to predict the energy delay by applying a weight to each counter. Suppose we create a matrix in which the rows specify all possible co-scheduled applications and the columns correspond to all measured performance event counters by using the collected data from our experiment in each scenario. Then randomly split the matrix into two matrices, matrix $A_{scenario\_i}$ for training and matrix $A'_{scenario\_i}$ for testing the model:



Total number of co-scheduled applications in our experiment is 49 and by experimental observation we found the best result is obtained when we use $\frac{2}{3}$ of data for training the model.

For scenario i we have $\overrightarrow{W_i} = (\theta_1 \dots \theta_8)$ is a vector which contains the weight to be determined of each hardware counter. Finally, $\overrightarrow{P_i} = (p_1 \dots p_n)$ is a vector specifying the energy delay for all n possible pair of co-scheduled applications. The only unknown variable is $\overrightarrow{W_i}$.



Suppose $A_{s_i} = A_{scenario\_i}$, we have:

$$A_{s_i} \times \overrightarrow{W_l} = \overrightarrow{P_l} \qquad (5)$$

Then we easily can find $\overrightarrow{W_l}$:

$$A_{s_i}{}^T A_{s_i} \times \overrightarrow{W_l} = A_{s_i}{}^T \overrightarrow{P_l}$$

$$\Rightarrow \overrightarrow{W_l} = (A_{s_i}{}^T A_{s_i})^{-1} \times A_{s_i}{}^T \overrightarrow{P_l} = Pinv(A_{s_i}) \times \overrightarrow{P_l} \qquad (6)$$

For this project, the application properties in the matrix A_{s_i} are: ipc, cache miss rate, number of pre-fetches, and number of memory references each row of for the two applications that are co-scheduled.

Results:

Base-line idle power

In the first step of the experiment the baseline-idle power consumption of Octavia is 217 watt/h. This power is measured when the machine is on and all cores are idle. The machine is turned to the operational state for five hours before sampling to make sure there is a stable situation. The result is average of two hours power consumption sampling. By observing the logged data file we found there is no significant changes in power consumption when the machine is on and all cores are idle. The minimum and maximum recorded power was 214W and 219W.

Solo measurements

Table 2 shows the result of running each application solo on the machine. All cores were idle except the one which ran the selected application. The difference between the measured power consumption (overall power) in this step and the baseline-idle power (217w) is the application's power consumption.

Application name	Runtime (s)	Overall Power (w)	Application Power (w)
mcf	797	262	45
milc	1045	282	65
gcc	121	256.6	39.6
sphinx	1063	265	48
gobmk	115	256	39
povray	303	264	47
gamess	339	268	51

Table 2: Runtime and power consumption (solo measurement)

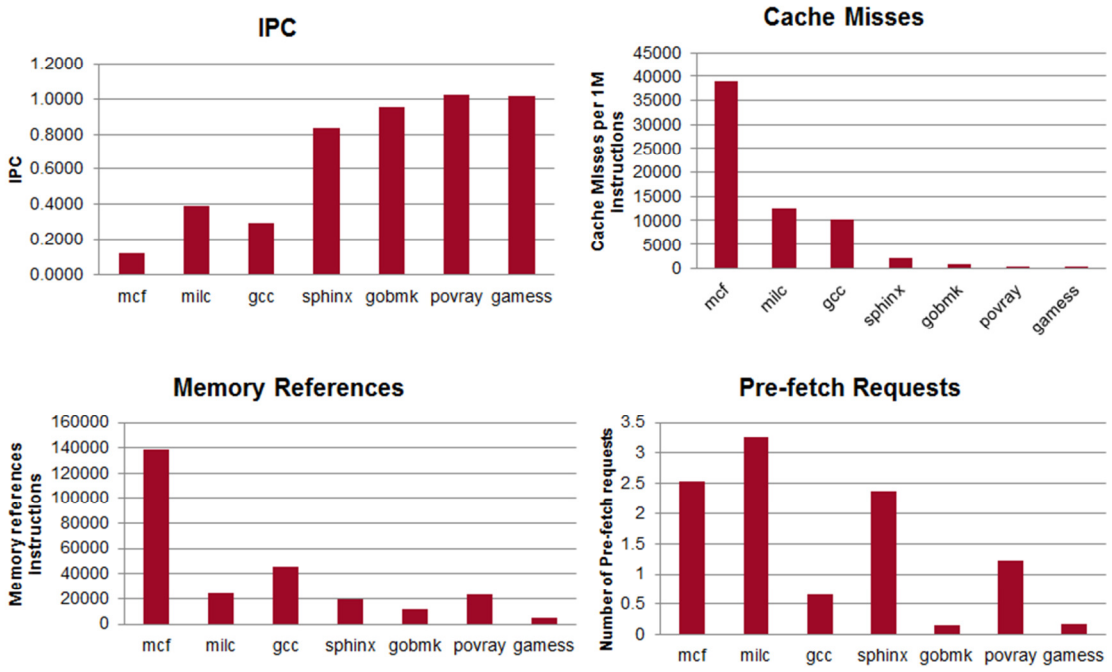


Figure 4: Application properties. IPC, Cache Misses, Memory References, and Pre-fetch Requests show the application properties for the selected benchmarks.

Co-scheduling measurements

Based on the experimental design, each application from Table 1 was picked and run with all other applications in both scenarios. IPC, cache miss rate, memory references, and number of pre-fetch requests were measured. Also we need runtime for considering the performance and finding energy delay. The other measured data is the power consumption during the experiment. Based on this information and the idle baseline power of the machine, we can find the amount of power consumed in each scenario.

Basically, energy delay is obtained by applying the equation (2) using the power consumption and runtime for each scenario. By comparing the energy delay of both scenarios we are able to find the optimum scenario in terms of energy consumption. Table 3 shows the result for mcf co-scheduled with each application. Similar information was obtained for all other benchmarks (Appendix C).

Applications		Energy Delay		Gain in Scenario 2 over Scenario1 (%)	Optimum Scenario	Strength of optimality
Based	co-schedule	Scenario 1	Scenario 2			
mcf	mcf	92.44	79.22	14.32	2	Strong
	milc	102.90	107.93	-4.89	1	Weak
	gcc	13.86	12.28	11.40	2	Strong
	sphinx	128.82	100.25	22.18	2	Strong
	gobmk	10.89	10.51	3.50	2	Weak
	povray	26.29	26.95	-0.0251	1	Weak
	gamess	27.96	27.48	0.0172	2	Weak

Table 3: Defining the optimum solution based on experiments when mcf co-schedule by other applications

The "Gain in Scenario 2 over Scenario 1" is defined to be the percent improvement in energy delay for scenario 2 where compared to scenario 1. Based on this metric we can find the optimum scenario; when the value is positive, scenario 2 is the optimum scenario and when this value is negative, the first scenario is optimum.

$$Gain\ Scenario\ 2\ over\ Scenario\ 1 = \frac{EnergyDelay_{Scenario\ 1} - EnergyDelay_{Scenario2}}{EnergyDelay_{Scenario\ 1}} \times 100 \qquad (7)$$

Strength of optimality defines whether there is a significant difference between two scenarios. We define Tolerance-Factor as a constant value which determines strength of optimality (Equation 8). If the gain in scenario 1 over scenario 2 is small then we say strength of optimality is weak which shows selecting scenario 1 or scenario 2 is immaterial in terms of trade-off between power and speed. In this case the scheduler can assign the threads to the cores based on availability. It makes the scheduler more flexible and efficient. However, if the Tolerance-Factor is exceeded then scheduling based on optimal scenario would be beneficial. Tolerance-Factor is 5% in our experiment. This factor could be changed based on migration overhead and application runtime. If the average runtime of applications that run in a machine is not long, say seconds, overhead of

migration would be higher than gain in energy; therefore, a low Tolerance-Factor is not efficient. A scheduler should define this value based on the scheduling policies.

$$Strength\ of\ Optimality = \begin{cases} Strong, if\ |Gain\ Scenario\ 2\ over\ Scenario\ 1| \geq ToleranceFactor \\ Weak, if\ |Gain\ Scenario\ 2\ over\ Scenario\ 1| < ToleranceFactor \end{cases} \quad (8)$$

Figure 5 and Figure 6 show the energy changes after co-scheduling with the second application. As we know, mcf, milc, and gcc have a high cache miss rate, povray and gamess have a low cache miss rate. Also, mcf, milc and sphinx have the high rate of pre-fetch requests. The results show the most of cache hungry applications and all applications with high pre-fetch have greater power sensitivity when they are co-scheduled with another application in scenario 1 (running on the cores with the same memory domain). However, they are less power sensitive in scenario 2 (running on the cores with separated memory domain).

If we consider mcf for example, it uses about 20% more energy when co-scheduled with milc on the cores in the same memory domain (scenario 1) in comparison to co-scheduling them on cores in different memory domain (scenario 2). This ratio for mcf and gobmk is 7%.

The reason is contention for shared resources. For example if two memory extensive applications are co-scheduled in a shared memory domain, they start using the same components such as cache memory and system bus simultaneously. Because the cache space is shared between two cache hungry applications we have higher cache miss rate and therefore the number of memory accesses increases. It increases the cost of memory accesses for CPU and the result is more CPU stalls and longer runtime. As runtime gets longer, the amount of energy required to run the application increases. Contention for shared resources also puts pressure on the memory controller and interconnects, causing the system to use more power.

CPU intensive applications such as povray and gamess show a different behaviour. By observing Figure 5 and Figure 6, we can determine if they have better energy consumption when they are co-scheduled in cores with a shared memory domain. Gobmk and gamess in the first scenario perform 0.3% better in comparison to running them in second scenario. In fact, the changes in energy for these applications in both scenarios are much lower than cache hungry applications. The reason could be similar to the previous case. Less resource contention results in a better outcome when they are co-scheduled in the same memory domain. Runtime does not suffer by co-scheduling in the same domain because there is no competition for shared resources and because we use one shared resource and instead of two separated ones, we can save to power required to operate the second resource.

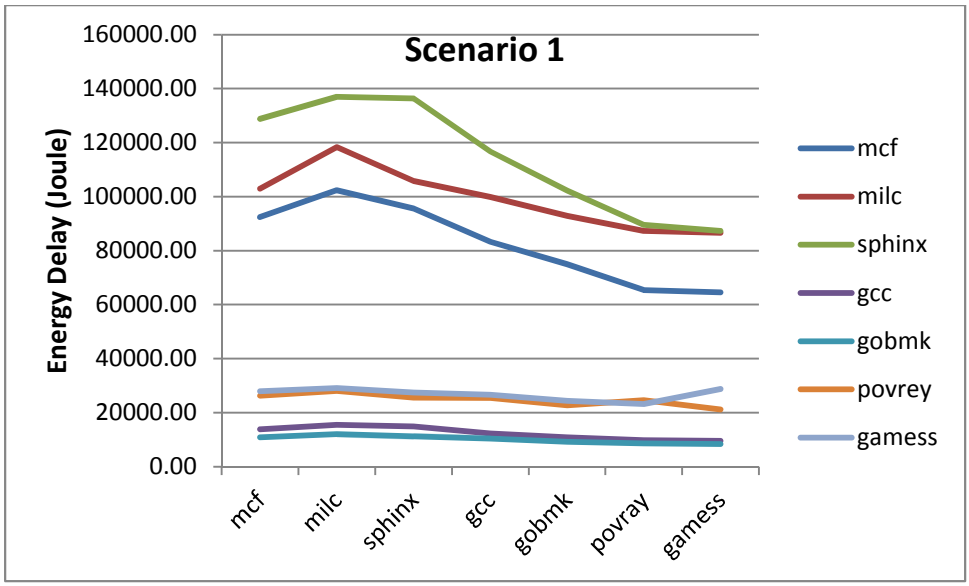


Figure 5: Energy consumption change when running with different applications in scenario 1.

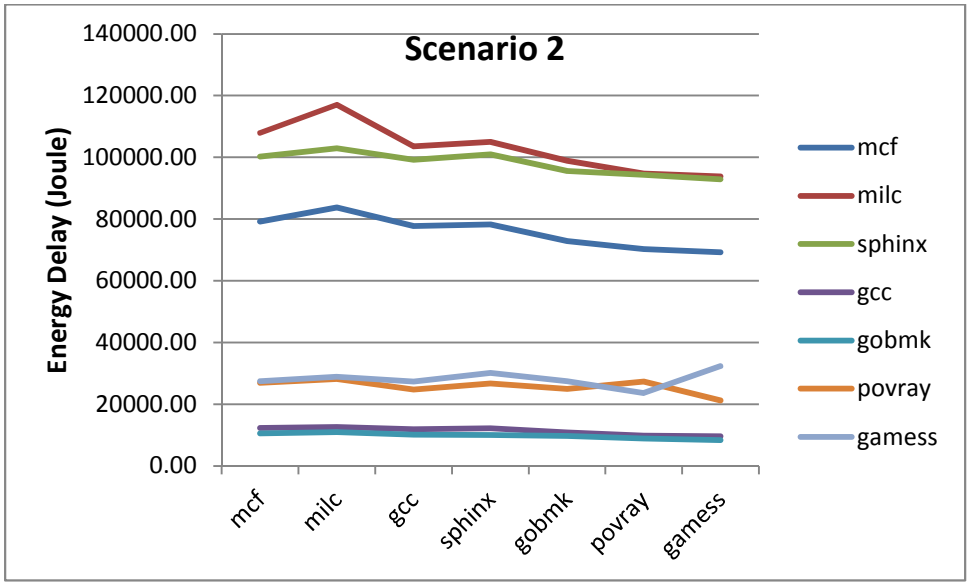


Figure 6: Energy consumption change when running with another application in scenario 2.

Figure 7 illustrates some applications with high cache miss rates such as milc, mcf, gcc and also sphinx (an application with a high rate of pre-fetch requests). All perform better in the second scenario, co-scheduling in cores with a non-shared domain. When the cache miss rate is significantly low, like gamess and povray, scenario 1 would be prefered in terms of power

consumption. This is what we should expect since, if there are many cache misses, and if we dedicate a large cache to each application, then the number of access to main memory will drop and as a result we will have fewer circuits and transistors involved in the data transfer and so power will be saved.

However, when at least one of the co-scheduled applications is CPU intensive there are fewer memory references and therefore not much access to cache and main memory. In this case the two applications would benefit from co-scheduling in cores with shared memory because the second application would be able to fully utilize the whole memory space.

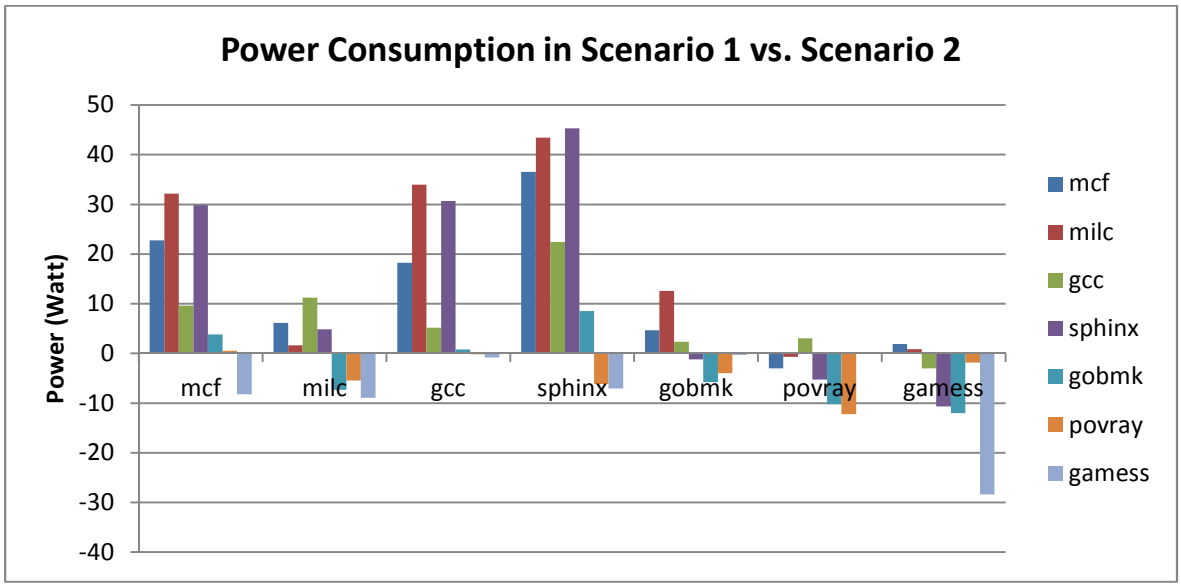


Figure 7: Power consumption changes when an application co-schedule with other applications in different scenarios. The baseline in this graph is power consumption in scenario 2 and the graph shows changes in power consumption in scenario 1 in comparison with scenario2

Figure 8 and Figure 9 show the energy delay changes for each application when running with other applications. Mcf, sphinx and gcc have lower energy delay in scenario 2 when co-scheduled with another cache hungry application. These applications perform better in scenario 1 when they run with low cache miss rate applications, such as povray and gamess.

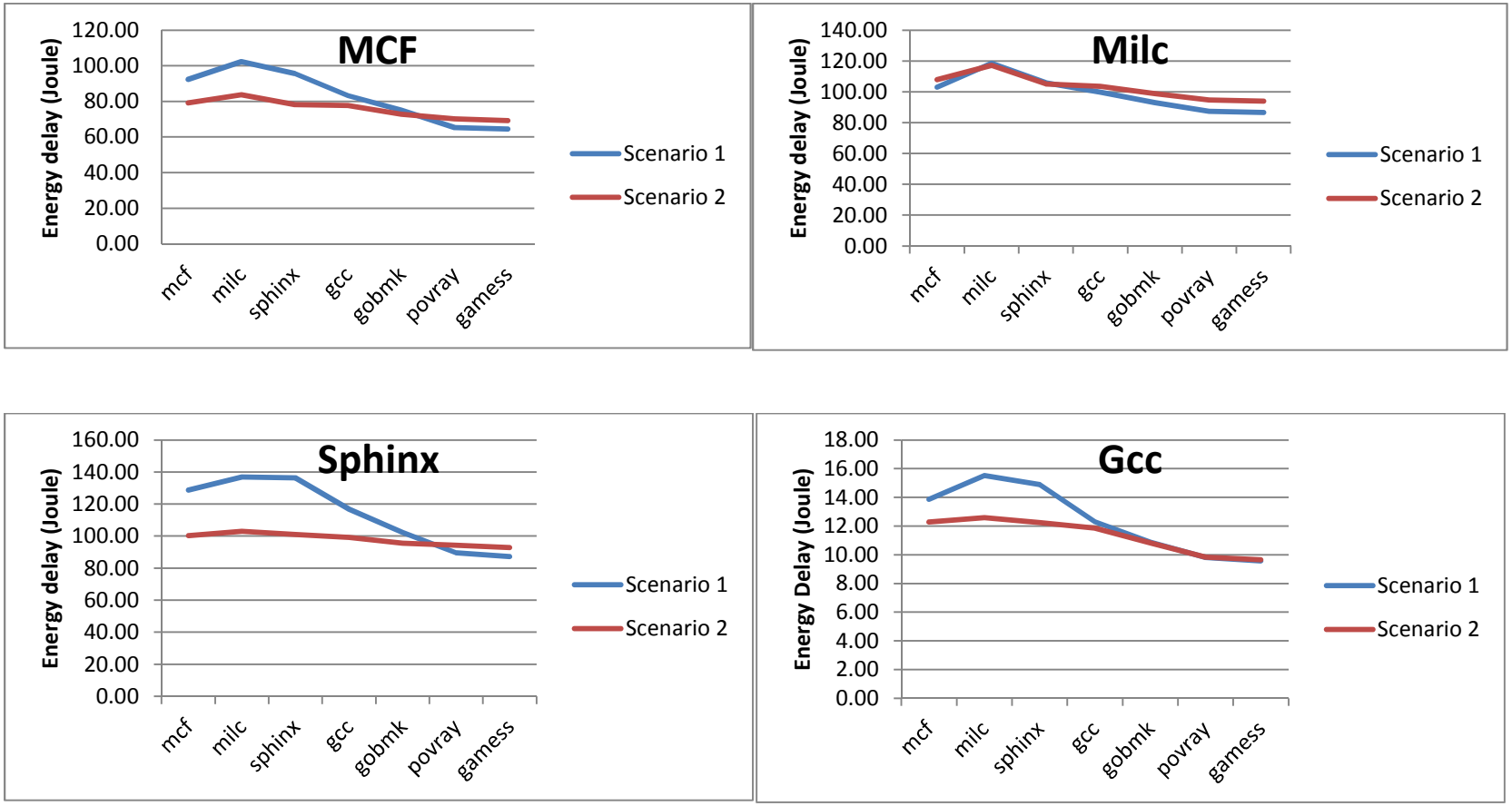


Figure 8: Energy Delay changes in different scenarios. Each graph shows the changes in energy consumption when based application (defined in the title of each graph) co-schedule with other applications.

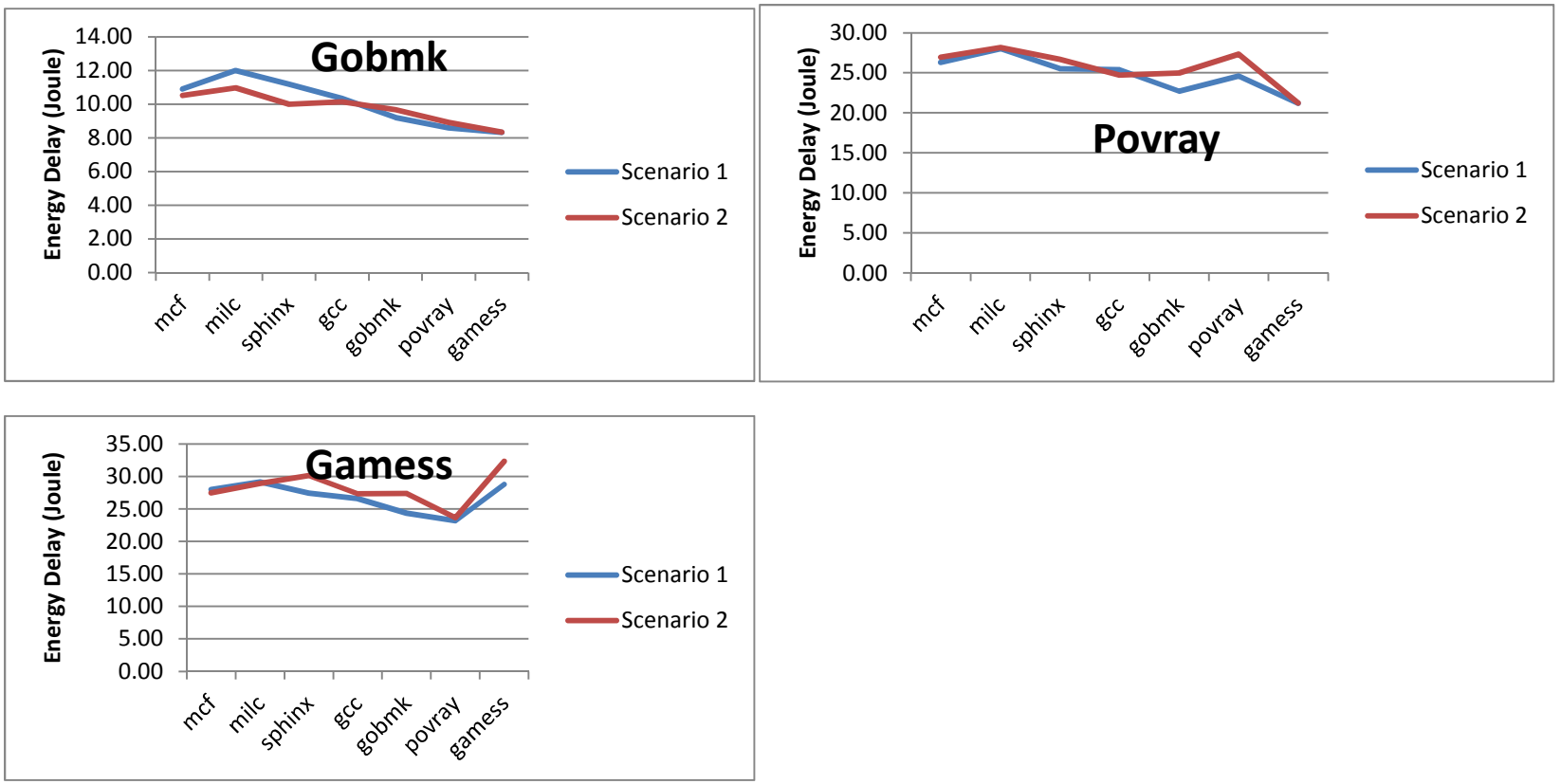


Figure 9: Energy Delay changes in different scenarios. Each graph shows the changes in energy consumption when based application (defined in the title of each graph) co-schedule with other applications.

Prediction Function

As we discussed before, for each scenario we have a table of measured performance counters for all combinations of selected applications (Matrix $A_{scenario\_i}$) as training data. Also we have measured the power consumption and have found the energy delay for each case ($\vec{P_i}$). By defining matrix $A_{scenario\_i}$ and vector $\vec{P_i}$, using the observed data in equation 6, we can find out the weight of each performance counter in scenario *i*.

	Base Application				Co-Scheduled Application			
	Ipc (ipc1)	cache miss rate (cm1)	Memory References (mr1)	Prefetch (pf1)	Ipc (ipc2)	cache miss rate (cm2)	Memory References (mr2)	Prefetch (pf2)
Weight Vector Scenario 1	-9.2716	-0.0003	0	3.022	-3.4639	-0.0015	0.0004	3.9337
Weight Vector Scenario 2	-1.1086	0.0000	0	0.1049	1.5522	0.0002	-0.0001	3.8108

Table 4: Computed weight vector for scenario 1 and scenario 2. Product of the weight vector and hardware performance counters defines the estimated energy consumption.

Now, we have two functions, $E_{Scenario\ 1}(x_1, x_2)$ and $E_{Scenario\ 2}(x_1, x_2)$, that predict energy delay in scenario 1 and scenario 2 for two co-scheduled applications, where x_1, x_2 are properties of base application and the second application that are running co-scheduled.

$$x_1 = \{ipc_1, cm_1, mr_1, pf_1\}, \quad x_2 = \{ipc_2, cm_2, mr_2, pf_2\}$$

$$E_{Scenario\ 1}(x_1, x_2) = -9.2716\ ipc_1 - 0.0003\ cm_1 + 3.022\ pf_1 - 3.4639\ ipc_2 - 0.0015\ cm_2 + 0.0004\ mr_2 + 3.9337\ pf_2 \qquad (9)$$

$$E_{Scenario\ 2}(x_1, x_2) = -1.1086\ ipc_1 + 0.1049\ pf_1 + 1.5522\ ipc_2 + 0.0002\ cm_2 - 0.0001\ mr_2 + 3..8108\ pf_2 \qquad (10)$$

The properties are as follow: *ipc* is number of retired instructions per cycle, *cm* is cache miss rate, *mr* is memory references, and *pf* is pre-fetch requests. The indices indicate the application which is running on machine.

A power aware model for co-scheduling a multi-core processor should predict the energy delay for each scenario and select the optimum case. Choosing the best scenario using values of performance counters (application properties) is a minimization problem between two linear functions. Applying the prediction functions for both scenarios to the measured performance counters gives us an estimate of energy delay and if we compare two scenarios we can predict which scenario would perform better. Therefore, the prediction function would be:

$$E(x_1, x_2) = Min\{E_{Scenario\ 1}(x_1, x_2), E_{Scenario\ 1}(x_1, x_2)\} \qquad (11)$$

Verification

To verify the validity of our model we need to show how effective and accurate the model is and we use the cross-validation method. We divide our data gathered during the experiments into testing data and training data. Previously we used training data to find the prediction function and build our model. Now we use test data (matrix $A'_{scenario\_i}$) to verify the model.

There are four ways to choose the strategy for co-scheduling two applications in our model:

> ***Strategy 1:*** use a uniform random function to select either scenario 1 or scenario 2.
>
> ***Strategy 2:*** always using scenario 1, co-schedule the application on two cores with the same memory domain.
>
> ***Strategy 3:*** always using scenario 2, co-schedule the application on cores with different memory domains.
>
> ***Strategy 4:*** use our prediction function to choose scenario 1 or scenario 2 to co-schedule the applications.

We apply test matrices $A'_{scenario\_1}$ and $A'_{scenario\_2}$, containing the application properties in scenario 1 and scenario 2, to all these strategies to evaluate the behaviour of our model. Test matrix ($A'_{scenario\_i}$) contains application properties in scenario *i*. Furthermore, we have the runtime and power consumption data of co-scheduled applications in each scenario which come from our experiments.

Co-Scheduled Applications	Experiment Result		Predicted Scenario Based on Different Strategies			
	Optimum Scenario	Strength	Always Scenario 1	Always Scenario 2	Random	PMMP
mcf/mcf	Scenario 2	Strong	Scenario 1	Scenario 2	Scenario 1	Scenario 2
mcf/povray	Scenario 2	Weak	Scenario 1	Scenario 2	Scenario 1	Scenario 2
milc/gamess	Scenario 2	Strong	Scenario 1	Scenario 2	Scenario 1	Scenario 2
gcc/mcf	Scenario 2	Strong	Scenario 1	Scenario 2	Scenario 2	Scenario 2
gcc/milc	Scenario 2	Weak	Scenario 1	Scenario 2	Scenario 1	Scenario 2
gcc/gamess	Scenario 1	Weak	Scenario 1	Scenario 2	Scenario 1	Scenario 1
sphinx/sphinx	Scenario 2	Strong	Scenario 1	Scenario 2	Scenario 1	Scenario 2
gobmk/gcc	Scenario 2	Strong	Scenario 1	Scenario 2	Scenario 2	Scenario 1
gobmk/sphinx	Scenario 2	Strong	Scenario 1	Scenario 2	Scenario 1	Scenario 2
gobmk/gobmk	Scenario 2	Weak	Scenario 1	Scenario 2	Scenario 1	Scenario 1
gobmk/povray	Scenario 1	Strong	Scenario 1	Scenario 2	Scenario 2	Scenario 1
povray/milc	Scenario 1	Strong	Scenario 1	Scenario 2	Scenario 1	Scenario 2
povray/gcc	Scenario 2	Weak	Scenario 1	Scenario 2	Scenario 2	Scenario 2
povray/povray	Scenario 1	Strong	Scenario 1	Scenario 2	Scenario 2	Scenario 1
povray/gamess	Scenario 1	Weak	Scenario 1	Scenario 2	Scenario 2	Scenario 1
gamess/mcf	Scenario 1	Weak	Scenario 1	Scenario 2	Scenario 2	Scenario 2
gamess/milc	Scenario 1	Strong	Scenario 1	Scenario 2	Scenario 1	Scenario 1
gamess/gcc	Scenario 2	Weak	Scenario 1	Scenario 2	Scenario 2	Scenario 1
gamess/gamess	Scenario 1	Strong	Scenario 1	Scenario 2	Scenario 1	Scenario 1

Table 5 : Prediction results in different strategies: select scenario 1 or scenario 2 based on a uniform random selection algorithm, always co-schedule applications based on scenario 1, always co-schedule applications based on scenario 2, co-schedule applications based on PMMP model

Based on equation 11 our model selects scenario 1 if $E(x_1, x_2)$ is equal to $E_{Scenario\ 1}(x_1, x_2)$ and scenario 2 if $E(x_1, x_2)$ is equal to $E_{Scenario\ 2}(x_1, x_2)$.

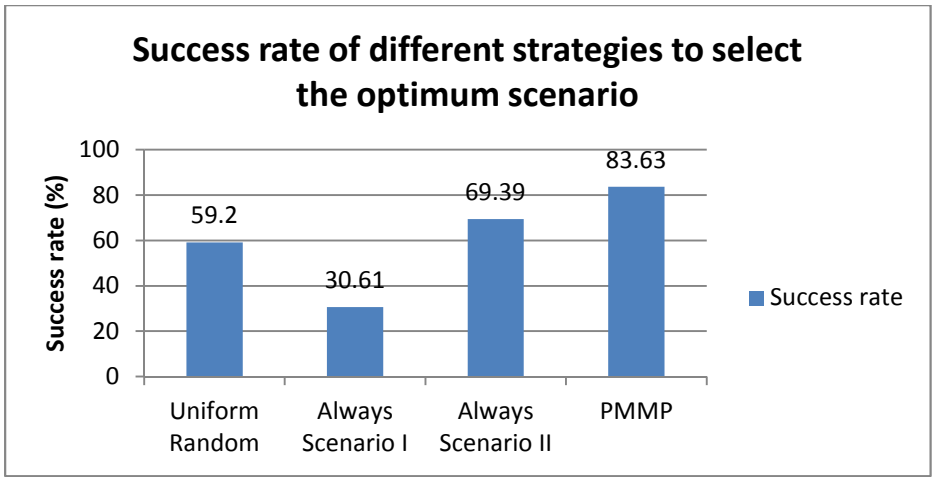


Figure 10: Success rate to predict the optimum scenario for two co-scheduled applications.

The model fails to select the optimum scenario only in 16.4% of cases and in comparison to other strategies performs at least 17% better.

Figure 10 shows the PMMP model perform 29.2% better in comparison to uniform random strategy (generally all schedulers which are not power-aware). The experiments show we can improve energy delay by 16% when we apply the PMMP strategy to predict the optimum scenario to co-scheduling the application in comparison to uniform random strategy.

The following pseudo code gives us an idea that how PMMP would be used in a scheduler:

```
Begin:
    configure hardware performance counters
Loop:
    foreach interval
    {
        read hardware performance counters for applications
        predict optimal scenario
        schedule according to optimal scenario
    }
```

Interval time could be varying based on scheduling policies and hardware specifications. For Octavia a one second interval is suggested [9].

Conclusion

We have described an energy aware scheduler model for multicore systems. The model helps schedulers to improve energy consumption by predicting the energy delay of applications and finding the optimum scenario for co-scheduling them. Because we use hardware performance counters, our approach adds very little additional complexity overhead to existing schedulers, but it is very effective in reducing the overall power consumption of a workload without suffering any significant reduction in performance. We have shown results that indicate success in developing a model that meets our goals in this study.

This model does not always guarantee selection of the best scenario. However, based on our verification process, this model predicted correctly about 84% of our test cases.

We expect PMMP model to work for more than two applications. The only limit for the model is the machine architecture. This model is designed for those machines which have two cores in each memory domain and at least two memory domains.

It is our hope that the results presented in this project will help OS developers to reduce the energy footprint of their schedulers. In addition, we wish to encourage more research in the area of energy aware computing. Still there is more work to do in the future. Scalability of the model is still an open problem. This model works well for the architecture which has two cores in each shared memory domain. The model needs to be extended for machines with different architectures, such as those with more than two cores in the same memory domain.

Appendices

Appendix A

Power Analyzer Datalogger Model 380803

Communicating with the Power Analyzer

To request data from the power analyzer, send a character to the POWER ANALYZER through PC's RS-232 port. Any character can be sent except the reserved characters (9, 4, 2, 1, G, N, R, W, U, S, T, X, EEE). It is recommended that user send a SPACE character to the power analyzer (The ASCII code of SPACE key is 20 in hex). As having received a character from PC, power analyzer will sent out 20 bytes to PC. The 20 bytes of data are divided into 4 groups, and each group has 5 bytes. Each group represents data for Watt, A, V, and PF. Watt is for Power, A for electrical current, V for voltage, and PH for frequency.

If only one of the W, A, V, or PF is needed, user can send F1(W), F2(PF), F3(V), F4(A), or F5(Hz) for the data desired.

Decoding the 5 Data Bytes

Each byte of the 5 data bytes contain specific information and used for specific purpose. The information contained in the 5 data bytes is as following:

- 1st byte : 02 (Leading byte).
- 2nd byte : xx (Function and Range).
- 3rd byte : xx (Polarity and Digital Reading).
- 4th byte : xx (Digital Reading and Decimal Point).
- 5th byte : 03 (Ending byte).

Leading and Ending bytes

1st and 5th Byte: The first byte and the fifth byte are fixed numbers of 02 and 03. They can be used as check bytes. If they are not 02 and 03, then probably there is a hardware problem

either with PC or the power analyzer. It is also possible that the baud rates of PC and the power analyzer mismatch.

Function and Range Byte

2nd Byte: The second byte contains the information of the function and the range. To find out the corresponding function and range, refer to the following table.

HEX	Decimal	FUNCTION/RANGE
03	3	200.0V
04	4	1000V
05	5	Hz
31	49	2.000A
21	33	20.00A
C0	192	200.0W
C1	193	2000W
D0	208	PF
FF	255	HOLD

Table 6: Function code table

Polarity, Digit and Floating Point

3rd and 4th Bytes: The format of the 3rd and 4th bytes is as following.

	4th byte	3rd byte
Bit No	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0

Bit 0 Polarity (1: +,0: -) except in measuring Hz.
0: KHz, 1: MHz in frequency measurement.

Bit 1	Most Significant Digit.(0 - 1)
Bit 2 - 5	The second digit 0 - 9(0000 - 1001)
Bit 6 - 9	The third digit 0 - 9(0000 - 1001)
Bit 10 - 13	Least Significant Digit 0 - 9(0000 - 1001)
Bit 14 - 15	Decimal point position.

Bit 14	Bit 15	
0	0	None (0000)
0	1	Right most position (000.0)
1	0	Next position(00.00)
1	1	Left most position(0.000)

We have written a C code, lpower.c to access and monitor data analyzer and log the result for further research.

This program sends a query through a serial port and log all the received information iteratively until user stop it. The sample rate is 2 samples per second. The command line parameters allow the user to specify the serial port, baud rate, serial command and log file name.

A simple hash function converts the baud rate parameter in command line into an integer.

The output could be a file or console.

Data Structure

Data structure of the data analyzer data packet is:

```
struct analyzerData {
      char watt[5];
      char pf[5];
      char volt[5];
      char amp[5];
 };
```

Appendix B

MCF

429.mcf is a benchmark which is derived from MCF, a program used for single-depot vehicle scheduling in public mass transportation. The program is written in C. The benchmark version uses almost exclusively integer arithmetic.

The program is designed for the solution of single-depot vehicle scheduling (sub-) problems occurring in the planning process of public transportation companies. It considers one single depot and a homogeneous vehicle fleet. Based on a line plan and service frequencies, so-called timetabled trips with fixed departure/arrival locations and times are derived. Each of these timetabled trips has to be serviced by exactly one vehicle. The links between these trips are so-called dead-head trips. In addition, there are pull-out and pull-in trips for leaving and entering the depot.

The network simplex algorithm is a specialized version of the well-known simplex algorithm for network flow problems. The linear algebra of the general algorithm is replaced by simple network operations such as finding cycles or modifying spanning trees that can be performed very quickly. The main work of our network simplex implementation is pointer and integer arithmetic.

MILC

The MILC Code is a set of codes written in C developed by the MIMD Lattice Computation (MILC) collaboration for doing simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines. The code is used for millions of node hours at DOE and NSF supercomputer centers.

433.milc in SPEC CPU2006 uses the serial version of the su3imp program. The single processor version of this application is important and relevant, because parallel performance depends on good single processor performance.

The program generates a gauge field, and is used in lattice gauge theory applications involving dynamical quarks. Lattice gauge theory involves the study of some of the fundamental

constituents of matter, namely quarks and gluons. In this area of quantum field theory, traditional perturbative expansions are not useful. Introducing a discrete lattice of space-time points is the method of choice.

GCC

403.gcc is based on gcc Version 3.2. It generates code for an AMD Opteron processor. The benchmark runs as a compiler with many of its optimization flags enabled.

403.gcc has had its inlining heuristics altered slightly, so as to inline more code than would be typical on a Unix system in 2002. It is expected that this effect will be more typical of compiler usage in 2006. This was done so that 403.gcc would spend more time analyzing its source code inputs, and use more memory. Without this effect, 403.gcc would have done less analysis, and needed more input workloads to achieve the run times required for CPU2006.

Sphinx

Sphinx-3 is a widely known speech recognition system from Carnegie Mellon University.

CMU supplies a program known as livepretend, which decodes utterances in batch mode, but otherwise operates as if it were decoding a live human. In particular, it starts from raw audio, not from an intermediate (cepstra) format.

Although in real life IO efficiency is obviously important to any speech recognition system, for SPEC CPU purposes we wish to concentrate on the CPU-intensive portions of the task.

Gobmk

Most input is in "SmartGo Format" (.sgf), a widely used de facto standard representation of Go games. A typical test involves reading in a game to a certain point, then executing a command to analyze the position.

Povray

POV-Ray is a ray-tracer. Ray-tracing is a rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world but it does so backwards. In the real world, rays of light are emitted from a light source and illuminate objects. The light reflects off of the objects or passes through transparent objects. This reflected light hits the human eye or a camera lens. As the vast majority of rays never hit an observer, it would take forever to trace a scene. Thus, ray-tracers like POV-Ray start with their simulated camera and trace rays backwards out into the scene. The user specifies the location of the camera, light sources, and objects as well as the surface textures and their interiors.

For every pixel rays are shot from the camera into the scene to see if it intersects with any of the objects in the scene. Every time an object is hit, the color of the surface at that point is calculated. For this purpose rays are sent to each light source to determine the amount of light coming from it or if the object is in shadow. For reflective or refractive objects more rays are traced in order to determine the contribution of the reflected and refracted light to the final surface color.

Geometry objects are represented in POV-Ray by their mathematical form and a transformation. For example spheres are represented by a radius, planes by a normal vector and distance from the origin and tori by their inner and outer radii. Intersections of rays with geometry objects are computed by solving complex mathematical equations directly or by numeric approximation algorithms. All these algorithms are sensitive to floating-point accuracy.

Gamess

A wide range of quantum chemical computations are possible using GAMESS. The benchmark 416.gamess does the following computations for the reference workload:

- Self-consistent field (SCF) computation (type: Restricted Hartree-Fock) of cytosine molecule using the direct SCF method
- SCF computation (type: Restricted open-shell Hartee-Fock) of water and cu2+ using the direct SCF method
- SCF computation (type: Multi-configuration Self-consisted field) of triazolium ion using the direct SCF method

Appendix C

Defining the optimum solution based on experiments when base application co-schedule with other applications.

Applications		Energy Delay		Gain in scenario 2 over scenario 1	Optimum Senario	Degree of optimality
Based	Co-schedule	Scenario1	Scenario2			
mcf	mcf	92.44	79.22	14.30	2	Strong
	milc	102.90	107.93	-4.89	1	Weak
	gcc	13.86	12.28	11.35	2	Strong
	sphinx	128.82	100.25	22.18	2	Strong
	gobmk	10.89	10.51	3.51	2	Weak
	povray	26.29	26.95	-2.52	1	Weak
	gamess	27.96	27.48	1.69	2	Weak
milc	mcf	102.38	83.74	18.21	2	Strong
	milc	118.37	117.04	1.12	2	Weak
	gcc	15.52	12.59	18.86	2	Strong
	sphinx	136.96	102.98	24.81	2	Strong
	gobmk	11.99	10.97	8.57	2	Strong
	povray	28.00	28.16	-0.55	1	Weak
	gamess	29.14	28.92	0.73	2	Weak
gcc	mcf	83.31	77.73	6.70	2	Strong
	milc	99.83	103.52	-3.70	1	Weak
	gcc	12.31	11.86	3.61	2	Weak
	sphinx	116.71	99.18	15.02	2	Strong
	gobmk	10.33	10.14	1.83	2	Weak
	povray	25.40	24.73	2.66	2	Weak
	gamess	26.57	27.33	-2.83	1	Weak
sphinx	mcf	95.55	78.24	18.11	2	Strong
	milc	105.75	105.05	0.66	2	Weak
	gcc	14.89	12.25	17.78	2	Strong
	sphinx	136.41	100.98	25.97	2	Strong
	gobmk	11.19	9.99	10.72	2	Strong
	povray	25.51	26.68	-4.59	1	Weak
	gamess	27.42	30.11	-9.82	1	Strong

Applications		Energy Delay		Gain in scenario 2 over scenario 1	Optimum Senario	Degree of optimality
Based	Co-schedule	Scenario1	Scenario2			
gobmk	mcf	75.01	72.81	2.94	2	Weak
	milc	92.92	98.91	-6.45	1	Strong
	gcc	10.88	10.82	0.63	2	Weak
	sphinx	102.20	95.56	6.50	2	Strong
	gobmk	9.20	9.67	-5.11	1	Strong
	povray	22.71	24.97	-9.97	1	Strong
	gamess	24.36	27.39	-12.45	1	Strong
povray	mcf	65.37	70.22	-7.42	1	Strong
	milc	87.28	94.77	-8.58	1	Strong
	gcc	9.81	9.82	-0.18	1	Weak
	sphinx	89.52	94.36	-5.41	1	Strong
	gobmk	8.59	8.91	-3.76	1	Weak
	povray	24.60	27.32	-11.03	1	Strong
	gamess	23.17	23.63	-2.01	1	Weak
gamess	mcf	64.52	69.27	-7.36	1	Strong
	milc	86.55	93.85	-8.43	1	Strong
	gcc	9.58	9.65	-0.74	1	Weak
	sphinx	87.30	92.82	-6.33	1	Strong
	gobmk	8.31	8.34	-0.29	1	Weak
	povray	21.21	21.23	-0.11	1	Weak
	gamess	57.51	64.67	-12.46	1	Strong

Table 7: Optimum solution based on experiments when base application is running with other applications

Bibliography

[1] W. L .Bircher and Lizy K. John, Complete System Power Estimation: A Trickle-Down Approach Based on Performance EventsIEEE International Symposium on Performance Analysis of Systems Software, pp. 158-168, 2007.

[2] Gilberto Contreras and Margaret Martonosi, Power prediction for Intel XScale processors using performance monitoring unit events. In Proceedings of the ISLPED 2005, San Diego, CA, USA 2005.

[3] F. Bellosa and A. Weissel and M. Waitz and S. Kellne, Event driven energy accounting for dynamic thermal management. In Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP03), 2003.

[4] Jan Stoess and Christian Lang and Frank Bellosa, Energy management for hypervisor-based virtual machines. In Proceedings of the USENIX Annual Technical Conference, Berkeley, CA, USA, 2007.

[5] Kyeong-Jae Lee and Kevin Skadron, Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors. In Proceedings of the (IPDPS'05) - Workshop 11, 2005.

[6] David C. Snowdon and Stefan M. Petters and Gernot Heiser, Accurate On-line Prediction of Processor and Memory under voltage scaling. In proceedings of EMSOFT'07, Salzburg, Austria, 2007.

[7] Canturk Isci and Margaret Martonosi, Phase Characterization for Power: Evaluating Control-Flow-Based and Event-Counter-Based Techniques. HPCA-12, Princeton University, February, 2006.

[8] K.K. Pusukuri and D. Vengerov and A. Fedorova, A Methodology for Developing Simple and Robust Power Models Using Performance Monitoring Events. In Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture, pp. 54-60, June 20, 2009, Austin, Texas, USA

[9] Sergey Zhuravlev, Sergey Blagodurov and Alexandra Fedorova, Addressing Shared Resource Contention in Multicore Processors via Scheduling, in Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 129-142, ASPLOS, 2010