Power Modelling in Multicore Computing

Jason Mair

a thesis submitted for the degree of

Doctor of Philosophy
at the University of Otago, Dunedin,
New Zealand.

2015
Abstract

Power consumption has long been a concern for portable consumer electronics, but has recently become an increasing concern for larger, power-hungry systems such as servers and clusters. This concern has arisen from the associated financial cost and environmental impact, where the cost of powering and cooling a large-scale system deployment can be on the order of millions of dollars a year. Such a substantial power consumption additionally contributes significant levels of greenhouse gas emissions.

Therefore, software-based power management policies have been used to more effectively manage a system’s power consumption. However, managing power consumption requires fine-grained power values for evaluating the run-time tradeoff between power and performance. Despite hardware power meters providing a convenient and accurate source of power values, they are incapable of providing the fine-grained, per-application power measurements required in power management.

To meet this challenge, this thesis proposes a novel power modelling method called W-Classifier. In this method, a parameterised micro-benchmark is designed to reproduce a selection of representative, synthetic workloads for quantifying the relationship between key performance events and the corresponding power values. Using the micro-benchmark enables W-Classifier to be application independent, which is a novel feature of the method. To improve accuracy, W-Classifier uses run-time workload classification and derives a collection of workload-specific linear functions for power estimation, which is another novel feature for power modelling.

To further improve accuracy, W-Classifier addresses a number of common misconceptions in power modelling, which were found to impact both mod-
elling accuracy and evaluation. These misconceptions have arisen from differences in the experimental setup and configuration, such as, execution time, handling of thermal effects and performance event selection. These differences can influence the perceived modelling accuracy, resulting in potentially misleading or erroneous conclusions if sufficient care is not taken. As a result, W-Classifier has adopted a number of additional steps to ensure good modelling practices, which were not available in previous work.

In addition to improving modelling accuracy, the workload classification used in W-Classifier can be leveraged by power management policies to provide execution context to the power values. The workload context enables more informed power policies to be implemented, improving the balance between power and performance in achieving effective power management.

According to experimental results, the modelling accuracy of W-Classifier is significantly better than previous multi-variable modelling techniques due to a collection of workload-specific power models derived through run-time workload classification. Therefore, W-Classifier can accurately estimate power consumption for a broader range of application workloads, where a separate power model can be used for each distinct workload.
Acknowledgements

There have been many people who have helped me throughout my PhD, without who the completion of this thesis would not have been possible. Firstly, I would like to thank my supervisors, Zhiyi Huang, David Eyers and Haibo Zhang for the invaluable advice and guidance they have provided. I’m especially grateful to Zhiyi for his continued involvement in my work. Secondly, I’d like to thank everyone in the Systems Research Group and the Department of Computer Science for their support and collaboration. Thirdly, I’d like to thank everyone who has provided feedback and suggestions on my work and thesis. Finally, I would like to acknowledge the support of my friends and family during my dissertation, particularly their help in proof-reading this thesis.
## Contents

1 Introduction 3
   1.1 Motivation ......................................................... 3
   1.2 Motivating discussion ............................................ 4
   1.3 Terminology ..................................................... 8
   1.4 Power modelling misconceptions ................................. 9
   1.5 Contributions of this thesis .................................. 11
   1.6 Outline ......................................................... 12

2 Background 14
   2.1 Measuring power .................................................. 14
      2.1.1 System power measurements ................................. 14
      2.1.2 Integrated power metering ................................ 15
   2.2 Modelling power ................................................ 16
   2.3 Related work ................................................... 19
      2.3.1 Modelling power within a simulated environment ....... 19
      2.3.2 Performance-based power modelling ...................... 24
      2.3.3 Workload characterisation ................................. 31
   2.4 Summary ......................................................... 36

3 The Relationship between Power Use and Performance 37
   3.1 Techniques for measuring performance ...................... 37
      3.1.1 Source instrumentation .................................... 37
      3.1.2 Binary instrumentation .................................... 38
      3.1.3 Performance Monitoring Counters ......................... 39
      3.1.4 Operating system events .................................. 40
      3.1.5 Virtual machine performance ............................. 41
   3.2 Performance monitoring selection criteria .................. 42
   3.3 Interaction between power and performance .................. 45
      3.3.1 Central Processing Unit .................................. 45
      3.3.2 Memory .................................................... 49
      3.3.3 Fans ....................................................... 50
      3.3.4 Hard drive ................................................ 51
      3.3.5 Network .................................................... 51
      3.3.6 Idle system power ........................................ 52
   3.4 Method of performance event selection ...................... 53
      3.4.1 Event exploration ......................................... 54
4 Modelling Methodology 57
4.1 Power model with workload classification 57
4.2 Synthetic micro-benchmark 58
4.2.1 Reproduced workloads 58
4.2.2 Mixed workload execution 59
4.3 Deriving a performance-based power model 61
4.4 Workload characterisation 65
4.5 Methodology discussion 67
4.6 Summary 68

5 Implementation 70
5.1 Experimental setup 70
5.2 Software configuration 71
5.3 Run-time performance event monitoring 72
5.4 W-Classifier model implementation 73
5.5 Summary 77

6 Evaluation of the Power Model 78
6.1 W-Classifier evaluation 78
6.2 Considering thermal effects 82
6.3 Principles of model evaluation 84
6.3.1 Static versus dynamic power 85
6.3.2 Statistical evaluation 87
6.4 Evaluating modelling accuracy 88
6.4.1 Measuring estimation error 89
6.4.2 Identifying an acceptable modelling error 90
6.5 Power model use-cases 91
6.5.1 Improved user feedback for effective power management 91
6.5.2 Increase system deployment efficiency 92
6.5.3 Meeting the exascale challenge 93
6.6 Summary 95

7 Improvements in power modelling 97
7.1 Impact of sample rate and execution time 98
7.1.1 Various sample rates 98
7.1.2 Importance of sample rate on proposed power model 100
7.2 Thermal effects 103
7.2.1 Dynamic response to processor temperatures 103
7.2.2 Effectiveness of processor warm-up 104
7.2.3 The challenges in accounting for thermal effects 107
7.3 Correlation of memory events and power 108
7.3.1 Potential reasons for observed relationship 109
7.4 Exploring software configurations 110
## List of Tables

3.1 An example of the increasing number of hardware performance event registers per core for different processor micro-architectures. .......................... 44  
3.2 A range of processor power values taken from processor specification sheets. ........................................................................................................... 47  
3.3 Power states for two Western Digital hard drives. ................................. 52  
3.4 A selection of representative performance events available on the AMD 10h family processors (AMD, 2013). .............................................................. 55  

5.1 OpenMP NAS Parallel Benchmarks\(^1\) used to evaluate the power model. 72  
5.2 Processor performance events polled within our experiments. ................. 74  

6.1 Power and performance values for a selection of processors\(^2\) used to evaluate the power efficiency and the scale required to calculate a theoretical exaFLOP with peak performance. ......................................................... 96  

7.1 Pearson’s Correlations for different workload types with different warm-up times. ........................................................................................................... 106  
7.2 Statistics for different optimisation levels. ............................................... 112  
7.3 Bias of architecture selection. .................................................................. 116  

A.1 The 32 performance events found to have a weak correlation with power for the Mandelbrot benchmark. ......................................................... 139  
A.2 The 15 most strongly correlated performance events with the Mandelbrot benchmark, calculated using Spearman’s rank correlation. ............ 140
# List of Figures

1.1 Green500 supercomputers that use CPUs or a combination of CPUs and workload accelerators. ................................................................. 6
1.2 The historical trend for the Green500 power efficiency, with the minimum required efficiency for an exascale supercomputer. ................. 7
3.1 The raytrace benchmark executed in various core and frequency configurations. ......................................................................................... 48
4.1 Pseudocode for the micro-benchmark. .......................................................... 60
4.2 A scatter plot illustrating the correlation between the $FPU_{\text{perf}}$ intensity and power for an FPU-dominant micro-benchmark workload. ........ 64
4.3 Regression lines for each micro-benchmark ratio and $FPU_{\text{perf}}$. ............ 66
5.1 Power trends for the performance events. ....................................................... 75
5.2 Pseudocode for workload classification. ...................................................... 77
6.1 Power estimation for NPB benchmarks. .......................................................... 79
6.2 Mean error and MAE of power estimation in percentage for NPB benchmarks. ......................................................................................... 80
6.3 Measured and estimated power for NPB DC. ............................................... 82
6.4 Impact of excluding thermal effect from NPB. ............................................. 84
6.5 DC temperature comparison with a memory workload. .................................. 85
6.6 The relationship between a system’s dynamic power and the estimation error for dynamic power, on a 200W system with a total error of 5%. 86
6.7 Power versus time for the DC benchmark. .................................................... 87
7.1 Intensity of FPU operations correlated with power, measured over the entire execution of the micro-benchmark running an FPU workload. .. 99
7.2 Intensity of FPU operations correlated with power, sampled every second for $\approx$3 minute execution of the micro-benchmark running FPU workload. Vertical stripes are a result of a warm-up effect. ................................. 100
7.3 Intensity of FPU operations correlated with power, sampled every second for $\approx$25 minutes execution of the micro-benchmark running FPU workload. The linear trend is more clear though vertical stripes still exist due to a warm-up effect. ......................................................... 101
7.4 Power use of the NPB DC benchmark. ......................................................... 102
7.5 Normalised CPU temperatures and power meter readings for a long-run execution of the micro-benchmark running an FPU workload. ......... 104
7.6 Intensity of FPU correlated with power, sampled every second for the micro-benchmark running an INT workload. Sampling starts after a 60-second period of processor warm-up.

7.7 Intensity of FPU correlated with power, sampled every second for the micro-benchmark running an INT workload. Sampling starts after a 90-second period of processor warm-up.

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

7.9 Scatter plot of power and retired-\(\mu\)ops performance values for two compiled versions of the FT NPB benchmark. Each version is compiled using a different optimisation parameter.

7.10 Power versus time for two executions of the FT benchmark, that use \texttt{gcc-4.4.7} and \texttt{gcc-4.6.3} respectively.

7.11 Average power use of various parallel benchmarks, utilising all 16 processing cores, illustrating alternate dynamic power ranges.

C.1 A comparison of model accuracy for the multi-variable and W-Classifier power models.
List of Publications

Part of this dissertation has already appeared in:

Book chapters


Journal


Conference papers


Chapter 1

Introduction

This dissertation presents a performance-based power modelling methodology that utilises workload classification to derive a collection of workload-specific, linear modelling functions. The results show that this classification-based power model is able to match, and in some cases surpass, the accuracy of existing modelling methodologies.

In this chapter I outline the primary motivation for modelling power consumption, and present some common misconceptions. I then describe some common terminology, before presenting the contributions made in this dissertation, finally summarising the contents of each chapter.

1.1 Motivation

With the increasing global power use of computer technology, there is a continuing drive to develop more energy efficient and sustainable systems, with two primary motives for saving energy.

- Users are increasingly aware of the environmental impact of energy production, where energy consumption indirectly contributes to greenhouse gas emissions.

- There is an associated financial incentive for reducing power consumption. Over the lifetime of a system deployment, the total cost of ownership has begun to exceed the initial purchase price.

Unfortunately, the most commonly adopted method for reducing power consumption is a hardware-based approach, where current, inefficient hardware is replaced with newer, more power efficient alternatives. However, hardware upgrades may contribute a significant amount of electronic waste, creating an unsustainable solution. Instead,
software power management policies can be used to more effectively manage power consumption on currently deployed systems. Such policies allow for a diverse range of alternative methods for managing power consumption, that may include:

- **Power-aware task schedulers** allow applications to be scheduled based upon their power efficiency, where more efficient applications receive additional processor cores, maximising overall power utilisation.

- A **power-aware run-time system** is able to leverage hardware power saving mechanisms by dynamically adjusting component power states at run-time in response to the current utilisation level.

- **Power caps** can be enforced by software to restrict the power use of specific applications or the entire system.

- Software-based power controls can allow for increased server deployment densities in data centers, increasing system efficiency.

These power management policies require the operating system to be capable of evaluating the tradeoff between performance and power use at run-time. However, this is dependent upon the availability of fine-grained, run-time power measurements on a per application basis. Unfortunately, making fine-grained, run-time power measurements is not possible with current hardware. Existing power meters that measure power use from the mains outlet, can only provide coarse-grained power measurements for the whole system. Alternatively, embedded power sensors, like those in the Intel Sandy Bridge micro-architecture, only provide power measurements for the entire processor, not per-core (Rotem, Naveh, Ananthakrishnan, Rajwan, and Weissmann, 2012). Therefore, performance-based power models have been developed that accurately estimate system power without the use of a power meter or special hardware. While performance-based power models have previously been proposed, they are often not suited to a diverse range of application workloads, and do not provide any execution context to power estimation. This dissertation is concerned with the development of a performance-based power model that is capable of accurately estimating power for a variety of application workloads.
1.2 Motivating discussion

Power consumption has long been a primary concern for portable consumer electronics, such as laptops and smart phones. However, it has recently become an increasing concern for larger, power hungry systems such as servers, clusters and data centers. The drive towards improved energy efficiency is in part due to the increasing prevalence and scale of deployments in more businesses and industries, which have two primary motives for saving energy. First, there is an economic incentive to reducing the costs of powering and cooling a system, which can be on the order of millions of dollars for a large-scale deployment (Greenberg, Hamilton, Maltz, and Patel, 2008). Furthermore, over the lifetime of a system deployment, the total cost of ownership has begun to exceed the initial purchase price for the hardware (Koomey, Belady, Patterson, Santos, and Lange, 2009). If this trend continues, it will become a prohibitive cost for many businesses upgrading existing systems, limiting the scale of future deployments. Second, there is an increasing awareness of the environmental impact of energy production, where large energy consumers indirectly contribute significant levels of greenhouse gas emissions. For instance, the ICT industry is estimated to have carbon emissions equivalent to the entire airline industry, at 2% of the global total (Gartner, 2007). Businesses can become more environmentally friendly by reducing energy use.

The most commonly adopted method of improvement is a hardware-based approach, where current inefficient hardware is replaced with newer, more power efficient alternatives. This provides an efficiency improvement that is transparent to the end user, requiring no changes in user behaviour. Despite this distinct advantage, hardware upgrades have several key drawbacks. The main problem is the potentially significant cost associated with replacing existing hardware, particularly for any large deployment. Moreover, this is likely to contribute to a great deal of hard to handle electronic waste, making it a fairly unsustainable solution (GAP, 2007). Furthermore, it was additionally noted by Larrick and Cameron (2011) that efficiency improvements are often mischaracterised as being equivalent to a corresponding reduction in consumption. However, gains in efficiency may result in higher overall power consumption due to increased scales of deployments (Tomlinson, Silberman, and White, 2011). Such increases are further exacerbated by the tendency for data centers to substantially over-provision resources, resulting in a number of idle systems (Barroso and Holzle, 2007).

While advances in hardware have provided sustained improvements in performance and power efficiency; the rate of advancement has begun to decline since the end of de-
mand frequency scaling (Harrod, 2012). This presents new challenges in satisfying the insatiable demand for high performance computing, driven primarily by big data and scientific research. To address this challenge, the development of an exascale supercomputer, that will be capable of executing a sustained exaFLOP \((10^{18} \text{ FLOPS})\) within a power budget of only 20 megawatts, has been proposed (Harrod, 2014). This is equivalent to a system with \(1,000 \times\) the performance of current supercomputers with a similar power budget. This ambitious target is intended to drive broad advances in power and performance, eventually applying to general-purpose systems through a trickle-down effect. The Green500\(^1\) list provides a reference point for the hardware power efficiency of current supercomputers, organised as a ranked list of the Top500. The top homogeneous and heterogeneous system from each list is plotted in Figure 1.1, illustrating the rate of development for the two key system architectures. In comparison, an exascale system requires a significantly improved efficiency of 50 gigaFLOPS/watt, to execute a theoretical exaFLOP at 20MW. Placing the objective on the graph, shown in Figure 1.2, illustrates the immense challenge to be overcome, relative to the recent rate of improvement. Therefore, it is unlikely that advances in hardware power efficiency alone will be sufficient to meet the exascale challenge, necessitating the exploration of additional sources of improvement.

As an alternative to relying on gains in hardware efficiency, software can be used to reduce energy consumption. This is achieved by either adjusting the execution characteristics of a given task or leveraging any power saving mechanism of the underlying hardware. For instance, early software-based power management policies monitored system utilisation levels, making modest power savings by placing idle system components into low-power sleep states (Benini, Bogliolo, Cavallucci, and Riccó, 1998). Similar policies have been developed for data centers, where workload consolidation is used to migrate virtual machines onto the fewest nodes possible, allowing underutilised nodes to be placed into a low-power state. However, policies that merely respond to system states do not provide many opportunities for saving power. Instead more proactive, power-aware policies need to carefully evaluate the potential impact of power saving on system performance.

Therefore, advanced software-based power saving policies need to not only consider application performance, as is currently done, but additionally require detailed, application-specific power measurements. Such values would enable a power-aware policy to adequately evaluate the tradeoff between power and performance, and to

---

\(^1\)The Green500 is available at [http://www.green500.org/](http://www.green500.org/)
Figure 1.1: Green500 supercomputers that use CPUs or a combination of CPUs and workload accelerators.

determine an optimal run-time schedule. Power meters have traditionally been used to provide application-specific power measurements for single processor systems where only a single task can be executed at a time. However, modern multi-core processors present a more significant challenge as multiple applications are able to be run concurrently. Unfortunately the development of hardware meters has not kept pace with other hardware advances, which means that power meters are incapable of isolating the power of an individual application on a multi-core processor. However, power-aware scheduling policies require accurate, application-specific power estimates, necessitating the development of run-time power estimation.

It is well understood that a strong relationship exists between power and performance within a system, where modest changes in execution performance have a corresponding effect on power use. It is this very relationship that is often leveraged by many power saving policies, where processor operating frequency (performance) is lowered to save power. This is an intuitive relationship, where the power use of a system is dependent upon component utilisation levels. Therefore, a change in performance may be expected to result in a predictable change in power. For example, if the current application is compute-intensive, the processor will have a correspondingly high utilisation level, resulting in high power use. Alternatively, a memory-intensive application will perform a large number of memory accesses, leaving the processor mostly idle,
resulting in lower overall power consumption. By quantifying the strength of relationship between power and performance, an analytical model can be derived, enabling performance-based, run-time power estimation.

In addition to improving the effectiveness of existing power saving policies, power models enable the development of significantly more advanced policies that were previously not possible. One such policy is the use of software enforced power caps, where applications execute freely on a system until total power use exceeds a set threshold. Once exceeded, applications can be progressively stalled or throttled until power decreases. Such a policy can additionally be implemented at a finer-granularity, on a per-application or per-component basis. This enables rogue applications with excessive power consumption to be reigned in without impacting the execution of any other application. Alternatively, power models can be used to charge for power within a data center, similar to how other resources may currently be charged for (Narayan and Rao, 2013). Furthermore, it can be used to profile a given application, providing insights to developers for the execution power characteristics. These are but a sample of the potential use-cases for power models, which would not otherwise be possible.
1.3 Terminology

This section presents a descriptive glossary of key terms used throughout this dissertation to avoid any potential ambiguity.

Power and energy are two terms that are often used interchangeably, however each term has a specific definition. Power is a measure of the electrical energy use of a given component, for a specific period of time, measured in watts. Alternatively, 
\[ \text{energy} = \text{power} \times \text{time}, \]
taken as the integral of the power curve for a set period of time, measured in Joules. This distinction becomes important for long-run scheduling policies, where low execution power does not necessarily result in low execution energy.

The power consumption within a system can be classified as one of two key types i.e. static or dynamic power. Static power is the constant base power consumption of a given component or system that is workload independent. Whereas, dynamic power is the workload-dependent power consumption which varies with the current workload utilisation level. For power modelling, static power is commonly represented as a constant value, while dynamic power is the aspect of power consumption being modelled. While this distinction has rarely been made in existing work it can significantly impact the perceived modelling accuracy when comparing alternate modelling methodologies.

The performance events discussed throughout this dissertation specifically refer to Performance Monitoring Counter (PMC) values. PMCs are a set of special-purpose hardware registers designed to provide low-level processor performance event details during execution. Performance events are available on a per-core basis, within multi-core processors, allowing concurrently executing applications to be monitored. The low-level nature of the performance events provides for fine-grained performance details such as cache misses for the different level of the cache hierarchy, branch stalls, processor instructions completed and functional unit utilisation levels.

Dynamic Voltage and Frequency Scaling (DVFS) or frequency scaling, is a technique used to dynamically configure the speed and power consumption of a processor at run-time. This is achieved by simultaneously changing the processor’s supply voltage and switching frequency to a predetermined configuration. Specific configurations are intended to ensure stable and reliable processor operation at each alternate level.
1.4 Power modelling misconceptions

Throughout the development of the proposed modelling methodology, a thorough analysis was performed on the experimental data collected during each step. In comparing these results with the existing literature, a number of misconceptions were observed in previous modelling methodologies. These observed misconceptions were found to impact both the model implementation and evaluation. The implementation misconceptions have arisen from differences in the experimental configuration and the handling of other external factors that impact modelling accuracy.

1. *Sampling rate and execution time can be left unspecified*—In much of the previous research the sample rate for performance events and power values are seldom specified. Using coarse-grained, aggregate values will potentially obscure important execution characteristics, significantly affecting the accuracy of a power model.

2. *Thermal effects are insubstantial*—While the impact of temperature on power consumption is widely acknowledged, many existing modelling methodologies fail to mention the impact of thermal effects. Furthermore, some commonly proposed methods for mitigating thermal effects were found to be ineffective at managing processor temperatures.

3. *Memory events correlate well with power consumption*—Previous research often assumes a correlation between memory accesses and power consumption. While it is intuitive that cache misses will strongly correlate with the power of memory fetches, this was found not to be the case on the experimental system. Irrespective of the cause, such correlations should not be assumed.

In comparison, the evaluation misconceptions do not directly impact the power model, but influence the perceived modelling accuracy. A number of assumptions are often made when making direct comparisons between different modelling methodologies, resulting in potentially misleading or erroneous conclusions.

4. *Compilation configuration does not need to be reported*—The compilation configuration is often perceived to have a negligible impact, if any, on power use and is consequently given little consideration while modelling power. However, this was found to not necessarily be the case.
5. *Metrics for performance evaluation of models are comparable*—A variety of different evaluation metrics have been used by existing modelling methodologies, making direct comparisons between models infeasible. Doing so will likely result in erroneous conclusions; as some of these evaluation methods can mischaracterise model accuracy.

6. *Experimental setup can remain unspecified*—The experimental system configuration is often left unspecified, as such details are considered to be irrelevant when following assumed standard practises. However, seemingly minor details, such as the number of cores or processor operating frequency can influence model accuracy.

Therefore, the modelling methodology presented in this thesis adopts a number of additional steps that are intended to mitigate the potential impact of such misconceptions which will be discussed throughout the dissertation. A more detailed discussion of each misconception, corresponding experimental observations and relevant configuration details are presented in Chapter 7.

1.5 Contributions of this thesis

This thesis explores the development and use of a performance-based power model, that utilises workload classification to improve the models’ accuracy for a diverse range of application workloads. The following research questions were explored:

- Can workload classification be used to improve the accuracy of a power model?
- Can a single micro-benchmark be used to derive an application-independent power model?
- What metrics should be used to compare power models?

This thesis first proposes a micro-benchmark, designed to reproduce a selection of representative, synthetic workloads. The parameterised design allows for a mix of workloads to be reproduced by assigning a configurable weight to each respective workload. The use of multiple workloads is more representative of general-purpose applications than previously proposed micro-benchmarks that executed specific functions in isolation of all other workloads.
A general modelling methodology is then proposed, that leverages the micro-benchmark’s parameterised design to derive a collection of workload-specific linear functions for power estimation. A separate power model is used for each of the representative workload types additionally requiring workload classification to be performed by the model. A single data set from the micro-benchmark provides sufficient detail to be used for, performance event selection, workload classification and to derive the workload-specific power models. A general discussion is given for each step in the modelling methodology.

The thesis also presents an implementation of the proposed modelling methodology, W-Classifier, evaluated on a benchmark suite. W-Classifier is additionally compared to a standard, multi-variable linear regression model, commonly used in the existing literature. In using multiple linear functions, W-Classifier has the distinct advantage of being able to accurately estimate power for a diverse set of applications, that would otherwise not be possible. It is generally not feasible to use a single linear function to characterise the power features of various workload types in a wide range of applications.

The thesis also discusses a number of common misconceptions in the existing literature for performance-based power modelling. A number of experimental observations are presented in support of each misconception, enabling a detailed discussion of the potential causes and the corresponding impact. In addition to impacting modelling accuracy, these misconceptions have the potential to significantly misrepresent modelling results. If sufficient care is not taken while deriving or evaluating a power model, it will be misleading to make direct comparisons between different modelling methodologies, making it difficult to evaluate the current state of the art.

A number of experiments are presented throughout this dissertation, aiding the discussion within each corresponding chapter. These experiments are conducted on a single, multi-processor server described in Chapter 5, unless explicitly stated otherwise.

1.6 Outline

The remainder of the thesis is organised as follows:

Chapter 2 presents the background for this dissertation, beginning with existing techniques for measuring power and the corresponding drawbacks, motivating a discussion on modelling power. This is followed by a discussion of related work for modelling
power and the use of workload-specific power saving policies.

Chapter 3 provides an overview of alternate sources of performance events, before discussing the selection of processor performance events as the source of performance values. Component-specific interactions between power and performance are used to illustrate the primary sources of power within a standard system. The principle of domain-specific knowledge is introduced, before the discussion and evaluation of all available performance events.

Chapter 4 introduces the parameterised micro-benchmark used to generate the synthetic workloads for model training. This is followed by a discussion of the principle behind deriving workload-specific power models and classifying the corresponding workloads.

Chapter 5 presents the implementation details of the W-Classifier power model. These include the experimental hardware setup, software configuration, the run-time performance monitoring, the workload classification thresholds and performance event selection.

Chapter 6 evaluates W-Classifier on a suite of benchmarks, comparing the results against those of a traditional multi-variable regression model. This is followed by a discussion of the misconception of making direct comparisons between different modelling methodologies, as is commonly done in the existing literature. Finally, a thorough discussion of different use-cases is given.

Chapter 7 discusses many of the challenges that are to be overcome in deriving an accurate power estimation model. Each subsequent challenge is demonstrated with experimental observations followed by a general discussion of potential causes and impacts. Many of the observed misconceptions are commonly made in the existing work, which can be partially mitigated by adopting good modelling practises.

Chapter 8 concludes the thesis and highlights some potential future work.
Chapter 2

Background

In this chapter I begin by discussing the most prevalent power metering methods and the key limitations preventing their use in power-aware software management policies. I then present additional details on the principles of modelling power, and the respective advantages over power meters. The remainder of the chapter discusses related work for power modelling and power-related workload classification.

2.1 Measuring power

Hardware power meters are capable of directly measuring power consumption at two key granularities, system-level or integrated metering. System-level power meters are placed between a given system and the wall outlet. Integrated power meters are placed inside a system, enabling component-specific power measurements. Despite hardware power meters providing the most accurate source for system power measurements, they are incapable of providing the fine-grained, application-specific measurements required by many advanced power management policies. This section presents an overview of these two primary classes of power meter, discussing their corresponding limitations.

2.1.1 System power measurements

The most commonly adopted method for monitoring system power is with an external power meter. Commodity wall connected power meters, such as the Watts Up? Pro,\(^1\) iSocket\(^2\) (InSnergy Socket), and OmegaWatt\(^3\) are commonly used for standalone

\(^1\)http://www.wattsupmeters.com
\(^2\)http://web.iii.org.tw
\(^3\)http://www.omegawatt.fr/
systems. Moreover, similar metering functionality is increasingly being integrated into standard power supply units (Kamil, Shalf, and Strohmaier, 2008). Deploying this capability within commodity servers enables additional remote administration solutions (McGary and Pan, 2006). Alternatively, large-scale deployments can use intelligent power distribution units (PDUs), which are standard PDUs with the additional capability of separately metering each compute node. This provides a convenient solution for large-scale deployments, where it is impractical to install metering equipment for individual nodes (Kamil et al., 2008). Power meters have the significant advantage of easy deployment by not requiring any changes to existing compute equipment. However, large-scale deployments will incur a substantial financial cost.

While hardware power meters are capable of providing accurate, system-level power metering, they have two drawbacks that make their use unsuitable for many advanced power-management policies. First, power meter sample rates, typically once a second, have been found to be insufficient for detailed, fine-grained power analysis of applications, where key execution characteristics may be obscured (Diouri, Dolz, Glück, Lefèvre, Alonso, Catalán, Mayo, Quintana-Ortí, et al., 2013). Second, power meters only return coarse-grained measurements for the entire system, resulting in a semantic gap between the observed power and corresponding task executions (Hu, Jiménez, and Kremer, 2005). Therefore, it is impractical to readily determine the relative power use of specific applications or system components. Without this capability, power management policies can only be practically enacted upon the entire system, rather than individual applications. The problem becomes more significant as the core count of processors increases, resulting in an increasing number of concurrently executing applications.

2.1.2 Integrated power metering

A solution to the problem of coarse-grained power measurements is to integrate low-level power sensors within a system, enabling the metering of individual components. Such fine-grained measurements are possible by independently monitoring the DC power rails for each system component. Two of the most commonly used measurement techniques are shunt resistors and clamp meters. Shunt resistors are placed inline with each power rail, measuring the voltage drop across the resistor, enabling the current and power draw to be calculated. Clamp meters provide a more convenient solution, as they are placed around each power rail, using Hall effect sensors to measure power. While such power meters are used during system development and testing, they are
rarely included in commercial products, which may be attributed to cost (McCullough, Agarwal, Chandrashekar, Kuppuswamy, Snoeren, and Gupta, 2011).

However, recent processor microarchitectures have begun to embed power sensors, making power values available through specific registers. This functionality has been provided in the RAPL (Running Average Power Limit) by Intel (2014), AMD 15h family processors (AMD, 2014) and the NVIDIA Management Library NVIDIA (2012). Unfortunately, the current measurement granularity is limited to a few select configurations. For instance, the Sandy Bridge micro-architecture supports three alternate measurements: all processor cores, the integrated graphics processor, or the entire package, including both the graphics and processing cores (Rotem et al., 2012). Additional system components can be metered using a third party solution such as PowerMon2,\(^4\) which provide independent inline metering of the 3.3V, 5V and 12V power rails. More specific metering is provided by the NI\(^5\) and DCM\(^6\) meters, which only monitor the processor’s 12V power line. This highlights one of the main limitations of internal meters, where it will often only be practical to monitor a subset of system components, as not all solutions provide enough sensors to allow full system metering.

Furthermore, integrated power sensors make the significant assumption that all component rails will be exposed, i.e. power lines are not embedded in the motherboard, making them unusable. Even if exposed, power lines may be grouped, resulting in errors over 50% if lines are not initially separated when using a clamp meter Kamil et al. (2008). Installing additional equipment within a node will disrupt the airflow, raising system temperatures, resulting in higher power consumption (Milenkovic, Milenkovic, Jovanov, Hite, and Raskovic, 2005). Despite the incurred costs and deployment inconvenience, integrated power meters are incapable of isolating application-specific power measurements for shared resources. System components, such as multi-core processors, are powered by a single power rail, whereby core-specific power use will be indistinguishable.

\section{2.2 Modelling power}

Unfortunately, hardware power meters are incapable of providing fine-grained, application-specific power measurements on modern multi-core processors. Therefore an alternative solution is required. For this, performance-based, power estimation models are

\(^4\)http://github.com/beppodb/PowerMon
\(^5\)http://www.ni.com
\(^6\)A non-commercial power meter by Universitat Jaume I
a commonly proposed method. Power models work by quantifying the relationship between observed changes in system utilisation (performance), with the corresponding change in power, resulting in an analytical model capable of estimating run-time power consumption. Standard power models use a single, generalised function to model power for a diverse set of workload types in a wide range of applications. However it can be infeasible for a single set of performance events to characterise the execution power of significantly varied workload types with potentially orthogonal performance events. For example, cache-misses consume power that is related to the memory accesses within a given workload, but are less related to the floating point operations of a compute-intensive task. Instead, processor-related performance events, such as retired instructions, are strongly related to compute-intensive workloads.

In contrast, this thesis proposes a performance-based power model that utilises workload classification to derive a collection of workload-specific modelling functions. This allows the power model to tailor the estimation function used for a given workload, mitigating any potential workload bias by using the most appropriate model in every instance. In addition to improving modelling accuracy, workload classification can be utilised to achieve the broader objective of closing the semantic gap that exists between a given power value and the corresponding execution workload. That is, a single power value could be a result of two different workloads, where each respective cause requires a different power management response. For instance, a low processor power level could be a result of executing a memory workload, causing low processor utilisation due to frequent memory fetches. Alternatively, the same processor power level could be a result of a CPU-intensive workload that is currently experiencing a period of thread synchronisation, resulting in a temporarily lower processor utilisation level. For the memory-bound workload, power can be saved by lowering the processor’s operating frequency, without adversely impacting the execution performance. However, enacting the same policy for the CPU workload will have a detrimental impact on execution performance. Without the context for the current workload, it will be difficult for a power-aware policy to make effective scheduling decisions.

Power models provide a number of significant advantages over power meters which are as follows:

**Fine-grained power values** — Unlike power meters, power models are capable of providing fine-grained application-specific power estimates within systems that share resources. Power models can be derived at any granularity for which performance measurements are available. For instance, by using the hardware per-
formance events within processor cores, a power model can estimate power at a sub-component, per-core granularity, allowing for individual application threads to be modelled if desired. This fine-granularity of monitoring significantly surpasses that of hardware power metering techniques.

**Responsiveness** — Power meters are unresponsive to run-time changes in workload execution, returning aggregate power measurements for a fixed time period, typically once a second, irrespective of any changes in application execution within the corresponding interval. In contrast, the use of fine-grained performance events creates the potential for the development of a power model capable of attributing power consumption to specific applications within a task switching environment. That is, performance events can be assigned to a given application, where the assignment can change in response to context switches. Therefore, a closer association will be possible between applications and power, regardless of the broader monitoring interval.

**Effective deployment** — While power meters are expensive and inconvenient to deploy, particularly for large-scale deployments, software-based power models provide a cost effective solution that can readily be scaled across an entire data center. After initial training, power models essentially have no additional requirements for existing system deployments, without the requirement of additional hardware upgrades or purchases.

**Execution context of power** — The performance events used in modelling power consumption can additionally be leveraged to provide execution context, through workload classification, to power values. While this may be dismissed as irrelevant, power-aware policies are a primary motive for the development of application-specific power modelling.

**Generality and extendibility** — The generality of power models ensures they remain microarchitecture independent, where the use of specific metering functionality, such as Intel or NVIDIA’s integrated metering, would impose a significant restriction on the usability of the power model. However, while some commonly used performance events have microarchitecture-specific implementations, they are sufficiently abstracted from such details, and are broadly supported on other system architectures. Furthermore, this generality allows the model to adapt to changes in system architecture, like new system components. This provides for a
robust power model, that remains relevant within a changing environment, where new technologies may be deployed in the future.

All of the discussed advantages collectively provide for an accurate and effective solution to providing fine-grained, application-specific power accounting and evaluation. These models subsequently facilitate the development of advanced power-aware task schedulers in operating systems, novel evaluation metrics, and an unprecedented level of user feedback and control, all of which would not be readily achievable with traditional hardware-based power metering.

2.3 Related work

This section presents previous work related to the proposed workload-aware power model. The discussion begins with the early use of power models in simulation environments, which led to the development of run-time power models, derived from performance events. A summary of power policies that utilise workload classification is presented, before discussing recent work that utilises the principles of workload classification to improve modelling accuracy.

2.3.1 Modelling power within a simulated environment

Simulation environments provided some of the earliest methods of performing application-specific power modelling through the use of extensions that enabled low-level power accounting. This section summarises previous work for cycle- and instruction-level power modelling methods.

Cycle-accurate simulator

Early power models were implemented within microarchitectural simulators, through the inclusion of detailed, cycle-level power accounting methods. Functional unit power is typically modelled using either; the hardware data sheet specifications, or by modelling the physical circuit-level power dissipation using the standard CMOS equation, \( \text{power} = \alpha cv^2f \), where \( \alpha \) is the activity factor, \( c \) is circuit capacitance, \( v \) is the processor voltage and \( f \) is the switching frequency. Wattch was one such early cycle-accurate power simulator proposed by Brooks, Tiwari, and Martonosi (2000). Wattch was built as an extension on the SimpleScalar simulator, modelling power by calculating the circuit capacitance of the microarchitectural units. It provided a parameterised design,
allowing fine-grained, microarchitectural details to be specified including, the num-
ber of cache entries, their corresponding width, and the number of read/write ports.
This customisability enabled Wattch to be used for evaluating the power efficiency of
both microarchitectural changes in processor layout and the corresponding impacts on
application execution.

A similar approach was taken in Simplepower (Ye, Vijaykrishnan, Kandemir, and Ir-
win, 2000; Vijaykrishnan, Kandemir, Irwin, Kim, and Ye, 2000), which also extends the
SimpleScalar simulation environment to provide a means of estimating power. Power
is modelled using the capacitance of each functional unit, however unlike Wattch,
SimplePower additionally classifies functional units as either bit-independent or bit-
dependent. That is, the power of bit-dependent functional units is determined by
current state and additionally, the transition from the previous state. Alternatively,
bit-independent power is solely determined by the current state. To achieve this, Sim-
plePower uses a series of lookup tables for each bit-dependent functional unit, to deter-
mine the corresponding power. While this improved modelling accuracy, simulations
incurred significant additional overhead and complexity, due to the prohibitively large
lookup table. This necessitated the use of additional analytical models and clustering
techniques intended to reduce the search space to improve execution time.

The cycle accurate processor power models were later extended to model full system
power by incorporating analytical power models for the remaining system components,
e.g. memory and hard disk. Gurumurthi, Sivasubramaniam, Irwin, Vijaykrishnan,
and Kandemir (2002) proposed the SoftWatt power simulator, built on top of the
SimOS simulation environment processor power consumption was based upon Wattch,
while the other system component’s power consumption was determined with analytical
power models that were derived offline from simulation log files. This methodology
enabled the isolation of both component- and instruction-specific power. Furthermore,
this facilitated a number of observations regarding the potentially significant power
consumption of the broader system components that were previously neglected in many
power simulation environments.

The addition of all system components enabled the simulation environments to be
extended beyond user-space applications through the inclusion of kernel-level instru-
m entation, requiring simulation of the entire operating system. With this previous
focus on performing user-level simulations, the SimWattch model, proposed by Chen,
Dubois, and Stenstrom (2003), took the alternative approach of combining a full system
simulator with a user-level power and performance simulator. This enabled compar-
isons to be made between user-space and kernel-space execution characteristics. It was observed that scientific benchmarks spent little time executing system calls, while commercial database benchmarks spent 50.9% of their execution time in kernel-space. This highlighted the importance of modelling the power for the entire application execution, including kernel-space, as some applications can spend a substantial amount of execution time performing system instructions. Without the inclusion of kernel-space, the resulting power estimates were found to potentially underestimate total energy consumption by more than 100%.

The use-cases of these simulation environments were later extended with the capability of performing detailed, post processing analysis of application power consumption. For instance, Shafi, Bohrer, Phelan, Rusu, and Peterson (2003) presented an event-based power simulator that logged per-cycle events from the simulation environment for determining power and performance for later analysis. Built as an extension to the PowerPC simulator Mambo, it provided the unique capability of isolating the power dissipation of key performance events for visualisation during post-processing. This functionality was intended to aid software developers by associating simulation results with the corresponding source code, enabling a novel form of power analysis.

**Instruction level simulation**

Instruction level power simulations are intended to provide a less fine-grained approach by modelling the power consumption of execution instructions. After the power consumption of each instruction has been determined, they are placed in a lookup table, which can readily be accessed within the simulation environment. Thereafter, power is calculated during execution by recording a count for each instruction type, which is later used to sum the occurrences of each instruction to determine full system power.

An early methodology for instruction level power modelling was proposed by Tiwari, Malik, and Wolfe (1994), which used synthetic micro-benchmarks to determine the power consumption of each instruction for the given microarchitecture. The power for each instruction was found by independently executing each instruction in isolation, within an infinite loop while monitoring processor power with an ammeter. It was supposed that these values could be summed to derive full processor power, however the power values for each instruction were found to be slightly higher during typical application execution. This discrepancy was attributed to the state changes incurring additional power, where the initial data collection did not contain such changes. This variation was rectified by empirically determining a small constant value to be added
to each instruction to represent the circuit state overhead.

However, these observations are contrary to those made by Sinha and Chandrakasan (2001) while deriving the instruction-based model, JouleTrack, where the variations in instruction current draw during benchmark execution were not as significant as had previously been indicated. While collecting the power of each instruction in isolation, the maximum variation in average current consumption was found to be 0.072A, 38%. However, the variation during benchmark execution was significantly lower at 8%. This lack of variation during benchmark execution allowed JouleTrack to adopt a novel method for constructing the lookup table, where instructions were clustered instead of having a separate entry for every instruction. This resulted in four groups of instructions with a similar power consumption, substantially simplifying the modelling procedure and reducing potential run-time overhead.

Haid, Kaefer, Steger, and Weiss (2003) proposed an alternative method for improving modelling performance, where a co-processor is used to log instruction counts at run-time in a set of special-purpose hardware registers. The register values are then able to be used by the instruction level power model, similar to those previously proposed where instruction power is initially derived offline through empirical observations.

Instruction level power models have been used for similar purposes to those previously discussed for the cycle-level power models, such as modelling an application’s full system power use. EMSIM is an instruction level, full system power simulator for embedded systems, capable of performing a full execution traces (Tan, Raghunathan, and Jha, 2002). The EMSIM framework provides energy values for each function within an application during execution by recording a separate power balance sheet for each system task, and in turn each function within the corresponding task. By providing a detailed power breakdown for the operating system and application functions, developers can gain significant insights into how power is consumed, allowing applications to be optimised for power in a similar manner to how performance may be optimised.

Problems with using simulation

All of the discussed simulation-based modelling methodologies suffer from several key drawbacks, limiting their general adoption. For instance, cycle accurate power models rely on the availability of detailed microarchitecture specifications of functional unit capacitance for modelling power, which, as noted by Qu, Kawabe, Usami, and Potkonjak (2000), manufacturers may be reluctant to provide. An alternative proposal is to incorporate such microarchitectural details into the binary distributions of library rou-
tines, allowing such details to be used without having been explicitly specified. While this would enable the monitoring of library routines within the simulation environment, it more significantly neglects user-defined routines, making for a generally impractical solution.

A primary limitation of instruction-based power modelling is the inability to model or validate the possible power effects of various instruction execution combinations. For example, the IA-32 instruction set contains 331 instructions, giving a total of 109,561 inter-instruction combinations when only two consecutive instructions are considered (Wang, Chen, and Shi, 2011). Furthermore, the scale of the problem increases exponentially when considering more than two instructions or newer processor microarchitectures with larger instructions sets for supporting new functional units, registers and pipeline sizes.

While these drawbacks are specific to the respective approaches, all of the simulation-based power modelling methodologies share some more general limitations which may prove to be more significant. The primary drawback for all simulations is the required execution time, which is commonly much slower than a native execution. Consequently, this necessitates a tradeoff between performance, model detail and the resulting accuracy. A commonly adopted compromise is to limit the scope of a simulation by performing a partial simulation, intended to provide a representative sample of the entire application’s execution characteristics. However, sufficient care must be taken when selecting the code region to be simulated, where failing to do so will result in non-representative results (Sherwood and Calder, 1999). This presents an additional challenge to be overcome before these power models can be used effectively. The intended solution will depend on the type of benchmark, which will have one of two dominant execution patterns, either convergence towards steady state execution or cyclical execution phases caused by the dominance of loop execution. It is with the later workload characterisation that care must be taken, where only simulating part of an execution phase can result in misleading conclusions. Furthermore, it was noted that the start of execution is often different to the remaining execution pattern, where the tendency to simulate applications from the beginning, and not to completion, will additionally lead to erroneous conclusions.

An alternative method for determining representative execution phases within an application is with basic block analysis, which is used to find small program sections which have an execution workload proportionally equivalent to that of the entire application (Sherwood, Perelman, and Calder, 2001). This was achieved by collecting
a number of basic block vectors for execution segments, which are compared to the entire execution in turn until a basic block is found that has a small difference in the proportion of relative execution time spent in each of the basic blocks. By isolating these small execution phases, they can be used with simulations with confidence that they are representative of the entire execution. However, performing such a detailed analysis of every application before execution is impractical. For a power model to be practical, it has to be capable of accurately estimating power for previously unseen applications, which is not be achievable in this case.

Furthermore, simulation environments commonly only perform deterministic executions, where the same output is consistently produced for a given input application, irrespective of the number of iterations performed. In comparison, a native execution will experience some execution variability in: the execution path, execution time of workload segments and the corresponding power consumption. While these variations may be minimal, they have the potential to introduce important cumulative effects, resulting in erroneous conclusions (Alameldeen and Wood, 2003). Moreover, execution variability will become increasingly important with the growing prevalence of multi-core processors and parallel applications, which will need to be factored into subsequent power models. However, incorporating artificial variability into a simulation environment will require multiple execution iterations, exacerbating existing performance issues.

Given these drawbacks, simulation-based modelling methodologies are not capable of meeting all of the desired properties of a practical, run-time power model which is to be used within a task scheduler. Therefore, we need to look at alternative modelling methodologies.

2.3.2 Performance-based power modelling

Run-time performance events are commonly used to derive an analytical power model, quantifying the relationship between a change in performance and the corresponding change in power use. These models are based upon the relationship between system component utilisation and power consumption. The proposed modelling methodologies vary in the source of performance events and the corresponding modelling granularity, making a tradeoff between estimation accuracy and run-time performance.
High-level utilisation-based models

In deriving a performance-based power estimation model, a series of tradeoffs are required between: model accuracy, model complexity, training complexity, run-time overhead, portability, and generality. This extensive set of tradeoffs necessitates careful consideration during initial model development and construction. Consequently, some simple, high-level utilisation-based power models have been proposed that seek to leverage the observed relationship between high-level utilisation metrics, such as IPC (Instructions per Cycle), and power to construct a general model. For instance, Valluri and John (2001) conducted a set of experiments using the Wattch simulation environment to evaluate the relationship between compiler optimisations and the resulting power consumption. From these experiments, it was observed that power consumption has a direct relationship with IPC, where optimisation levels that resulted in higher average IPC, resulted in higher power use.

The strength of this observed relationship was further verified by Li and John (2003) through a thorough evaluation of operating system execution routines. The variation in IPC and power was able to be quantified using a linear regression model, enabling a linear IPC-based power model to be derived for each routine. Subsequent evaluation found the simple linear model provided more accurate power estimates than the more complex, performance-based models that had previously been proposed.

Aside from IPC, alternative metrics can be used to represent utilisation, such as $\mu$ops (micro-operations). Bircher, Valluri, Law, and John (2005) found the use of $\mu$ops improved results for the Pentium 4 over the more common IPC metric. Specifically, fetched $\mu$ops are used, which includes both retired and cancelled $\mu$ops. This is an important distinction as $\mu$ops that fail to complete execution still spend time within the processor’s pipeline, consuming power. Furthermore, the use of $\mu$ops provides slightly finer-grained performance values than instructions, as different instructions will consist of different numbers of $\mu$ops, which may explain part of the improved accuracy over IPC for the observed experiments.

The principle of modelling power with utilisation metrics can be expanded to additional system components in order to derive a full system power model, such as MANTIS (Economou, Rivoire, Kozyrakis, and Ranganathan, 2006). MANTIS uses a combination of component-specific performance events and operating system event statistics to derive a single linear model for the entire system that was capable of modelling power for: the processor, memory, hard disk and network I/O. The utilisation values for each respective component are: CPU utilisation, off-chip memory access...
count, hard disk I/O rate and network I/O rate. Such an approach strives to bridge the gap between detailed performance event models and high-level utilisation models through the selective adoption of techniques from the two alternate methods.

The SPAN power model, proposed by Wang et al. (2011), derives a run-time power estimation model using the fewest performance events possible in order to construct a general and portable power model. The most novel aspect of this work is the inclusion of processor operating frequency in the estimation model, where frequency was chosen as it is a rarely considered factor that has a direct impact on power. The selected performance event was IPC, which was acknowledged to not always be the most strongly correlated performance event with power, but that this was an acceptable tradeoff for model simplicity and portability. An extensive set of micro-benchmarks, with IPC values extended to uncommon extremes, were used to collect the training data, from which a linear regression model was derived, capable of providing accurate power estimates.

**Processor-based performance event power modelling**

A significantly more detailed power model can be derived from the fine-grained, processor performance events, enabling more accurate power modelling. This is primarily achieved by either decomposing the processor into its key functional units, which are explicitly modelled by their respective performance events or by determining the events most strongly correlated to various types of execution workloads. While such fine-grained performance events may improve modelling accuracy, explicitly depending on particular performance events will result in a microarchitecture-specific power model, restricting portability and widespread adoption.

Work by Bellosa (2000) provided some of the earliest observations for the strong relationship between the low-level processor performance events and power for a selection of alternate workload types. A collection of synthetic micro-benchmarks were used to artificially create four representative workloads: integer operations, floating-point, cache-misses and memory accesses. For each of these workloads, a single performance event was found to provide a strong linear correlation between the respective utilisation levels and power consumption. Based upon these observations, the Joule Watcher power model was derived using the four performance events: retired-μops, FPU μops, l2-cache misses, memory (load/store operations). In subsequent work, Joule Watcher was used to implement a power-aware run-time policy for dynamically throttling application workloads, reducing overall power consumption without detrimentally impacting
execution performance (Bellosa, 2001).

While the approach taken by Bellosa was to determine the performance events that strongly correlate to a limited set of common software workloads, the other dominant approach adopted in performance-based power modelling is to determine the performance events that strongly correlate with the underlying hardware. That is, the performance events are used to isolate the utilisation level of specific sub-components and functional units, thereby enabling the corresponding unit-specific power to be modelled. These two methods largely differ in the granularity of the proposed models, with the latter providing a finer-grained approach. However, both are guided by the same general principles, where workload-based power models are somewhat guided by microarchitectural knowledge when performing workload selection. While indirect, each workload corresponds to the key processor functional units generally modelled in the alternative approach.

Joseph and Martonosi (2001) were early proponents of modelling power by decomposing the processor into key sub-components, deriving the Castle power model. However, it was noted during model development that performance events were not provided for all of the relevant functional units, due to the different requirements of performance analysis and modelling power. Therefore, Castle additionally derived heuristic performance events to model those not explicitly provided. That is, a second order performance event was found, allowing the desired event to be calculated using a pre-defined, microarchitecture-specific heuristic equation. For instance, the \textit{fetch\_access} event was used as a proxy for \textit{icache}, which is not directly measurable. Furthermore, the number of required performance events exceeds the limited performance register count of each processor, preventing the simultaneous monitoring of all performance events. This is a direct consequence of the fine-grained modelling approach taken. Therefore, the authors propose the use of event multiplexing, where pairs of events are monitored for a short time slice before being swapped with an alternative set of performance events. Unfortunately, such an approach relies on stable execution in order to correlate the different sets of events. After resolving these issues, the resulting model was derived and evaluated on the Wattch simulator for an Alpha 21264 processor and on native hardware with a Pentium Pro. In later work, Castle was used to perform a detailed analysis of the tradeoffs between power and performance for the Pentium Pro processor, providing some preliminary insights for data cache and branch prediction (Joseph, Brooks, and Martonosi, 2001). This demonstrates one of the potential use-cases of similar power models.
The same principle of processor decomposition was adopted by Isci and Martonosi (2003) to derive a power model for the more advanced Pentium 4 processor. Each of the 22 sub-components was identified from details of the processor die layout, as opposed to the circuit-level details used by Castle. A further point of difference is in the use of a more modern processor with dynamic power saving features, such as aggressive clock gating. This presented additional challenges in modelling power as the power of each sub-component was observed to have significantly larger power variations for alternate utilisation levels. Therefore, sub-component power was calculated using weighted access rates as a proportion of the maximum power consumption. For some components, this resulted in a linear model, while some other components were found to have a more complicated piece-wise relationship. In these cases, power was found to rapidly increase from low access rates, with the rate of increase diminishing as the maximum is reached. Such a function relies on obtaining the maximum power of each component, which was initially set as the proportion of the processor’s maximum power corresponding to the relative die area occupied by the given component. These approximate values were later fine-tuned through the use of specific micro-benchmarks designed to utilise individual components. However, a key limitation of this modelling methodology is the reliance on the availability of a large number of performance event registers to enables all sub-components to be monitored simultaneously. Despite the Pentium 4 having 18 performance event registers, multiplexing was required. For other microarchitectures which do not have such a large number of registers, a prohibitive level of event multiplexing will be required.

Most processors only support a small set of performance event registers, such as the Intel PXA255 which only has two event registers. Such limitations on the simultaneous collection of performance events requires a substantially more general modelling methodology. One such methodology was proposed by Contreras and Martonosi (2005) for the PXA255 processor, where only 5 performance events are required. Instead of modelling all of the processor functional units, a set of representative performance events are used to model the corresponding functional unit power. Using a restricted set of performance events will generally reduce the training time, however, in this instance, event multiplexing is not used, requiring multiple iterations of each benchmark. Furthermore, the training procedure is repeated for each of the available processor operating frequencies. A separate, parameterised linear model is derived for each processor frequency, which is unique to the proposed modelling methodology as much of the previous work did not explicitly consider the impact of changes in frequency.
While previous work had used the processor’s performance events to specifically model the processor power, Bircher and John (2007) used the performance events to derive a complete system power model, which included the processor, memory, I/O, chipset and disk. Unlike the previous approaches to modelling system component power, which typically used operating system component statistics, the proposed power model only used processor performance events. A full system power model is made possible by determining the ‘trickle down’ effects from processor events to each component. For instance, TLB misses cause a data page to be fetched, which, due to their size, will often result in hard drive accesses. The power model was derived from performance event observations for a set of benchmarks, and synthetic micro-benchmarks. Domain-specific knowledge was used to limit the set of evaluated performance events, reducing the required search space and corresponding training time. Following a thorough evaluation, a separate linear or quadratic equation was used to model each system component, which were collectively used to model full system power use.

The power model proposed by Singh, Bhadauria, and McKee (2009) uses domain-specific knowledge to derive a per-core power model for the multi-core AMD Phenom processor. The microarchitectural knowledge is used to determine four performance categories for which power will be modelled: FP Units, Memory, Stalls, and Instructions Retired. Only four categories were chosen as this is the number of physical performance event registers available on the experimental system, thereby allowing real-time performance monitoring without the use of event multiplexing. The model makes a novel separation between event selection and model training, where initial event selection is performed using a suite of benchmarks to find the most strongly correlated performance event within each of the four categories. Once selected, a synthetic micro-benchmark is designed for each performance event, which is intended to create a large variety of utilisation levels, from which the power model will be derived. It was observed that the power/performance characteristics varied markedly between low and high levels of utilisation. Therefore, a piece-wise linear model was used to derive a separate function for both low and high utilisation levels, achieving a better overall fit with the data. The resulting per-core power model was used to implement software power caps, enabling application executions to be stalled whenever the cap was exceeded, where execution would be resumed once sufficient power headroom was available.

A significantly more exhaustive approach was taken by (Bertran, Gonzalez, Martorell, Navarro, and Ayguade, 2010), where a set of 97 micro-benchmarks was used to
decompose the processor into the smallest power components possible. During performance event selection, each micro-benchmark applies various utilisation levels to a specific processor component. Despite taking such a fine-grained approach, not all processor components are capable of being directly measurable with performance events. Furthermore, some of the processor components are tightly coupled together, sharing a single performance event, making them inseparable. Consequently, many of the collected events were grouped into clusters, resulting in only seven components being monitored. The final model uses multiple linear regression to quantify the marginal effect on power of each microarchitectural component.

While these processor performance event power models are capable of accurately modelling processor power for a variety of different evaluation environments, many of the proposed modelling methodologies enforce a strict microarchitectural dependence that imposes practical limitations on the usability of such models. The primary source of microarchitecture dependence arises from the use of micro-benchmarks tailored to a given microarchitecture’s instruction set or the provided functional units. Therefore, additional hardware abstraction will provide for a more widely deployable modelling solution.

**System-based performance events**

System performance event statistics can be used to derive a general, microarchitecture independent power model. Despite the accuracy provided by processor-based performance models, they are impractical to deploy on different architectures, given the model’s strict dependence on microarchitectural support for key performance events. Alternatively, a system-based power model is only dependent upon the underlying operating system, and the provided operating statistics. Some of the previously discussed power models have leveraged the generality of these performance events for modelling non-processor system components. Moreover, these models typically used a few select performance events, and did not perform an exhaustive analysis of system event usability. A comprehensive approach was adopted in the modelling methodology proposed by Da Costa and Hlavacs (2010), which used both processor performance events and system-level performance statistics, giving a total of 165 different events. This allowed for the development of a full system power model, capable of accurately modelling power for the processor, memory, hard disk and network card. The model was derived using a set of configurable, synthetic micro-benchmarks, designed to explore a wide spectrum of component-specific utilisation levels, resulting in a multivariate regression
Furthermore, modelling power within a virtualised environment presents an additional set of challenges, where hardware is not directly accessible. Many of the previously proposed power models rely on the use of low-level performance events, monitored in hardware, that are not available within a virtualised environment. Therefore, higher-level system performance events provide a more convenient source of run-time performance values. Furthermore, virtual machines are commonly deployed in large data centers which may consist of a heterogeneous system deployment, necessitating a microarchitecture independent approach. The iMeter power model, proposed by Yang, Zhao, Luan, Qian, Xie, Mars, and Tang (2013), used system-level performance events to enable power modelling within a virtualised environment, without requiring direct access to hardware performance event registers. Principal component analysis was used to select the most strongly correlated performance events from the multitude of alternatives. Support vector regression is then applied to the chosen events to derive a single, full system power model.

While system-level performance events allow for a portable solution capable of accurately modelling power for additional system components, such as network cards, this functionality comes at the cost of fine-grained, thread-specific power. This will become an increasingly significant limitation as parallel applications become more widely supported, where the use of system events will inhibit many thread-specific management policies.

### 2.3.3 Workload characterisation

A strong relationship exists between various execution workloads, the utilisation of specific functional units and power consumption. Knowing the current execution workload allows additional information to be inferred regarding the utilisation of system components and power, which can additionally be leveraged by power models or policies for saving power.

#### Power saving policies

Many power saving policies have been proposed that utilise knowledge of the current workload to inform schedule selection. Currently, the most prevalent policies are used to dynamically adjust the processor’s operating frequency at run-time in response to specific execution workloads. For example, Weissel and Bellosa (2002) used a suite
of synthetic micro-benchmark workloads to evaluate the relationship between various performance events and power for alternate operating frequencies. It was observed that lowering the operating frequency for memory-intensive workloads did not adversely impact execution performance, with the lower frequency providing additional power savings. Alternatively, the performance of compute-intensive workloads was significantly degraded by lower operating frequencies, resulting in higher overall energy consumption. From these observations, a policy for classifying workload characteristics was proposed, requiring only two performance events, memory requests per clock cycle and instructions per clock cycle. The classification policy enabled additional guidance to be provided to the operating system when determining the operating frequency to be selected for each processor during execution, helping to save power without sacrificing performance.

An alternative dynamic frequency scaling policy was proposed by Freeh and Lowenthal (2005), which uses a rudimentary workload characterisation method and a user-defined evaluation function. The workload characterisation is based upon the results of an initial, offline trace of each application, taken while executing at the highest operating frequency. Applications are partitioned into blocks, whose boundaries are defined by MPI function calls, or a marked change in the memory behaviour. Adjacent blocks are combined into phases according to the similarity of memory use. The resulting phases are then ranked according to the likelihood of providing power savings in accordance with the evaluation function. Finally, the operating frequency of each phase is determined at run-time, where each execution phase begins at the highest frequency before being progressively lowered until the power and performance violates the evaluation function. The previous valid frequency is then set for the current phase before execution continues to the next phase, where the procedure is repeated. While this simple procedure was able to provide good results, it is substantially limited by the requirement of a full execution trace.

Ge, Feng, Feng, and Cameron (2007) proposed the CPU MISER run-time policy for determining execution workload characteristics and the corresponding operating frequency. An application is partitioned into Workload phases, pre-defined as a set execution period, each of which are characterised by their degree of CPU-boundedness. That is, each of the periodic samples is defined as an index of compute-intensiveness $0 < k < 1$, where 1 is CPU-bound, and 0 is memory or I/O bound. The operating frequency is set based upon the workload characterisation, where memory-intensive workloads are allocated low operating frequencies to save power, while compute tasks remain at
a higher operating frequency. Furthermore, the policy predicts the workload for the next sample period, which is set according to the previous prediction, weighted by the degree of misprediction. Therefore, it is assumed that rapid changes in the workload do not occur, otherwise this basic prediction mechanism will result in significant errors.

For an operating system to make an informed operating frequency selection, it would ideally be aware of the effects on both power and performance. However, this requires both power and performance prediction which is hard to achieve. Snowdon, Van Der Linden, Petters, and Heiser (2007) proposed a method for predicting performance for each operating frequency by initially characterising the level of memory utilisation for a given workload at each frequency. Thereby, the relationship between each workload, and performance at each alternate frequency to be determined, allowing the performance for each frequency to be predicted. This was later supplemented with a performance event based power model (Snowdon, Petters, and Heiser, 2007), which followed a similar methodology to the performance prediction, enabling the power to be predicted with a reasonable accuracy. Each of these prediction models was then incorporated into the Koala operating system power manager (Snowdon, Le Sueur, Petters, and Heiser, 2009), enabling optimal frequency selection for memory-intensive workloads, through the evaluation of tradeoffs between power and performance. However, the predictability of these values is significantly aided by the use of simple, in-order processors, meaning the methods used may not be as reliable on modern, multi-core, superscalar processors with out-of-order pipelines.

While many of the existing workload characterisation techniques have been used for dynamically controlling processor voltage and frequency scaling, workload characterisation can be used in a range of alternative use-cases. For example, Curtis-Maury, Blagojevic, Antonopoulos, and Nikolopoulos (2008) proposed ACTOR, a workload characterisation method for performing concurrency throttling. A suite of scientific workloads was partitioned into separate phases at the boundary of each parallel code segment, allowing each distinct workload phase, within the application, to be allocated its local optimum thread count. For instance, it is desirable to allocate a low thread count for non-parallel code segments, while allocating more threads during parallel execution, maximising performance. Dynamic thread allocation policies can improve performance and power use, by preventing resources from being over or underutilised. However, this requires the scheduler to determine the number of threads during execution without prior knowledge. This is achieved by characterising the current execution phase as either scalable, \( \text{IPC} \geq 1.0 \), or non-scalable, \( \text{IPC} < 1.0 \), where IPC predictions
are made using a multivariate regression model that is capable of per-core configuration predictions. While this policy does not explicitly consider power when making scheduling decisions, it is still able to reduce power by maximising execution performance, while simultaneously minimising resource utilisation.

A dynamic, run-time workload classification method for modern multi-core processor threads was proposed by Chang, Liu, and Wu (2013), which separately classifies the workload of each thread within a parallel application. At run-time, execution intervals are sampled once every 1 million retired instructions. A collection of sample intervals are then grouped in order to identify an execution phase, where the similarity of workload phases are evaluated using their respective utilisation levels, IPC. This enables accurate, run-time workload classification, without prior knowledge for the given application. To enable this classification methodology to be used for run-time optimisation, several workload prediction methods were evaluated. It was found that 1-level run-length Markov encoding, provided the most accurate phase prediction for an acceptable overhead. The use of execution samples for performing classification was used in order to further mitigate any potential overhead, thereby allowing the model to be applied at run-time.

A common problem with the presented approaches is the dedicated use of all performance events, prohibiting other, related uses of the performance events. For instance, much of the existing work uses all of the available performance events when performing workload characterisation, preventing concurrent performance analysis or power modelling, which are derived separately. Weissel and Bellosa (2002) proposed the future development of specific power event registers within processors, which would ease some of these issues, however, such a solution does not provide for a practical solution for existing systems. Furthermore, the results of a given policy will vary between different microarchitectures, requiring the development of more adaptive, online policies (Miyoshi, Lefurgy, Van Hensbergen, Rajamony, and Rajkumar, 2002). Many of the previously proposed methods rely on microarchitecture-specific features, making them difficult to deploy on different systems.

**Power modelling**

Much of the previous work has illustrated the presence of a strong causal relationship between workload performance and power, where a change in execution workload results in a corresponding change in power consumption. The strength of this relationship is such that power has been used to infer the current workload classification. For
example, Da Costa and Pierson (2011) were able to classify each of the NAS parallel benchmarks using only system power and network communication. Therefore, we can leverage this relationship between specific workloads and power to achieve both objectives, where a single set of workload-specific performance events are used to model power and additionally perform workload classification. To explore the accuracy of power-based workload classification Isci and Martonosi (2006) performed a systematic comparison between the common approach of using application basic block vectors, and performance event based power characterisation. Performance-based power characterisation was able to perform better for all of the evaluated clustering techniques, despite the performance event values being more variable than the basic blocks. Each of the performance event phases was more closely related to the workload-specific power than the basic block classifications, where similar groupings of performance events had similar power values. However, various control flows detected by the basic blocks may have different characteristics, but share a common power phase, making them less suitable for power phase characterisation.

Wu, Jin, Yang, Liu, and Tan (2006) used a collection of tailored micro-benchmarks to individually utilise each of the Pentium 4 processor’s functional units, while recording various performance events and power. These values were then used to derive a linear power model, achieving a close fit to the micro-benchmark data. However, when the power model was evaluated on the SPEC benchmark suite, it was found to perform poorly. This was attributed to differences in execution between the benchmarks and the micro-benchmarks, where the micro-benchmarks used a single instruction to utilise each functional unit, while the benchmarks use a variety of instructions, causing increased power variability. Alternatively, K-means clustering was used to identify the various power phases within the results for the SPEC benchmark. The centroid point within each primary cluster was used to provide the power estimate for the corresponding workload, resulting in a significantly improved estimation accuracy.

While executing a suite of benchmarks in various configurations, Dhiman, Mihic, and Rosing (2010) observed that similar utilisation levels for different benchmarks resulted in markedly different power values. This illustrates a potential limitation of general utilisation metrics, such as IPC, which may provide in insufficient detail to adequately distinguish between alternate workloads. Therefore, the authors proposed the use of a rudimentary clustering of workloads and a Gaussian Mixture Model (GMM), with four input performance events: instructions per cycle, memory accesses per cycle, cache transactions per cycle and CPU utilisation. The resulting power model was
able to outperform both a linear and a multivariate regression model by accurately classifying and modelling high utilisation workloads during evaluation.

Instead of using general, high-level workloads for classification Chetsa, Lefevre, Pierson, Stolf, and Da Costa (2012b) proposed a method for performing fine-grained workload classification, where changes in power or performance denote a workload boundary. The two are treated equivalently due to the close relationship that exists. Each workload is characterised by a column vector of performance event values, that collectively provide a unique identifier for the corresponding execution period. Workload power is initially modelled using the set of performance events, but can subsequently be assigned to each workload for future use. The workload characterisation is performed offline using a full execution trace, which is used to derive a state graph. Each state transition is assigned a conditional transition probability, which can be used during execution to predict future execution states, allowing power policies to be applied preemptively. The workload characterisation procedure was later extended to incorporate partial phase recognition to allow online workload characterisation (Chetsa, Lefevre, Pierson, Stolf, and Da Costa, 2012a; Tsafack Chetsa, Lefèvre, Pierson, Stolf, and Da Costa, 2014). The resulting power model used to evaluate the execution power of alternate cluster nodes and to additionally apply a run-time power saving policy.

A more coarse-grained approach was adopted by Jarus, Oleksiak, Piontek, and Wkeglarz (2014), where applications are grouped into general execution classes according to the similarity of execution characteristics. The initial execution classes are derived from a suite of representative applications, where the power and performance values are used to cluster the applications, resulting in three classes. A decision tree is then constructed from the application clusters, allowing run-time classification of previously unseen applications. Power is modelled separately for each cluster with a multi-variable model, using processor temperature and operating frequency as the primary explanatory variables. However, the proposed cluster restricts each application’s entire execution to a single workload classification, where the initial data set is derived from a restricted set of execution configurations. Varying the number of cores, operating frequency or execution parameters for the application during model development, results in an inconsistent classification method by splitting a given application across multiple clusters. Furthermore, despite processor temperatures providing a strong correlation with power, the observed relationship is strictly dependent upon a stable execution temperature. Unfortunately, under different configurations, such as processor warmup at the beginning of execution, temperature will provide inaccurate
and inconsistent modelling results, limiting model usability.

While workload classification has been used to improve the accuracy of modelling power, the existing approaches are often limited in the generality of workload classification, lacking the required fine-grained generality. Furthermore, each workload is commonly considered in isolation, which may be sufficient for modelling scientific workloads, however general-purpose applications consist of multiple, distinct workload phases during execution. Therefore, a practical power model needs to be capable of classifying periodic execution samples without requiring an initial offline trace.

2.4 Summary

In this chapter I discussed the limitations of commonly used hardware power meters, where the coarse-grained power values make them impractical for use in run-time power management policies. In comparison, performance-based power models are capable of providing significantly finer-grained power values without incurring the costs associated with hardware-based techniques.

In the remainder of the chapter I presented related work for power modelling, where a common limitation in much of the existing work is the microarchitecture-specific training methodology, that is strictly dependent upon the availability of specific performance events. Moreover, many of the existing models make impractical assumptions regarding the real world use-cases, where a single generalised power model is expected to provide accurate results across a diverse set of workload types.

In the next chapter I will discuss the relationship between power and performance within a system, determining the performance events to be used in deriving a power model.
Chapter 3

The Relationship between Power Use and Performance

In this chapter I begin by describing a selection of common methods for run-time performance measurement and their respective limitations for power modelling. I then present the selection criteria for the chosen performance measurement technique, before discussing the interaction between power and performance for key system components. In the remainder of the chapter, I discuss the use of domain-specific knowledge for performance event selection and the advantages for the proposed modelling methodology.

3.1 Techniques for measuring performance

While it is well understood that performance events strongly correlate to system power consumption by providing a proxy for hardware utilisation levels, determining the specific performance events that account for this relationship across various systems remains a challenge. Many alternative methods exist for monitoring application performance, where each has been designed to serve a different purpose. Techniques range from instrumenting applications prior to execution, to monitoring low-level hardware events, intended to unobtrusively glean run-time performance insights. This section presents a range of such monitoring methods, and discusses their measurement granularities and practical use for performance monitoring in existing system deployments.

3.1.1 Source instrumentation

Source code instrumentation is one of the most prevalent techniques for measuring an application’s execution performance. Analysis routines are inserted into the source code
at key points of interest, providing targeted performance values intended to highlight potential execution bottlenecks, where a developer’s optimisation efforts will be most effective. These performance values can be varied but may typically include: the data stored in code variables, memory usage, and the frequency and duration of function calls.

Source code instrumentation is achieved either by manual instrumentation by the developer, where additional code is selectively inserted into the main body of code, or more extensively, through the use of compiler tools, such as Gprof (Graham, Kessler, and Mekusick, 1982), or ATOM (Srivastava and Eustace, 1994). Gprof enables a complete execution trace to be performed, producing a full function call graph, indicating which functions are called during execution and the time spent within each code segment. Alternatively, ATOM allows user-defined analysis routines to be inserted during execution, enabling automated instrumentation of targeted routines.

Unfortunately, this dependence on the availability of application source code imposes some substantial limitations on the usability of such performance monitoring. Most significantly, this excludes performance analysis of widely distributed, commercial applications, commonly used in general-purpose system deployments. Additionally, many of the external code libraries used within applications are distributed as binaries that are included at compile time. Therefore, a large proportion of applications are incapable of being instrumented, regardless of source code availability. Moreover, developers may be reluctant to incorporate persistent code instrumentation, due to the potential overhead that will be incurred. Any detrimental impact on execution performance will be unacceptable to most commercial solutions, which commonly strive to maximise performance.

Despite these issues, it is not a practical solution for run-time performance measurement, primarily due to the collection of a full execution trace, with the resulting performance profile being used for offline post-processing. Furthermore, Hsu and Kremer (2003) noted that source code instrumentation can alter the execution characteristics of an application, resulting in unrepresentative power and performance measurements. Finally, the whole application measurement granularity is insufficient to distinguish between individual thread contributions on shared resources, limiting the usability on modern multi-core systems.
3.1.2 Binary instrumentation

Binary instrumentation enables performance analysis routines to be dynamically inserted into an application binary when the source code is not available. A common method of achieving this is by using JIT (Just in Time) recompilation to execute predefined analysis routines within the source binary whenever a specified set of conditions are met. The possible trigger conditions may include: calls to a specific function, data allocation, any function call and memory writes, among others. However, the specific trigger conditions for an analysis routine will be dependent upon the type of analysis to be performed. For instance, valgrind provides extensive analysis of an application’s memory usage, therefore analysis routines are triggered for almost every instruction, thereby enabling it to track the validity of memory accesses and allocations. Other, common frameworks for binary instrumentation exist, such as the Pin tool from Intel (Luk, Cohn, Muth, Patil, Klauser, Lowney, Wallace, Reddi, and Hazelwood, 2005) and ETCH (Romer, Voelker, Lee, Wolman, Wong, Levy, Bershad, and Chen, 1997), which support user-defined analysis routines.

Despite binary instrumentation overcoming the requirement for source code availability, it still suffers from many of the other limitations of source instrumentation. The primary limitation of binary instrumentation is the performance overhead incurred during execution. For example, Pin was found to have a base overhead of 30% from the use of JIT, before any analysis routines are even executed. Depending on how extensive the intended analysis is, this overhead will potentially become prohibitive. Many techniques have been proposed to mitigate such overhead including limiting the profiling time by dynamically attaching/detaching the profiler during execution. While this may achieve the intended objective, it fails to meet the requirement of persistent run-time performance measurement for power modelling.

3.1.3 Performance Monitoring Counters

Performance Monitoring Counters (PMCs) are a set of special-purpose hardware event registers capable of measuring low-level, fine-grained processor performance events. These performance events include: the number of FPU instructions, cache misses, retired micro-operations and dispatch stalls, among others. The fine-grained detail provided by the performance events enables insights into execution performance that is not possible with the other performance monitoring techniques. Performance analysis is further aided by the per-core placement of the performance registers, allowing per-
thread performance analysis of parallel applications, or the independent monitoring of applications concurrently executing on a single multi-core processor.

Unfortunately, the use of specific processor registers results in a microarchitecture-dependent solution, where support can be inconsistent between processors. Many profiling tools have been developed in an attempt to mitigate some of these issues for end users, including, perf,1 Oprofile2 and PAPI (Mucci, Browne, Deane, and Ho, 1999). For instance, PAPI (Performance Application Programming Interface) provides a high-level, abstract interface for accessing a small set of widely supported performance events. Thereby, the user can use the API to read the desired performance event, while the low-level PAPI functions handle the microarchitecture-specific register details.

However, such tools do not address the issue of architectural dependence, but merely use abstraction to mitigate some of the impact. The most prevalent microarchitectural variations are in the number of supported performance events and the number of processor registers, which determine the number of performance events able to be simultaneously monitored. For example, the AMD 10h processor family (AMD, 2013) supports four performance registers for monitoring 120 possible performance events, while the Intel Sandy Bridge (Intel, 2013) microarchitecture supports eight per-core event registers, providing four per thread when using hyper-threading. Therefore a large difference exists between the number of possible performance events and the number that can practically be monitored simultaneously during execution. Some of this limitation is mitigated by the performance tools using event multiplexing, where the selected performance events are dynamically changed at run-time, providing event sampling instead of constant measurement. Furthermore, seemingly similar performance events may have different low-level implementations on alternate microarchitectures. For instance, a measure of the FPU utilisation is common across architectures, however differences may arise in the inclusion of specific instructions, such as x87, SSE and MMX, in the corresponding event count. While hardware performance events allow for substantially more detailed performance analysis, greater care is required during their use.

### 3.1.4 Operating system events

While the previous monitoring techniques were able to provide detailed analysis of processor and memory performance, they are incapable of measuring the performance

---

1https://perf.wiki.kernel.org/index.php/MainPage
2http://oprofile.sourceforge.net/
of other system components, such as the hard drives and network adapters. Instead, the operating system is uniquely positioned to monitor the performance of all system components, enabling a broad picture of the entire system’s performance. This is possible because much of the inter-component interactions are handled by the operating system through its control of the system bus, which is outside the scope of most user-level performance monitoring tools.

This has led to the development of profiling tools, like Oprofile and DTrace,\(^3\) that use special-purpose kernel modules to incorporate run-time monitoring into the operating system. The resulting kernel-level access enables the monitoring of system calls, system interrupts and bus utilisation, among others. However, the dependence on kernel modules reduces the portability between different operating systems.

### 3.1.5 Virtual machine performance

Virtualised environments present a challenging problem for performance monitoring, where guest machines do not have direct access to the underlying hardware. This prevents the use of many common profiling tools, such as Oprofile and perf, which rely on hardware performance registers to perform fine-grained analysis. Therefore changes are required for the virtual machines hypervisor, the guest operating system or user-level profiling applications that are run inside the guest machine.

Xenoprof (Menon, Santos, Turner, Janakiraman, and Zwaenepoel, 2005) is one such solution, that is used to instrument the Xen hypervisor and the guest operating system, enabling access to the hardware registers. Xen does not virtualise the required registers, but through the use of para-virtualisation, the guest can be made aware of the performance event registers, which can be accessed through the instrumented hypervisor. By using the hypervisor as an interface to the event registers, concurrent guests can be properly handled, ensuring all performance events are associated with the corresponding VM execution.

Alternatively, fully virtualised environments, such as VMware, require performance event registers to additionally be virtualised (Serebrin and Hecht, 2012). However, the emulation of event registers is complicated by the differences in emulated and native instruction execution, where emulated instructions require a larger number of operations than natively executed instructions. Therefore, the performance count for emulated instructions may result in erroneous values, where the utilisation is substantially mis-

\(^3\)http://dtrace.org/
represented, requiring performance events to be estimated. While virtualised event registers requires changes to the hypervisor, all changes are transparent to the guest system, allowing the use of standard profiling tools.

3.2 Performance monitoring selection criteria

Each of the alternate performance monitoring techniques impose a number of restrictions on the execution environment, intended to meet a specific set of requirements. Therefore, a number of tradeoffs need to be considered during initial selection, which may include:

**Sample rate** Performance events need to be measurable at a sufficiently high sample rate to allow for run-time power modelling, ensuring power estimates remain responsive to rapid changes in application execution characteristics. For instance, the execution traces provided by source instrumentation fail to meet this criteria, preventing the use of run-time performance events.

**Measurement granularity** Fine-grained performance events are necessary to achieve accurate, application-specific power modelling. The prevalence of multi-core processors has made it increasingly common for system resources to be shared between concurrently executing applications, requiring fine-grained performance events for determining per-thread or per-application utilisation levels.

**System agnostic** For a power model to remain usable across a variety of alternate systems, the selected performance method should have minimal system-specific requirements, helping ensure broad adoption. For example, Xenoprof is not system agnostic as it has strict dependence on the system hardware, virtualisation environment, guest operating system and performance profiling application.

**Application agnostic** Similar to the system agnosticism, the chosen method should not impose strict requirements on the form of a given application, such as the requirement of instrumented source code.

**Run-time overhead** The persistent execution of power models requires performance monitoring techniques incur minimal run-time overhead to provide an effective solution. Otherwise, this will inhibit adoption in performance sensitive environments, where any overhead is likely to contribute to increased execution times, resulting in higher overall power consumption.
In evaluating the selection criteria, hardware performance monitoring counters are the chosen source of run-time performance events, used to derive a dynamic power estimation model. The selection can be attributed to four primary advantages. First, performance monitoring counters are the only method capable of providing the required fine-grained, per-core performance measurements on the experimental system. This has become an increasingly important requirement for modelling power, where the number of cores within a single processor is continually increasing, resulting in a greater number of concurrently executing tasks. Therefore, it is necessary to model power at a sub-component level. Second, the overhead of reading performance counters is substantially lower than many of the alternate techniques owing to the underlying hardware implementation. Therefore, negligible overhead is incurred when reading each event register, which does not require additional instrumentation of applications. Third, the frequent updates to processor event registers allow performance counters to provide very high sample rates, facilitating very fine-grained performance analysis. This additionally enables power to be modelled within fast changing environments or for smaller execution phases than would otherwise be possible. Finally, hardware performance events are broadly supported on all major processor microarchitectures, including x86, x64, ARM and PowerPC. This enables the general methodology to be applied on a diverse set of systems.

However, this is not to say that hardware performance events are without their respective limitations, which are primarily due to the resulting hardware dependence. While event registers are broadly supported in modern processors, some variation exists between alternative microarchitecture implementations, which can be attributed to two main points of difference. First, the number of physical, per-core performance registers varies between processor microarchitectures, limiting the number of events able to be monitored simultaneously. While this has been a historical limitation, with older microarchitectures providing a limited number of registers, often only two, this limitation is becoming less restrictive with newer microarchitectures. Table 3.1 illustrates the current trend of increasing numbers of performance registers for newer microarchitectures, mitigating some of the limitations. Alternatively, performance event multiplexing can be used to help mitigate some of the restrictions by enabling event sampling instead of persistent measurement, allowing events to be periodically monitored. Second, each microarchitectures will provide a different set of measurable performance events which is dependent on the supported functional units and instruction set. However, many of the most commonly used performance events are supported across alternate architectures.
Furthermore, the impact of implementation-specific details can often be mitigated by profiling tools that provide a high-level interface for accessing common performance events, irrespective of the underlying hardware.

Table 3.1: An example of the increasing number of hardware performance event registers per core for different processor micro-architectures.

<table>
<thead>
<tr>
<th>Chip maker</th>
<th>Processor family</th>
<th>Number of per-core PMCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD</td>
<td>Opteron 10h</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Opteron 15h</td>
<td>6</td>
</tr>
<tr>
<td>ARM</td>
<td>Cortex-A5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Cortex-A8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Cortex-A9</td>
<td>6</td>
</tr>
</tbody>
</table>

While it may be perceived that using processor-centric performance measurements will unnecessarily restrict power analysis to the processor, this is not necessarily the case. This can largely be attributed to the processor playing a central role in all computations occurring within a system, where the utilisation levels of other system components can be inferred through indirect observations. For instance, Bircher and John (2007) were able to derive separate power models for each system component, such as the hard drive and network adapter, using only processor performance events to model the utilisation of each component. Regardless of this capability, it may prove to be unnecessary to model every system component, as their respective variation in dynamic power may be negligible in many cases. It was noted by Fan, Weber, and Barroso (2007) that processor power dominates many system deployments, making it the most important component to model. The interaction between power and performance for different system components will be discussed further in the next section.

Despite hardware performance events imposing some limitations on their general use and deployment, these have proven not to be prohibitive to their widespread adoption in standard performance profiling. Many of the techniques commonly used to mitigate these issues, such as profiling tools, equally apply in our proposed use-cases. Therefore, these issues are not seen as inhibiting the use of performance counters in power modelling.
3.3 Interaction between power and performance

While the performance characteristics of system components are generally well understood, the corresponding power characteristics remain less so. This can largely be attributed to the interaction between system components, where the performance of each component is considered to be independent, power has a number of potential flow-on effects, creating interdependencies between components. Therefore, the relationship between power and performance can only be understood after careful consideration of such effects. This section presents an overview of the power and performance states for key system components and their potential interactions.

3.3.1 Central Processing Unit

The processor is the most significant system component for both power and performance analysis, given the central role it plays in all computations, where the execution performance of compute tasks is strictly dependent on the processor’s operating frequency. For typical system deployments, the processor consumes the largest amount of workload-dependent power, which is determined by the performance states and corresponding utilisation level. Consequently, it is through the processor that the relationship between power and performance is most evident, and can therefore be modelled if adequately understood.

Performance

The run-time variation in processor utilisation levels, and corresponding execution performance, are a consequence of the diverse range of different workload types that can be run on a given system. The possible workload types lie somewhere on a spectrum between the two extremes of either: completely CPU-bound or memory-bound executions. A CPU-bound application consists of a large number of calculations, creating a compute-intensive workload, requiring a small working set of data that is capable of fitting within processor cache. This results in high processor utilisation, where execution time is dominated by the computations. Alternatively, the execution time of memory-bound applications is dominated by the memory latency, incurred by fetching data from memory or disk. Persistent data fetches stall the processor’s execution, which remains idle while awaiting data, resulting in low processor utilisation.

While this may seem to merely be a conceptual form of workload characterisation, it can practically be achieved using hardware performance counters. The low-level
performance events allow individual processor sub-components to be measured, which can then be used to infer the type of workload. For instance, the CPU-boundedness can be determined by the dispatch-stalls and retired-\(\mu\)-ops performance events which represent the level of processor utilisation and the number of processor operations performed. Alternatively, memory-boundedness can be indicated by the L2-cache-miss and L3-cache-miss performance events, which represent the number of memory access during a given execution period.

**Power**

Due to the variations in application workloads, the corresponding processor utilisation, and a sizeable idle power, the processor experiences a large range of dynamic power values, making it the system component with the largest power consumption. For instance, Fan et al. (2007) found processors accounted for 37% of total system power within a data center that used low power processors. Alternatively, Lim, Ranganathan, Chang, Patel, Mudge, and Reinhardt (2008) observed a larger processor power consumption of 61% within a server, illustrating a difference in the range of power values.

The most commonly used metric for comparing processor power consumption is the Thermal Design Power (TDP), which defines the maximum power a cooling system is expected to dissipate while executing real applications. TDP values can significantly vary between processor models, ranging from 8.5W to 135W, as can be seen in Table 3.2. However, this only presents a static, nameplate value for the expected maximum power, where the actual power consumption during execution responds dynamically to utilisation levels. A processor dynamic power consumption is often represented as \(P = \alpha CV^2 f\), where \(P\) is power, \(\alpha\) is that activity factor, \(C\) is the circuit capacitance, \(V\) is the processor voltage and \(f\) operating frequency. Therefore, as the processor utilisation increases, either through the operating frequency or the activity factor, the power consumption increases accordingly. For example, the AMD Opteron 8380 has a TDP of 115W, and an average power consumption of 75W while executing a standard application. Lowering the operating frequency and voltage from 2500MHz (1.325V) to 800MHz (1.050V), reduces the TDP to 53.9W (AMD, 2010), illustrating the dynamic range a processor might be expected to operate within.

Despite the TDP defining the intended maximum power consumption, this threshold may still potentially be exceeded. In such cases, the processor will begin to throttle performance. Failing this, a full processor shutdown will be triggered, preventing any
Table 3.2: A range of processor power values taken from processor specification sheets.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cores</th>
<th>Threads</th>
<th>Frequency (MHz)</th>
<th>TDP (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Atom S1260</td>
<td>2</td>
<td>4</td>
<td>2000</td>
<td>8.5</td>
</tr>
<tr>
<td>AMD Athlon II M320</td>
<td>2</td>
<td>2</td>
<td>2100</td>
<td>35</td>
</tr>
<tr>
<td>AMD Athlon 64 X2 4600+</td>
<td>2</td>
<td>2</td>
<td>2400</td>
<td>110</td>
</tr>
<tr>
<td>Intel Core i5-2320</td>
<td>4</td>
<td>4</td>
<td>3000</td>
<td>95</td>
</tr>
<tr>
<td>Intel Core i7-920</td>
<td>4</td>
<td>8</td>
<td>2667</td>
<td>130</td>
</tr>
<tr>
<td>AMD Opteron 8380</td>
<td>4</td>
<td>4</td>
<td>2500</td>
<td>115</td>
</tr>
<tr>
<td>AMD Opteron 6276</td>
<td>16</td>
<td>16</td>
<td>2300</td>
<td>115</td>
</tr>
<tr>
<td>AMD FX-4100</td>
<td>4</td>
<td>4</td>
<td>3600</td>
<td>95</td>
</tr>
<tr>
<td>Intel Xeon E5-2420</td>
<td>6</td>
<td>12</td>
<td>1900</td>
<td>95</td>
</tr>
<tr>
<td>Intel Xeon E7-2890 v2</td>
<td>15</td>
<td>30</td>
<td>2800</td>
<td>155</td>
</tr>
<tr>
<td>Intel Xeon X6550</td>
<td>8</td>
<td>16</td>
<td>2000</td>
<td>130</td>
</tr>
</tbody>
</table>

further heat generation before physical damage can occur.

The final consideration in processor power consumption is the availability of sleep states, which can significantly reduce power consumption. This is achieved by stopping all processor operations, and flushing the cache. However, additional latency will be incurred upon wakeup for restoring the previous processor state.

**Relationship between power and performance**

The relationship between power and performance has been leveraged by many power saving policies, where the processor’s operating frequency is adjusted at run-time in response to the workload characteristics of the current application. Lowering the processor’s operating frequency will reduce power consumption at the cost of execution performance. While this will have a detrimental impact on CPU-bound workloads, it will have a negligible impact on memory-bound workloads, whose execution time is dominated by the memory latency. Therefore, power can be saved by setting a lower processor operating frequency in response to the memory-boundedness of a given application. A number of alternate policies have been proposed for evaluating the run-time tradeoff between power and performance, in order to maximise power savings without adversely impacting performance.

For example, Figure 3.1 illustrates the relationship between power use and performance for an instance of the raytrace benchmark run in a variety of execution configurations. The OpenMP implementation of raytrace is adapted from (Woo, Ohara,
Torrie, Singh, and Gupta, 1995) and is compiled using gcc-4.4.1, using the optimization argument -O3. The experiments are run on a Dell PowerEdge R905 with four quad-core AMD Opteron 8380 processors, with further setup details given in Chapter 5. The x-axis plots the number of cores used, while the y-axis is the average power consumption for the entire system during execution. DVFS is used to set the operating frequency of each processor core to one of the four available frequencies i.e., 800, 1300, 1800 or 2500MHz. All idle cores are set to the lowest frequency, 800MHz, to minimise power consumption. Sleep states are not used as this functionality is not available on the experimental system. From Figure 3.1 it can be seen that for all operating frequencies, the total power consumption increases fairly linearly with the number of cores used, where each frequency has a different linear function. This is due in part to the scalability of the raytrace benchmark, which does not suffer any synchronisation bottlenecks when executing with 16 cores.

Furthermore, it can be seen that running at a higher performance level, higher operating frequency, results in a significantly higher average system power consumption. For instance, at 800MHz, the average power consumption is 254W, while at 2500MHz, power consumption rises to 402W. This illustrates the tradeoff to be made between power and performance, where an increase in performance typically comes at the cost of increased power use. However, the corresponding change in execution performance,
measured by total execution time, is not shown in the figure. At 800MHz, execution takes $2.6 \times$ longer than at 2500MHz. The significant degradation in execution performance is due to the compute-intensive workload of the raytrace benchmark. The increased execution time results in an increased energy consumption, $1.65 \times$ greater than the highest operating frequency.

Many of the existing power saving policies primarily benefit from memory-bound workloads, requiring the development of different policies for compute-intensive tasks. One such power saving technique adopted in modern microprocessors is the use of aggressive clock gating, where power is saved by temporarily shutting off parts of a processor’s circuitry while not in use. However, the use of hardware level power saving methods may result in hidden power states, outside the scope of the operating system, making it prohibitive for the operating system to effectively monitor and manage execution power.

### 3.3.2 Memory

While component-specific performance is readily identifiable, the power consumption for the processor and memory has a more complex interdependent relationship. This was indirectly shown in the previous section, where memory-bound workloads stall processor execution, causing lower processor utilisation and power use, which will be partially offset by an increase in memory power consumption from additional read/write operations. However, any changes in the power consumption of memory modules cannot be isolated from corresponding changes in processor power, due to changes occurring simultaneously.

This can result in potentially erroneous conclusions being drawn as to the significance of memory power. For instance, memory is often thought to consume a sizeable amount of power within a system, but the specification for an 8GiB Kingston memory module defines the maximum power as 2.041W (Kingston, 2014). For a system with eight memory modules (64GiB), the maximum memory power draw will be 16.328W, which may be insignificant in the context of other system components. Lim et al. (2008) found the memory modules accounted for only 7% of total power for a 340W server.

Much of the perceived impact of memory power may be attributable to the corresponding interdependence with processor power. However, this relationship may be further obscured by thermal effects and the corresponding power consumption of the cooling infrastructures.
3.3.3 Fans

The role of inter-component power effects is evident when considering the relationship between the processor and cooling infrastructure, where the processor’s thermal load increases in response to a rise in the utilisation level. The rising operating temperatures prompt the system fans to provide additional cooling, resulting in higher overall power consumption. This action-response relationship may be expected to be immediate, experimental results have shown this to be a general misconception, which is largely due to an inherent latency in processor thermal effects (Chung and Skadron, 2006). While processor power and performance rapidly fluctuate during execution, the corresponding temperatures take time to build up before a notable thermal response is triggered. For example, the experiments in Section 7.2, show that during micro-benchmark execution, it took about 400 seconds to reach a stable operating temperature, despite the stable performance level. This provides some indication of the potential disconnect between the observed processor’s power/performance and the power of directly related components, such as system fans.

However, any observed relationship may be further obscured by the thermal effects of neighbouring system components or compute nodes, where a rise in the ambient temperature will influence the effectiveness of the cooling system. For instance, in the experiments conducted by Kodama, Itoh, Shimizu, Sekiguchi, Nakamura, and Mori (2013), it was found that the operating temperatures of neighbouring nodes within a rack, influenced the operating temperature and power consumption for a given workload execution. Therefore, it may be difficult to observe the anticipated relationship between execution performance and operating temperature due to external factors, outside the scope of monitoring, having an undesired influence.

Furthermore, similar effects may be observed within a system, where highly utilised components influence the temperatures of other internal components. While performance events can be used to model per-core power, the corresponding thermal effects are not able to be isolated at the sub-component level, as cooling acts upon the entire processor. Therefore, the temperature of a single core will influence the power and temperature of all neighbouring cores. Overall, the power consumption of the processor fans can account for a sizeable 11.7% of total power (Lim et al., 2008), requiring some consideration in modelling power.
3.3.4 Hard drive

A number of memory accesses will result in pages being fetched from the hard disk, incurring additional access latency as data transfers are significantly slower than memory. However, any large data transfers will be independently handled by a direct memory access (DMA) operations, allowing the processor to continue task execution while the transfer proceeds. As a result, the processor will not be aware of the hard disk utilisation level at any given point of time. However insights may still be indirectly gleamed from key performance events. Bircher and John (2007) were able to accurately model hard disk power at various utilisation levels using two performance events, DMA accesses and processor interrupts. These two events were sufficient to capture all of the variations in utilisation, where the interrupts delineate the start and end of data transfers and DMA accesses track the access count.

While it is possible to accurately model hard disk power, it may not be necessary in practise, due to the relatively narrow range of power use while running. For instance, the WD VelociRaptor,\(^4\) shown in Table 3.3, has a maximum read/write power of 5.8W and an idle power of 4.2W, providing a maximum saving of 1.6W. This is a reasonable power saving as a proportion of total disk power, but is an insignificant share of server power, which can exceed 360W. Instead, a constant value for hard drive power may be considered an acceptable tradeoff between model accuracy and complexity, in certain circumstances. A similar case can be made for the more efficient solid state disk, such as the WD SiliconDrive,\(^5\) also shown in Table 3.3. However, the number of hard disks will determine the total error of using a constant value, where a large number of disks may require modelling.

However, the overall power contribution has been found to be almost insignificant in other related work. For instance, Fan et al. (2007) determined that hard disk power does not need to be modelled, due in part to the low range of dynamic power values. Furthermore, hard disk power was found by Lim et al. (2008) to account for 4.4% of full system power. Such observations support the general assertion that hard disk power does not require explicit modelling.

\(^5\)http://www.wdc.com/global/products/specs/?driveID=1120&language=1
Table 3.3: Power states for two Western Digital hard drives.

<table>
<thead>
<tr>
<th></th>
<th>WD VelociRaptor Workstation Hard Drive</th>
<th>WD SiliconDrive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Read/Write</td>
<td>5.1 Watts</td>
<td>1.0 Watts</td>
</tr>
<tr>
<td>Sequential Read/Write</td>
<td>5.8 Watts</td>
<td>1.0 Watts</td>
</tr>
<tr>
<td>Idle</td>
<td>4.2 Watts</td>
<td>0.4 Watts</td>
</tr>
<tr>
<td>Standby and Sleep</td>
<td>1.1 Watts</td>
<td>-</td>
</tr>
</tbody>
</table>

3.3.5 Network

Another key system component considered during power modelling, is the network interface controller (NIC), which acts like an I/O device in data centers that use network storage, where data is periodically transferred from remote servers. Similar to the hard disks, the NIC has a narrow power range, using nearly as much power while idle as during active operation. For example, Sohan, Rice, Moore, and Mansley (2010) analysed the power for two 10 gigabit Ethernet NICs, which consumed 18.0 and 21.2 Watts while idle. When under a heavy load, power increased by a negligible 0.2W for both NICs. Network activity did not effect power as much as was expected. Similar results were observed in the fiber protocol network adapter that had an offload processor. While the offload processor doubled idle power, it was found to result in a negligible power difference when switching from idle to active states.

For the given network adapter details, it is convenient to model network adapter power as a constant value, with a small estimation error. This is possible due to the narrow range of dynamic power, which is likely attributable to the requirement for the network state to be maintained. Furthermore, with a total power of about 20W, network adapters constitute a small proportion of total system power, making it uneconomical to model their power, given the additional overhead incurred by monitor network utilisation.

3.3.6 Idle system power

Power models seek to quantify the component-specific power response for various utilisation levels. However, many existing power models do not explicitly consider the base power while idle. This is important for many large-scale deployments, where many nodes can spend a significant proportion of time idle (Barroso and Holzle, 2007). Not
incorporating idle workload phases in a power model will lead to a divergence with real world power use over time. For simple deployments, this may occur at the beginning and end of executions, as was observed by Jarus et al. (2014).

Power models essentially estimate the dynamic power of a system by quantifying the dynamic response in power for a corresponding change in the utilisation level. Therefore, the system’s base power, that does not change with utilisation is the static power. The static power can include many unmodelled components such as, the motherboard and power supply unit, that are not easily measured. At a finer granularity, each system component can be thought of as containing both static and dynamic power. For instance, a processor power model solely derived from the corresponding utilisation may naively imply that no power is consumed while idle (no utilisation). However, the processor’s persistent base power is indirectly included in the system’s base (static) power. It will be shown in the model evaluation, Chapter 6, that failing to make a distinction between static and dynamic power has the potential to misrepresented modelling accuracy.

3.4 Method of performance event selection

A primary advantage of the hardware performance counters is the ability to perform fine-grained performance analysis of the processor’s many functional units. However, monitoring all of the possible performance events presents a substantial challenge, where the typical number of alternate events can exceed 120. A further restriction is imposed by the number of performance event registers, limiting the performance events that are able to be monitored simultaneously. Such limitations may make it prohibitive to perform a thorough performance analysis while evaluating a sizeable benchmark suite.

Moreover, a detailed analysis is required in determining the performance events most strongly correlated with power, for deriving the power model. This set of selected performance events has to be capable of modelling the power variation for a diverse set of alternate workloads, each of which may require a distinct set of performance events. For instance, a task that performs a large number of floating point calculations will heavily utilise the FPU, with infrequent memory accesses. In contrast, an integer-intensive workload will still heavily utilise the processor, but this utilisation will be reflected in a different set of functional units. Such variations exist for the multitude of alternate workloads, where each execution configuration ideally requires a different set
of performance events to accurately describe the corresponding power characteristics.

However, exploring the full suite of performance events for each of the possible workloads will be impractical. Therefore, the challenge is in determining a set of representative performance events that strongly correlate with power, across a diverse set of alternate workloads.

3.4.1 Event exploration

The first step taken when deriving the power model was to gain an initial indication of which performance events strongly correlate with power. This was achieved by running a number of benchmarks while logging the power and performance values. The perf tool was used to collect aggregate performance values, while the power meter periodically logged power. In total, 168 different performance events were collected, made up of 41 perf tool events and 123 processor-specific performance events. A full list of the performance events is given in Appendix A. The performance events were collected four at a time, requiring 42 iterations for a single set of performance values. This was necessary as the AMD Opteron 8380 processor only has four hardware performance event registers per-core, limiting the simultaneous collection of performance events without the use of event multiplexing. The experimental setup is discussed in more detail in Section 5.1. Additional iterations were performed to collect multiple data sets for each execution configuration. While this makes for a time consuming process, it allows thorough event analysis to be performed, ensuring robust conclusions are drawn, without any perceived bias.

The results showed that many of the collected performance events did not have a meaningful correlation with power. For example, out of the 168 performance events collected for the Mandelbrot benchmark, adapted from an implementation made by Zhang, Huang, Chen, Huang, and Zheng (2008), less than half of the events returned a non-zero value, 80. From this, 32 of the events had a weak correlation, shown in Table A.1 in appendix A. A further 15, shown in Table A.2, were found to have a reasonably strong correlation for various executions of the mandelbrot benchmark. Subsequent analysis of the most strongly correlated performance events found that they were strongly associated with the processor’s primary functional unit utilisation, including: FPU utilisation, dispatch stalls, and retired operations, among others. This indicates a strong, intuitive relationship between processor utilisation and power consumption.
3.4.2 Utilising domain-specific knowledge

While it is possible to perform an exhaustive analysis of all performance events during initial selection, the previous discussion illustrated that such analysis may not be necessary. Instead, a substantially smaller set of representative performance events can be used, as the most prevalent performance events were found to correspond with processor power consumption. Therefore, domain-specific knowledge can be used to narrow the scope of the initial performance event evaluation.

The resulting set of performance events is intended to be broadly representative of the processor’s various utilisation levels while executing a diverse set of workloads. Such a set of performance events are shown in Table 3.4. For example, the FPU event indicates the utilisation of the floating point functional unit, while retired-move provides a measure of the number of completed, non-FPU operations. Alternatively, the efficiency of the execution pipeline can be measured by the dispatch-stalls, while the rate and frequency of memory accesses can be gleamed from the L2 and L3-cache-misses. This selection of performance events provides various metrics related to the potential causes of power consumption.

Table 3.4: A selection of representative performance events available on the AMD 10h family processors (AMD, 2013).

<table>
<thead>
<tr>
<th>Performance event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>Number of execution cycles</td>
</tr>
<tr>
<td>cycles-not-halted</td>
<td>Clocks CPU was not in halted state</td>
</tr>
<tr>
<td>FPU</td>
<td>Cycles an FPU operation is present</td>
</tr>
<tr>
<td>dispatch-stalls</td>
<td>Number of cycles the decoder is stalled</td>
</tr>
<tr>
<td>prefetch-inst</td>
<td>Prefetch instruction dispatched</td>
</tr>
<tr>
<td>retired-move</td>
<td>Number of move micro-operations retired</td>
</tr>
<tr>
<td>retired-µops</td>
<td>Retired micro-operations</td>
</tr>
<tr>
<td>retired-branch-inst</td>
<td>Retired branch instructions</td>
</tr>
<tr>
<td>dcache-miss</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>icache-miss</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>L2-cache-miss</td>
<td>Level 2 cache misses</td>
</tr>
<tr>
<td>L3-cache-miss</td>
<td>Level 3 cache misses</td>
</tr>
<tr>
<td>DTLB-miss</td>
<td>Data cache access that miss L1 and L2 DTLB</td>
</tr>
<tr>
<td>DRAM-access</td>
<td>Number of memory accesses by DRAM controller</td>
</tr>
</tbody>
</table>
The use of domain-specific knowledge provides the following primary advantages to power modelling:

**Reduced selection time** A large data set will require time consuming analysis, where the use of domain-specific knowledge allows the set to be restricted to the key performance events of interest.

**Model portability** Deriving a model from a reduced set of broadly supported performance events, such as those shown in Table 3.4, ensures a high degree of model portability between alternate microarchitectures.

**Manual correlation** A smaller data set affords the possibility of performing manual correlation analysis to gain additional detailed insights that would not otherwise be possible using typical summary statistics, i.e. by plotting the performance events.

Domain-specific knowledge provides a practical method of performing performance event selection by narrowing the scope of the evaluated events. This is necessary for the development of a readily deployable modelling methodology, where hardware limitations make an exhaustive analysis impractical.

### 3.5 Summary

In this chapter I evaluated various sources of performance values before selecting processor hardware performance monitoring counters as the source of performance events in the proposed power model. These were chosen as they are the only source of performance events capable of providing fine-grained, per-core performance values at a high sample rate.

Following this, I discussed the relationship between power and performance for many of the key system components. While the performance of different components is considered to be independent, power has many flow-on effects, creating interdependencies between system components. This discussion was used to determine that the processor and memory are the primary system components to be modelled.

Finally, I presented some experimental observations supporting the use of domain-specific knowledge in restricting the set of performance events to be evaluated while deriving the power model. In the next chapter I will present the proposed modelling methodology.
Chapter 4

Modelling Methodology

In this chapter I present a novel performance-based power modelling methodology that utilises workload classification. I begin by outlining the principles behind my proposed modelling methodology, describing the parameterised micro-benchmark used to generate performance values for a range of general workload types. I then present a general method for modelling the workload-specific power from the micro-benchmark data, describing the steps for performing workload classification. Finally, I discuss some of the advantages of adopting such a general approach to modelling power consumption and how this may aid the broader objectives of power-aware policies.

4.1 Power model with workload classification

The proposed modelling methodology derives a collection of workload-specific linear functions for run-time power estimation, using performance events. The main principle of the proposed modelling methodology is that classifying a given workload type allows for the use of a specific linear function for power estimation of that workload type, rather than relying on a single modelling function for all workload types. The use of multiple linear functions has the distinct advantage of being able to estimate power consumption for a much broader range of applications than would otherwise be possible, because it is not feasible to use a single linear function to characterise the power features of various workload types in a wide range of applications. For example, the power consumption of FPU calculations are linear in the number of FPU operations, which are less related to the number of cache misses. Alternatively, the memory power consumption is related to the number of cache misses, which is largely independent of FPU operations. Consequently, deriving a single function that uses both FPU
operations and memory accesses (cache misses) is not as effective as using two separate linear functions for each workload type respectively.

The power model is able to accurately estimate power over a diverse set of application workloads by applying a general classification method to detect, e.g. FPU-intensive or memory-intensive workloads. While this creates the additional requirement of a workload classifier, it helps meet the broader objective of improving power-aware management policies by providing additional context for the power consumption, i.e. the workload causing the power use.

4.2 Synthetic micro-benchmark

To derive an accurate power estimation model, the training data must provide representative values for the widest possible range of workloads. While it is common for a single benchmark suite to be used as the source of model training data, the scope of performance values may be insufficient to derive a broadly representative model. This is primarily due to such benchmark suites largely consisting of scientific workloads that are dominated by compute-intensive tasks, resulting in an inherent bias towards compute tasks in the model.

Therefore, to derive a benchmark independent power model a single micro-benchmark is used to reproduce a selection of synthetic workloads. While some previous work used two sets of benchmarks, one for initial performance event selection and the other for deriving the power model, the proposed model is able to achieve both objectives with a single parameterised micro-benchmark. This allows the model to be more readily deployed on different systems as there is no requirement for developing new micro-benchmarks based upon the results of performance event selection. Such a requirement will be prohibitive in many common use-cases.

4.2.1 Reproduced workloads

For the model to accurately estimate power for a diverse range of workloads, the micro-benchmark is designed to reproduce a set of representative synthetic workloads that provide samples within the expected range of typical user applications. Otherwise the model will be incapable of accurately estimating power for a wide range of application workloads, as a strong bias towards a few reproduced workloads would exist. However, this does not mean it is necessary to reproduce a large number of workloads to derive an effective model. For instance, through experimental observations, it was found that
the following five key workloads were capable of providing representative samples for commonly encountered application workloads.

**FPU** This workload consists of a floating point intensive workload, performing a large number of floating point multiplications and divisions on several double precision variables.

**INT** The INT workload is intended to reproduce a high processor utilisation level by performing a large number of integer multiplication and division operations.

**Cache** The workload execution contains a large number of data accesses of a small block of memory that is capable of fitting within processor cache. A random access pattern is performed to ensure various cache levels are utilised, as the entire data block will not fit within L1-cache.

**Memory** A large number of random memory accesses are performed on a block of allocated memory that is unable to fit in processor cache. The pseudo random access pattern attempts to increase the number of cache misses by reducing the likelihood of sequential memory accesses.

**NOP** This synthetic workload executes a number of `NOP` assembly instructions which are intended to represent a busy idle execution. Such instructions are inserted by compilers to perform memory and instruction alignment.

While five initial workload types are presented, the proposed modelling methodology is not restricted to these workloads, but is extensible as additional workloads do not fundamentally alter the model. Furthermore, a mix of different workload types can be adopted to improve model generality. For instance, memory accesses are unlikely to occur in isolation of all other workloads. Instead, cache misses may occur in coordination with a compute task, such as FPU or INT, where the corresponding power consumption is more accurately represented by multiple execution workloads.

### 4.2.2 Mixed workload execution

The intention of the micro-benchmark is to provide a configurable method of generating various workload-specific utilisation levels that encompass anticipated application workloads. However, much of the previous work executed each workload-specific micro-benchmark independently, isolating the workload from any potential interactions with other workloads. In contrast, the micro-benchmark presented here uses a
combination of the selected workloads, where the dominance of each workload type is parameterised, allowing for varying levels of workload interaction. This is intended to improve the modelling accuracy by more faithfully reproducing execution interactions of general-purpose, mixed workload, applications. However, this does not exclude a single, dominant workload, it merely allows for additional execution configurations.

The micro-benchmark structure is illustrated in the pseudocode shown in Figure 4.1. The outer for loop performs a large number of iterations, controlling total execution time, while the small inner for loops are intended to execute short, sub-second, bursts of workload-specific code. This configuration of rapid changes in workload execution ensures that for any sample of performance events, taken once a second, a mix of multiple workloads will be collected. Moderate variations in workload execution times are provided by the pseudo-random numbers, which are multiplied by the assigned weight. Workload-specific weights are used to parameterise the micro-benchmark by providing a scaling factor for the dominance of a given workload.

```plaintext
for large_number_of_iterations do
    for pseudo_random_number \times fpu_weight do
        fpu_microbenchmark();
    end for
    for pseudo_random_number_{n+1} \times int_weight do
        int_microbenchmark();
    end for
    for pseudo_random_number_{n+2} \times memory_weight do
        memory_microbenchmark();
    end for
    for pseudo_random_number_{n+3} \times nop_weight do
        nop_microbenchmark();
    end for
    for pseudo_random_number_{n+4} \times cache_weight do
        cache_microbenchmark();
    end for
end for
```

Figure 4.1: Pseudocode for the micro-benchmark.
By default the weight assigned to each workload is set to one. However, the parameterised nature of the micro-benchmark allows the weight of a given workload to be adjusted independently of all other workloads. This enables the power response for each workload to be separately evaluated, which is used to derive each of the workload-specific power models. For example, to evaluate the impact of an FPU-dominant workload, the assigned weight, $FPU_{\text{weight}}$, can be increased from one to four, leaving all other workload weights unchanged. To avoid any potential synchronisation overhead, the micro-benchmark is run simultaneously on all processor cores.

4.3 Deriving a performance-based power model

The principle technique for modelling power is to determine the relationship between key performance events, the respective execution workload and the corresponding power consumption. For example, if the floating point performance event indicates a high FPU utilisation level, the current workload is likely to be dominated by floating point calculations, resulting in a high processor utilisation and a sizeable power draw. Alternatively, if a large number of cache misses occur, there may be many memory accesses. Each memory access will stall processor execution, resulting in a reduced processor utilisation, and a correspondingly low overall power consumption. By quantifying the strength of such relationships, an analytical model can be derived, enabling power to be estimated at run-time from performance event values.

To assess the relationship between performance events and workload classifications, a number of micro-benchmark iterations are required, where each iteration is run in a multitude of different workload configurations. This is achieved by adjusting the assigned weight for a given workload in isolation of all other workloads, allowing the workload-specific power characteristics to be independently evaluated. The more extensive the set of evaluated configurations, the more representative the resulting model will be. However, an exhaustive evaluation will be time consuming given the substantial number of possible configurations.

Furthermore, two additional steps are taken to isolate the results from external influences, such as system temperature, which address some of the common misconceptions in power modelling. First, the micro-benchmark is configured with a large iteration count, determined by the outer for loop, resulting in a long execution time of about 25 minutes. This ensures that each iteration reaches and sustains a stable operating temperature for the majority of execution, resolving misconceptions 1 and
2. The large data set will further help improve the correlation and regression analysis.
Second, a cool-down period is run between each iteration, where the system remains
idle so that the operating temperatures have sufficient time to return to an initial,
stable level. As a result, each iteration will experience a similar warm-up period at the
beginning of execution. Therefore, the more predictable warm-up period allows the
first couple of minutes of each execution to be trimmed from the data set, avoiding
misconception 2. It is important to remove a large portion of time, as our experimental
observations found that warm-up periods can last for 400 seconds. This further ne-
cessitates the use of a long execution time for each micro-benchmark iteration. While
some of the previous literature has attempted to mitigate warm-up effects by running
a set warm-up phase before data collection, our experimental observations found that
the required warm-up period varies between workloads, where no policy is able to pro-
vide a sufficient warm-up period with the potential of inadvertently over-warming the
processor. The experimental observations for these misconceptions will be discussed
further in Chapter 7.

A number of values are collected during each micro-benchmark execution, which in-
cludes, the performance events for each processor core, the TSC (Time Stamp Counter)
and power meter measurements. All of these values are periodically recorded and logged
to a file, i.e. once a second. The TSC is additionally collected as it provides a low-level
time value, measured as the number of cycles since last reset. This in turn allows for
the intensity values of each performance event to be calculated, providing a measure
of the number of performance events occurring within a given time period. This is
calculated by taking the difference between two adjacent events and dividing by the
elapsed time, as measured by the TSC, providing a result that is comparable between
alternate micro-benchmark configurations. For example, the execution intensity of the
processor’s FPU performance event, \( FPU_{\text{perf}} \), can be calculated as:

\[
FPU_{\text{perf}} \text{-intensity} = \frac{FPU_{\text{perf}}^2 - FPU_{\text{perf}}^1}{TSC^2 - TSC^1} \tag{4.1}
\]

Where two adjacent \( FPU_{\text{perf}} \) values (\( FPU_{\text{perf}}^1 = 178442783504931 \) and \( FPU_{\text{perf}}^2 =
178443858425259 \)) can be used, with the corresponding TSC values (\( TSC^1 = 167948366083588 \)
and \( TSC^2 = 167950859836401 \)):

\[
\frac{178443858425259 - 178442783504931}{167950859836401 - 167948366083588} = 0.431 \tag{4.2}
\]
This results in the $FPU_{\text{perf}}$ intensity for a single processor core, where the full system intensity is calculated as the sum of all processor cores for the same sampling period.

While it may be desirable to collect as many performance events as possible, this will take a prohibitively long time, given the extensive set of performance events available in modern microarchitectures, such as the AMD 10h Family processor which supports over 120 performance events (AMD, 2013). Instead, domain-specific knowledge is used to reduce the required scope of performance events to a smaller set of representative events, as was previously discussed in Section 3.4. This set of events should include, among others, various levels of the memory hierarchy, number of processor operations performed, states of key stages within the processor’s execution pipeline and the utilisation of any specialised functional units, like the FPU. Using a broad range of performance events helps to mitigate misconception 3, where the relationship between power and performance is not assumed. However, the number of performance events able to be simultaneously collected is limited by the number of processor event registers, requiring multiple micro-benchmark iterations to collect all events for a given workload configuration.

Having collected all of the required sample data, the next step is to determine which performance events have the strongest relationship with power for a given workload. This can readily be achieved by using correlation analysis to quantify the strength of relationship between each of the explanatory performance events, and the dependent power variable. For this, we use Spearman’s rank correlation to determine the relationship between the two variables, described using a monotonic function. This has the benefit of not making any assumptions, regarding the form of the relationship, which can be linear, logarithmic or exponential (Singh et al., 2009).

The subsequent analysis is intended to find a single performance event, whenever possible, that strongly correlates with the power consumption for a given workload type. For example, the $FPU_{\text{perf}}$ (intensity of the FPU performance event) has a strong linear relationship with power for an FPU-dominant workload, as shown in Figure 4.2. The x-axis is a measure of the intensity of FPU operations, calculated as the difference between two adjacent $FPU_{\text{perf}}$ values, divided by the difference in the corresponding TSC values. The y-axis is the whole system power use, measured by the power meter, for the same execution period. The modest spread of data points within each cluster is due to the micro-benchmark’s pseudo-random numbers, while the spread of clusters along the x- and y-axis is a result of variations in the $FPU_{\text{weight}}$ for different micro-benchmark iterations. From this example, it can be seen that a single performance
event, $FPU_{\text{perf}}$, is sufficient to model the variation in power for a specific workload type, FPU. Therefore, correlation analysis allows a single key performance event to be determined for each workload, which can in turn be used to model the corresponding workload-specific power.

With the performance events having the strongest correlation with power consumption, for a given workload type, the final workload-specific power models can be derived using linear least squares fitting. Linear least squares fits a regression line through the data points such that it minimises the sum of the squared differences between modelled and measured values. However the model can be impacted by outliers, so the extra step of removing points that are more than three standard deviations away from the mean should be taken first to improve accuracy. This works well for the collected data as a strong linear relationship exists between the performance events and power use, e.g. $FPU_{\text{perf}}$. However, the modelling methodology imposes no strict requirements on the form of each workload-specific model, allowing different workloads to be modelled with different performance events and regression functions.

Figure 4.2: A scatter plot illustrating the correlation between the $FPU_{\text{perf}}$ intensity and power for an FPU-dominant micro-benchmark workload.
4.4 Workload characterisation

While deriving workload-specific, linear function power models will aid overall power estimation for a diverse set of application workloads, it creates the additional requirement of accurate, run-time workload classification. While it may be expected that significantly more data is required to perform workload classification, this is not necessarily the case as the performance data previously collected for deriving the power models can additionally be used to perform workload classification. This is possible due to the tendency for individual performance events to be strongly associated with a specific workload type. For instance, at a high level, performance events such as L1 and L2-cache misses are representative of low-level cache accesses, while higher level cache misses, such as L3-cache miss, indicate workloads consisting of many memory accesses. Similarly, specific performance events can be found for each of the compute-intensive workloads, such as FPU operations for an FPU-dominant workload and retired instructions for an integer-intensive execution. Despite the presence of such relationships, the associated correlations should not be assumed (misconception 3), but derived from observations made from the data sets.

Therefore, the model training data set is used to identify the performance events that are representative of each respective workload type, allowing a distinction to be made between application workloads. To illustrate the use of performance events for workload classification, Figure 4.3 plots the linear least squares function for each workload type. The figure shows the relationship between the $FPU_{\text{perf}}$ intensity and the power for each of the five workload types. The x-axis is the $FPU_{\text{perf}}$ intensity, calculated as the difference in two adjacent $FPU_{\text{perf}}$ values, divided by the elapsed TSC. The y-axis is the whole system power measurement for each execution sample.

It can be seen in Figure 4.3 that there is a central crossing point for all workloads around $FPU_{\text{perf}}$ intensity of 1.8. This is due to all workloads starting with the same default micro-benchmark configuration, i.e., with a default weight of one for each workload type, such as $INT_{\text{weight}}$. As the weight assigned to a specific workload is changed, the points move along the axis in different directions, depending on the corresponding workload. The regression line for the FPU workload is the only one to show a strong positive correlation between $FPU_{\text{perf}}$ intensity and power while the FPU workload is dominant, i.e., $FPU_{\text{perf}} \geq 1.8$. That is, as the $FPU_{\text{weight}}$ is increasing, with all of the other weights kept at the default value of one, the relative impact of the FPU workload is increasing. The same process is followed to evaluate the impact of all other
workloads.

From a visual inspection of the spread of workloads along the x-axis, it can be seen that an FPU-dominant workload is the only one to experience a relative increase in the FPU\textsubscript{perf} intensity, i.e. $FPU_{perf} \geq 1.8$. Therefore, this can be used to define a classification threshold for an FPU workload, as all other workloads have an $FPU_{perf}$ intensity $< 1.8$. As the dominance of the non-FPU workloads increase, the proportion of each execution period spent performing floating point calculations decreases, lowering the relative $FPU_{perf}$ intensity.

While $FPU_{perf}$ is able to be used for classifying FPU-dominant workloads, it lacks sufficient detail to accurately distinguish between each of the remaining application workloads. Instead, additional performance events are required to determine all remaining classification thresholds. However, the potential exists for a performance event to strongly correlate with multiple, overlapping workloads. For instance, the retired operations performance event provides a measure of the processor utilisation, which is strongly associated with compute workloads such as INT and FPU. However, both of these workloads will have overlapping values given they both contribute to the overall processor utilisation, where the retired operations may be insufficient to accurately classify such workloads. Using a systematic classification procedure, $FPU_{perf}$ can initially be used to determine if the current workload is FPU-dominant. Following this, the FPU workload can be excluded from consideration by the retired operations clas-
sification decision, thereby allowing retired operations to determine the classification threshold for an INT-dominant workload, despite the overlapping values.

In practise, this general classification procedure allows each application workload to be independently classified, irrespective of the availability of a uniquely identifying performance event. This is in principle similar to how a decision tree may perform classification, which provides an alternate method for automated classification that can be applied to the training data.

A final, important consideration when determining the classification thresholds, is that not all of the perceived event correlations represent meaningful relationships. For instance, it was previously observed that an $FPU_{perf} < 1.8$ could conceptually be used to classify one of the non-FPU workloads. However, such an observation is misleading as there is no meaningful relationship between $FPU_{perf}$ and any other workload. The misperception arises from the micro-benchmark’s mixed workload structure, which explicitly includes all workloads during execution. General-purpose applications provide no certainty regarding the included workloads, where it is possible that some workloads, such as floating point calculations, will not be present. Deriving a classification procedure from such an indirect observation will result in inconsistent results, leading to generally erroneous conclusions. Sufficient care is required to avoid misconception 3, ensuring that each performance event has a strong positive correlation with the respective workload.

4.5 Methodology discussion

The modelling methodology has no strict requirement on the form of the model for each workload type, allowing different application workloads to be modelled with different performance events and regression functions. While a single variable linear function was used for the presented workloads, multiple variables may be used for other workloads and their regression functions. This flexibility additionally allows the methodology to work on a variety of architectures. For instance, many microarchitectures provide performance events for monitoring memory accesses, thereby allowing a memory workload to be modelled by a single variable regression function. Alternatively, an architecture without such an event can use multiple performance events to achieve the same result. Since the power model is more general and adaptable, it is more robust to architectural changes than traditional multi-variable models that use a single linear function to generalise the relationship between performance events and power use. This architectural
independence ensures the model remains usable across various system architectures and is capable of capitalising on any future architectural changes, such as new functional units or additional performance events. For example, if a new independent functional unit is added to a system architecture, its power can simply be modelled with an independent linear function using the performance events related to the unit.

The use of workload classification enables the power model to accurately estimate power for a large variety of application workload types. This is achieved by using a collection of workload-specific modelling functions, where each power model is tailored to a given workload. In addition to allowing different sets of performance events to be used for each workload-specific model, a single performance event can be used in multiple, distinct modelling functions. For instance, L2-cache misses may provide a strong measure of both cache and memory workloads, with each workload being assigned a different weight to cache misses. The use of workload classification allows a single performance event to be used in multiple models without compromise, which would not be possible using a generalised modelling function.

Furthermore, workload classification simplifies model training by only requiring performance events to be collected while each respective workload is dominant. Therefore, each regression function is applied to the dominant execution characteristics, excluding low utilisation values, which may have markedly different power characteristics. Alternatively, some of the existing literature required more complex modelling functions, such as a piece-wise linear function (Singh et al., 2009), to accurately model power for each of the execution phases. The selective use of dominant execution phases additionally helps to reduce the time required in collecting performance events, where the full breadth of alternate configurations does not need to be evaluated.

The use of workload classification additionally helps serve a broader objective of providing execution context for power use, which can be utilised by a power-aware operating system in evaluating power saving policies. This is very important because accurate power estimation is not all that is required in developing a truly energy efficient system.

4.6 Summary

In this chapter I presented a novel modelling methodology that derives a collection of workload-specific linear functions for run-time power estimation, using performance events. I then introduced the parameterised design of the micro-benchmark that re-
produces a set of representative, synthetic workloads for deriving the power model. Unlike traditional approaches which use a limited set of benchmarks, the proposed methodology uses a single parameterised micro-benchmark, made up of multiple workload kernels, to collect performance events and power values for training the power model.

In the remainder of the chapter I described the general approach taken to derive each of the workload-specific power models from the micro-benchmark data, before using the same set of performance events to determine a classification method for each respective workload. Finally, I presented a brief discussion of some of the advantages of the general modelling methodology and the adoption of workload classification. In the next chapter I will present an implementation of the power model and the accompanying experimental setup and configuration.
Chapter 5

Implementation

In Chapter 4 I presented a methodology for deriving a collection of workload-specific power models using workload classification. In this chapter I describe the experimental setup and system configuration used for evaluating the power model. Following this, I present W-Classifier, an implementation of the proposed power modelling methodology, that utilises the micro-benchmark to derive a power model and construct a general workload classification procedure.

5.1 Experimental setup

The experiments are run on a Dell PowerEdge R905 with four quad-core AMD Opteron 8380 processors, where each core has its own floating point unit (FPU). Each processor is located with 4GiB of RAM in a NUMA (Non-Uniform Memory Access) architecture, giving a system total of 16GiB of RAM. Each core has four different operating frequencies through DVFS (Dynamic Voltage and Frequency Scaling). However, the frequency is restricted to the highest (2.5 GHz) for all experiments, as it is the most commonly used frequency in practice.

The system’s power was measured using a 'Watts Up? Pro .net' power meter connected to an external monitoring system via USB, which allowed power values to be periodically collected, i.e. once a second. The power meter has a measurement accuracy of $\pm 1.5\% + 0.3$ Watts (Watts Up?, 2013). Additionally, an iSocket (InSnergy Socket)$^1$ power meter was used for supplemental data collection and verification purposes, with a measurement accuracy of 1%. The iSocket meter is connected to the monitoring

---

$^1$Institute for Information Industry, http://web.iii.org.tw/, who we thank for providing this measurement equipment.
system through a ZigBee base station, allowing run-time measurements to be collected simultaneously with the measurements from the Watts Up? power meter. The experimental system has a measured idle power of 249W with the processors operating at 2.5GHz. This is also the base (i.e. static) power of the system as no low power states, such as sleep or processor halt, are supported, making idle the default system state. During benchmark execution, a maximum power consumption of nearly 480W was achieved, illustrating the system’s high dynamic power of about 48%.

The external monitoring system was additionally configured to remotely monitor the experimental system’s temperatures, fan speeds, and component voltages. This was achieved by modifying the IPMItool² source code to enable run-time monitoring of key system events at the same rate as the power measurements. The monitoring system communicates with the experimental system over a LAN, using the IPMI (Intelligent Platform Management Interface) protocol. On the experimental system, all monitoring requests are handled by the Baseboard Management Controller (BMC), which is a special-purpose micro-controller embedded in the motherboard, that is capable of communicating with the system components without requiring any interaction with the operating system. Therefore, the experimental system does not incur any additional overhead during monitoring.

5.2 Software configuration

The power model is evaluated using the NAS Parallel Benchmark (NPB) suite (Bailey, Barszcz, Barton, Browning, Carter, Dagum, Fatoohi, Frederickson, Lasinski, Schreiber, et al., 1991), which provides a range of parallel, scientific workload kernels derived from computational fluid dynamics. The use of parallel benchmarks allows a single benchmark to be concurrently executed on all 16 processor cores, fully loading the system. Each of the benchmarks, and the corresponding problem sizes are shown in Table 5.1. The chosen problem sizes vary between each of the benchmarks as they were configured to provide a minimum execution time as close to ten minutes as possible, in order to ensure a sufficiently long evaluation period. However, this was not achievable in all cases, with some benchmarks being altered to marginally increase the problem size or iteration count, which did not impact the benchmarks workload characteristics.

All of the benchmarks are the OpenMP reference implementations from NPB 3.3, compiled with gcc-4.6.3, using OpenMP 3.0 (Board, 2008) and optimisation argu-

²http://ipmitool.sourceforge.net
ment -03, as the default configuration. In a limited number of cases, where explicitly stated, an alternative benchmark configuration was used. That is, either gcc-4.4.7, or optimisation argument -00. The micro-benchmark is compiled with gcc-4.6.3, and the optimisation argument -00. The explicit use of no compiler optimisations prevents the removal of any code which may appear to be redundant. All benchmarks are running on a standard installation of Linux, kernel version 2.6.32-25.

Table 5.1: OpenMP NAS Parallel Benchmarks\(^3\) used to evaluate the power model.

<table>
<thead>
<tr>
<th>NPB</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>B</td>
<td>Data Cube</td>
</tr>
<tr>
<td>EP</td>
<td>D</td>
<td>Embarrassingly Parallel</td>
</tr>
<tr>
<td>FT</td>
<td>D</td>
<td>Discrete 3D fast Fourier Transform, all-to-all communication</td>
</tr>
<tr>
<td>LU</td>
<td>C</td>
<td>Lower-Upper Gauss-Seidel solver</td>
</tr>
<tr>
<td>MG</td>
<td>D</td>
<td>Multi-Grid on a sequence of meshes, long- and short-distance communication, memory-intensive</td>
</tr>
<tr>
<td>SP</td>
<td>C</td>
<td>Scalar Penta-diagonal solver</td>
</tr>
<tr>
<td>UA</td>
<td>C</td>
<td>Unstructured Adaptive mesh, dynamic and irregular memory access</td>
</tr>
</tbody>
</table>

5.3 Run-time performance event monitoring

At the beginning of each monitoring period, the details for each of the chosen performance events are set in each of the performance event registers of the processor cores. This is handled by a special-purpose kernel module\(^4\) that enables the destination register for each performance event to be specified, ensuring reliable event monitoring.

Having set the desired performance events, a user-level application is used to read the current event values, once a second, from each performance register. The `perf` function is used to access the value of a given register by using the `rdpmc` assembly instruction whenever called. While the `rdpmc` instruction is not serialized, the use

---

\(^3\)Descriptions are taken from [http://www.nas.nasa.gov/publications/npb.html](http://www.nas.nasa.gov/publications/npb.html)

\(^4\)The kernel module is based on open source software written by Silas Boyd-Wickizer which was modified in collaboration with Kai-Cheung Leung.
of the performance events across multiple processor cores does not require cycle level accuracy.

```c
unsigned long perf(unsigned int c){
  unsigned int a, d;
  __asm __volatile("rdpmc" : "=a" (a), "=d" (d) : "c" (c));
  return ((unsigned long) a) | (((unsigned long) d) << 32);
}
```

The run-time monitoring is provided by calling this function for each of the performance events, once a second, by triggering a Linux timer event. For every event trigger, all four performance events and their corresponding timestamp counter value are logged to an allocated block of memory, which is written to disk once the current applications execution completes. By delaying the writing of values to disk, the potential impact of the monitoring procedure should be minimised as much as possible. The monitor additionally sleeps between event triggers in order to reduce the total time spent collecting performance values, resulting in an overhead of 11 micro-seconds for every one second sample period. The monitor application is run sequentially on each processor core, requiring 16 concurrent instances to monitor the experimental system. The event timers are synchronised at the beginning of execution, ensuring no communication overhead will be incurred at run-time.

5.4 **W-Classifier model implementation**

The implementation of the modelling methodology, W-Classifier, collected all of the data for the performance events from 16 concurrent instances of the micro-benchmark given in Figure 4.1, configured to use workload weights of 1, 2, 4, 6 and 8. These selected weights were found to provide a sufficient execution sample for determining the overall trend of power use for each respective workload type, i.e., a linear regression fit. The micro-benchmark workloads were configured with a small iteration count resulting in short bursts of execution, taking about 10 micro-seconds for the default workload weight. This ensures that for each one second sample, all of the micro-benchmark workloads are represented, with respective execution times depending on the corresponding weights. Further, implementation specific details can be found in micro-benchmark code, in AppendixB. In an attempt to mitigate the impact of thermal effects during execution, the micro-benchmark was additionally configured
with a large iteration count, resulting in execution times of about 25 minutes. Also, the first 400 seconds were trimmed from the event data logs in order to remove the effect of processor warm-up. These steps additionally help mitigate the impact of misconceptions 1 and 2.

Domain-specific knowledge, previously discussed in Section 3.4.2, was used to select an initial set of 13 performance events, shown in Table 5.2. Each of the performance events and the power meter measurements were logged to a file once every second during execution. This specific set of performance events was chosen for evaluation, as they provide a representative sample of system performance events for the AMD multicore machine, ranging from memory accesses to the various levels of the cache hierarchy to processor utilisation, avoiding misconception 3.

Table 5.2: Processor performance events polled within our experiments.

<table>
<thead>
<tr>
<th>1: FPU</th>
<th>2: Data cache miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>3: dispatch stalls</td>
<td>4: Instruction cache miss</td>
</tr>
<tr>
<td>5: CPU clocks not halted</td>
<td>6: L2-cache miss</td>
</tr>
<tr>
<td>7: Retired Move opt</td>
<td>8: L3-cache miss</td>
</tr>
<tr>
<td>9: Prefetch Instructions Dispatched</td>
<td>10: DTLB miss</td>
</tr>
<tr>
<td>11: Retired µops</td>
<td>12: DRAM access</td>
</tr>
<tr>
<td>13: Retired Branch Instructions</td>
<td></td>
</tr>
</tbody>
</table>

After collecting all of the event data, the strength of relationship between the workload type and performance event intensity is analysed with Spearman’s rank correlation. Further insights were gained by visualising the relationship on a scatter plot. For instance, the previous plot of an FPU-dominant workload, shown in Figure 4.2, illustrated the existence of a strong linear relationship between the $FPU_{perf}$ intensity and the power for an FPU-dominant workload. Similarly strong relationships were found for other performance events, such as retired-µops for INT-dominant workloads and dispatch-stalls for cache-dominant workloads, shown in Figures 5.1a and 5.1b respectively. It can also be concluded from the strength of these relationships that a single performance event is able to represent the power use for a dominant workload. In analysing the remaining performance events and workload types, similar linear relationships were found for each workload type, although not all relationships were as expected. For example, it was surprising to find that memory related performance events, such as L2-cache-misses, did not show a meaningful relationship with memory-
intensive workloads (misconception 3), a finding that will be discussed in detail in Section 7.3. This anomaly is mentioned here as it helps illustrate the importance of selecting a wide range of performance events, including events other than those expected to correlate well, when initially evaluating performance events for the power model.

Based on the strength of the determined relationships, the four performance events to be used in the power model are: 

- **FPU**—the number of cycles in which at least one FPU operation is present in the FPU.

- **Dispatch stalls**—the number of processor cycles where the decoder is stalled for any reason (i.e., it has one or more instructions ready but cannot dispatch them due to resource limitations in execution).

- **CPU clocks not halted**—the number of clocks that the CPU is not in a halted state (due to STPCLK or a HLT instruction).

- **Retired \( \mu \text{ops} \)**—the number of micro-operations retired. This includes all processor activity (instructions, exceptions, interrupts, etc.).

Only four performance events were chosen as this is the architectural limit for the number of events able to be read simultaneously on the experimental system. Having selected the performance events, each workload-specific linear function is derived by applying linear least squares fitting to the appropriate performance-power data sets for the specific workload. Since each workload uses a different linear function for power

---

5Performance event descriptions are taken from the AMD BIOS and Kernel Developers Guide (AMD, 2013)
estimation, a decision has to be made regarding which workload the performance values represent once the performance values are collected during power estimation. Given that each of the selected performance events is usually the primary descriptor of a specific workload type, it can be used as an indicator of the corresponding workload type. For example, the FPU performance event can be used to represent FPU-dominant workload, the performance event for retired μops can be used to represent INT-dominant workloads, and dispatch stalls can suggest cache-dominant workload.

The threshold, unique to a specific workload, can be determined by identifying the boundary point within each data set, indicating the point of separation between the dominant workload and all other workloads. For example, the threshold for an FPU-dominant workload can be identified as any $FPU_{perf} \geq 1.8$, as was previously shown in Figure 4.3. In this case, a clear point of separation between the dominant workload to be classified and all other workloads can readily be identified with the collected $FPU_{perf}$. Alternatively, for a cache-dominant workload, the boundary threshold will be determined by the maximum $cache_{perf}$ for the workloads. Any $cache_{perf}$ values greater than this threshold will indicate a cache/memory-dominant workload. Applying this general procedure to each workload results in a set of thresholds. For example, the thresholds were determined as $T_{FPU}$ (1.8), $T_{cache}$ (10.0), and $T_{INT}$ (1.0) by the boundary points within each data set, used in Figure 5.2. They can be used to classify FPU-dominant, cache-dominant, and INT-dominant workloads respectively. They can also be used combinatorially to represent mixed workload such as both INT-dominant and cache-dominant.

The general threshold determination procedure outlined above enables a rudimentary workload classification algorithm to be derived. The pseudo-code for the derived classification algorithm is presented in Figure 5.2. A detailed implementation of the classification procedure is presented in Appendix B.2. The classification implementation includes two mixed workloads, FPU/cache and INT/cache, which include the effect of cache on FPU or INT operations using the collected micro-benchmark data.

In future implementations, the selection of threshold values may be aided by the use of machine learning, such as support vector machines or decision trees, for the classification algorithm. These techniques will likely be required when processing significantly larger data sets that may be caused by increases in the number of performance events or types of workload. The current implementation does not suffer from these restrictions, as a limited set of performance events are used to represent a limited set of workloads. This allows a manual selection of threshold values from the input data.
if \( \text{cache}_{\text{perf}} > T_{\text{cache}} \) then
  if \( \text{FPU}_{\text{perf}} > T_{\text{FPU}} \) then
    mixed FPU and cache workload
  else if \( \text{INT}_{\text{perf}} > T_{\text{INT}} \) then
    mixed INT and cache workload
  end if
else if \( \text{FPU}_{\text{perf}} > T_{\text{FPU}} \) then
  FPU workload
else if \( \text{INT}_{\text{perf}} > T_{\text{INT}} \) then
  INT workload
else
  memory/idle workload
end if

Figure 5.2: Pseudocode for workload classification.

sets. It also allows insights to be gained which would not have been possible with a black box approach like a blind multi-variable linear regression. Despite these details being implementation specific, they do not impact on the applicability of the proposed general modelling methodology.

5.5 Summary

In this chapter I described the experimental setup used while deriving and evaluating the W-Classifier power model. I then presented the implementation of W-Classifier and justified why the four selected performance events are: FPU operations, dispatch stalls, CPU clocks not halted, and retired \( \mu \)ops. This set of performance events was then used to derive each workload-specific linear function by applying linear least squares fitting to the data set. Finally, I described the workload classification procedure and the resulting performance event boundary thresholds. In the next chapter I will evaluate W-Classifier’s performance on the NAS Parallel Benchmark suite.
Chapter 6

Evaluation of the Power Model

In this chapter I evaluate the W-Classifier power model by comparing the modelling accuracy with a generalised multi-variable regression model, showing the advantages of workload classification. I then explore the impact of thermal effects on modelling accuracy, describing some limitations of the statistical methods commonly used for model evaluation. Following this, I describe some possible sources of modelling errors and discuss an acceptable threshold for estimation errors in a model of power use. Finally, I present some potential use-cases of power models.

6.1 W-Classifier evaluation

W-Classifier is evaluated by running each of the OpenMP multi-threaded benchmarks from the NAS Parallel Benchmark (NPB) suite on all 16 processor cores. When each benchmark is running, W-Classifier is used to estimate power use for the entire system while the power meter measures the actual power use for comparison. For further evaluation, W-Classifier is compared to a multi-variable linear regression power model, derived from the same set of micro-benchmark performance events, previously shown in Section 5.4. As a result, the multi-variable power model uses the same data set as W-Classifier, except that workload classification is not used by the power model.

The model is derived by using a multi-variable, linear least squares regression function on the performance event data for the FPU, INT and cache workloads. This subset of workloads was used as it provided a similar workload to some previous research (Da Costa and Hlavacs, 2010) and provided better regression results than using all workloads. The CPU-clocks-not-halted performance event was removed from the model as it had a coefficient of zero. In the derived model, retired-\(\mu\)ops was the
strongest explanatory event, while FPU and dispatch-stalls accounted for less of the power variation in the data sets. Similar results were found in previous work (Bircher et al., 2005).

First, Figure 6.1 shows the mean power estimations for each benchmark by W-Classifier and the multi-variable model, together with the mean measured power. The mean power of each benchmark, taken as the average of the one second samples measured during the whole execution period, provide a coarse-grained metric of power estimation accuracy. It can be seen in the figure that W-Classifier is able to track real power use reasonably well. Compared with the multi-variable model, W-Classifier is more accurate in terms of mean power estimation for most benchmarks. Note that the mean power estimation of the multi-variable model for SP is out of the scale of the figure, at > 600W. Further details for the range of power estimates can be found in appendix C.

Figure 6.2 shows the mean error and Mean Absolute Error (MAE) of power estimation for both W-Classifier and the multi-variable model. MAE measures the absolute difference in the estimated and measured power for every data point, e.g. every second. It indicates how well the trend of the estimates follows that of the real power use measured by the power meter. An important objective of a power model is the ability to accurately track run-time power, which requires the trend of estimated power to closely match the measured power use. Therefore, MAE is used for model evaluation.
as it provides an indication of data trend. In contrast, the mean error simply measures the difference in mean power for the entire execution, where mean power is the average power estimate for each of the one second execution samples. However, such a measure provides no indication of how closely the estimates follow the real power consumption. It is conceivable that a low mean error can be achieved by a model that wildly over-estimates and under-estimates power, but the large positive and negative errors cancel out each other in the mean error. In these situations, however, MAE would return a more meaningful result with a large error.

According to Figure 6.2, W-Classifer has an average mean error of 5.78% for all benchmarks while the multi-variable model has an average mean error of 39.94%. W-Classifer has a much smaller mean error than the multi-variable model for most benchmarks. The only exceptions are EP and FT where the multi-variable model is slightly better in terms of mean error and mean power estimation, though the absolute differences between them are very small in terms of Watts. The slightly larger mean errors of W-Classifer for benchmarks EP and FT, compared with the multi-variable model, can mainly be attributed to the errors of base power estimation in W-Classifer, which affects the power estimation of W-Classifer for all benchmarks to a different degree. One of the possible factors contributing to this is the varied impact of temperature on the micro-benchmark and the NPB benchmarks, which will be discussed further in Section 7.2.
In Figure 6.2, W-Classifer has an average MAE of 6.95% for all benchmarks, while the multi-variable model has an average MAE of 40.74%. However, the average MAE for the multi-variable model is negatively impacted by the significant errors of 55.6% and 188.6% for the DC and SP benchmarks respectively. When these two benchmarks are excluded from the results, W-Classifer has an average MAE of 6.69%, while the multi-variable model has a significantly improved average MAE of 8.18%. W-Classifer has a smaller MAE than the multi-variable model for all benchmarks except EP. This demonstrates that W-Classifer is better able to track run-time changes in power consumption during the execution of each benchmark.

From the figure, it can also be seen that there is a large difference between mean error and MAE for W-Classifer for benchmarks like DC and FT. This difference is caused by large variations in real power use during the execution of DC and FT, with standard deviations of 41.1W and 32.2W respectively. All other benchmarks exhibit more constant power use during execution, with standard deviations ranging from 2.3–8.5W for real power use.

The overall trend between estimated and measured power values can be more easily understood by visualising the raw data points on a scatter plot. For example, Figure 6.3 plots the power values for the DC benchmark, which had a high standard deviation, caused by large variations in power during execution. The x-axis is the execution time in seconds, while the y-axis is the corresponding power in Watts. Now it is possible to easily recognise that the trend of power estimates made by W-Classifer closely follows that of the measured power use. However, the estimates are consistently about 30W below measured values, indicating the model is currently not capturing the right value for the base power, as mentioned previously. The overall strength of the trend supports the current selection of performance events for the execution workloads.

It can clearly be seen in Figure 6.3 that the multi-variable model significantly over-estimates power consumption. This is because the DC benchmark largely consists of a memory-dominant workload, where the execution time is dominated by cache-misses and memory accesses. This results in significantly lower processor utilisation levels than most of the other NPB benchmarks. For instance, the median retired-\(\mu\)ops for FT is 22 times higher than the DC median value, indicating the scale of this difference. Since retired-\(\mu\)ops is the dominant performance event used in the resulting multi-variable model, it contains a bias towards high utilisation workloads and is therefore not well suited to such memory-dominant workloads. In contrast, W-Classifer detects two separate workloads due to workload classification and uses a
different linear function for each type of workload. For some periods of time when the utilisation level is high (with power around 400W), a FPU/cache workload classification is used for power estimation. For other time periods when the utilisation level is much lower, where the workload is classified as being memory-dominant, a different power estimation function is used.

Overall, W-Classifier can better track run-time power changes than the multi-variable model. This is reflected in the smaller average MAE across all benchmarks, which is 6.95%. While the multi-variable model is able to estimate power with an average MAE of only 8.18% for the more compute-intensive benchmarks, the error significantly increases to 40.74% after the inclusion of DC and SP. This shows W-Classifier has the distinct advantage of being better able to estimate power for a broader range of application workloads. This will prove to be an important feature when estimating power for a wider range of more general applications, as they are more likely to have varying execution phases and workloads than the NPBs.

### 6.2 Considering thermal effects

While it is well known that temperature has an effect on processor power usage, it remains less well known what steps should be taken to mitigate some of the impact.
Therefore, the proposed power model adopted a range of techniques and good practices, that were highlighted in the modelling methodology and are discussed further in Section 7.2.

The most notable technique is the use of long execution times to ensure the processor reaches a stable operating temperature. For instance, the micro-benchmark was configured to run for about 25 minutes. While this may seem excessive, it was found that the processor may take up to 6 minutes to reach a stable temperature during micro-benchmark execution. The longer the execution continues after reaching a stable temperature, the less weight the warm-up period will have on power modelling. This technique was additionally applied to the NPB benchmarks shown in Table 5.1, where each benchmark was configured with a problem size that provided an execution time of about 10 minutes, ensuring a sufficiently long evaluation period.

An alternative to increasing execution times to mitigate the effects of processor warm-up, is to merely trim all data points from the beginning of a data set that are affected by the warm-up period. A comparison of these two approaches is made in Figure 6.4, which plots the MAE for W-Classifier run on the two alternative configurations of NPBs. The results for long executions of the benchmarks are given by ‘Including thermal effect’, while the results for ‘Excluding thermal effect’ have had the first half of the respective sample data set removed. The most noticeable variation in MAE is for the DC benchmark, where the error for warm-up removal is much higher than for long execution times. However, this variation in estimation error can be attributed to the characteristics of the benchmark. Looking at the scatter plot for DC in Figure 6.3, it can be seen that towards the end of each execution, the power values begin to fluctuate frequently, resulting in a significantly noisier data set once the warm-up period is removed. Therefore, neither technique provides a significant advantage over the other in helping to improve power estimation.

The techniques discussed so far are only capable of affecting the dynamic temperature changes from processor warm-up. They do not address the potentially more significant impact of differences between steady-state operating temperatures. Such a difference can be seen in Figure 6.5, which plots the temperatures (in degrees Celsius) measured while executing the DC benchmark and micro-benchmark with a memory-dominant workload. The observed difference in the measured temperatures (about 12 degrees) is quite significant, where the average temperature for DC is two thirds of the micro-benchmark’s maximum.

Such a significant difference in temperature between two similar workloads in DC
and the micro-benchmark has unquestionably impacted on the accuracy of the derived power model, W-Classifier. Because the sampled power values for the micro-benchmark were lower than the real power use of DC for a similar workload, due to the temperature difference, the estimated power of DC by W-Classifier is no doubt lower than the measured real power, as shown in Figure 6.3. This highlights the necessity of incorporating temperature values into the power model, which can help correct the variation between the measured and estimated power by W-Classifier. However, temperature values were not incorporated into W-Classifier as the experimental systems’ embedded sensors do not provide the fine-grained and reliable temperature data required for modelling. Any difference in operating temperature between the training micro-benchmark and evaluated benchmarks will contribute some error to the power estimation. This temperature difference partly contributed to the errors of the base power estimation in W-Classifier, as mentioned previously.

### 6.3 Principles of model evaluation

In evaluating modelling accuracy, previous work has adopted a number of evaluation metrics that do not allow direct comparisons to be made between different modelling methodologies. Therefore, additional care is required when making such comparisons, in order to avoid presenting misleading or erroneous conclusions.
6.3.1 Static versus dynamic power

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

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

The potential impact of dynamic power on modelling accuracy is further illustrated in Figure 6.6, which plots the estimation error for various system dynamic power levels for the previous example. From the figure it can be seen that a low dynamic power can result in errors as high as 50% for dynamic power. Therefore, without this additional contextual information, it is not possible to make a fair, unbiased comparison between different methodologies, derived and evaluated on two separate systems, with different power characteristics.

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

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

Furthermore, it is not strictly necessary to present all evaluation results relative to dynamic power. Instead, clearly specifying a system’s power characteristics may be sufficient to provide some additional context for model evaluation, as was done in the previous example and Section 5.1.

6.3.2 Statistical evaluation

Summary statistics like the median error are commonly used to present the evaluation error of power models, but this is not sufficient to truly represent the closeness of fit for the model and can lead to a misrepresentation of some results. In the most naïve cases, the absolute value for the measured error is not taken, leading to a significantly overstated accuracy of fitting. This is due to the tendency for the positive errors of over-fitting to cancel out the corresponding negative values of under-fitting or vice versa.

However, even if the absolute values of errors are used, the median error is not descriptive enough to give an adequate representation of the actual fitting. This is in a large part due to the type of benchmarks used for model evaluation, scientific workloads,
and the structure of distinct workload phases. This is illustrated in Figure 6.7, which plots power use of the DC benchmark during execution. The x-axis gives execution time in seconds, where the power is measured once a second, which is then shown on the y-axis. It can be seen that DC has two distinct workload phases. The first is a dominant, memory-bound workload, with low levels of power consumption: around 260W. The other workload is a periodic computation-intensive phase represented by the sudden, short spikes in power use. There is an additional horizontal line in the graph at 262.9W, which represents the median measured power value. From this, it can be concluded that the dominant workload is the memory phases, because the median value is down near the bottom of the graph. Therefore, at least half of the time is spent executing this lower power phase.

As a result, a power model will achieve a low median error overall if it is able to closely estimate the dominant execution phase, memory accesses. This in no way provides any indication that the estimation model is able to accurately estimate the less common, computational phases of execution. In fact, it does not have to do so in previous research in order to achieve a low median error.

A better metric to be used for evaluating the power estimation model is the Mean Absolute Error (MAE), as was used in Section 6.1, which calculates the absolute error for every power measurement before taking the average of all of these values. As opposed to the median, the mean will be influenced by outliers, meaning that an inability to model the changes in workload phases will result in larger errors. While this provides an improvement over median error, a single metric cannot be guaranteed to provide a fair comparison for the quality of fitting in all situations. Therefore, when further analysis is required, the standard deviation can be used to provide an illustration of how widely spread the errors are. This can give further insight into the accuracy of the model without requiring a plot of absolute values.

Therefore, sufficient care is required in evaluating modelling accuracy to ensure fair comparisons are made between different modelling methodologies, where evaluation metrics are currently used inconsistently in some of the previous literature.

### 6.4 Evaluating modelling accuracy

While it is desirable to achieve perfect power estimation, such an objective may be unobtainable in practice. Such attempts may incur significant run-time overhead, where a less accurate, but more efficient power model, may prove to be sufficient for many use-cases.
6.4.1 Measuring estimation error

In modelling processor power consumption, a tradeoff between model complexity and accuracy is required. This problem is exacerbated by the increasing complexity of newer microarchitectures, which can be attributed to the incorporation of: hardware threading, fine-grained power saving modes, and increasing component densities. Therefore, estimation models must approximate many of these power factors, balancing run-time performance monitoring and model complexity with estimation accuracy. Collecting a large number of performance events, and modelling all of the corresponding interactions reaches a point where it becomes more akin to simulation, with gains in accuracy being outweighed by the run-time overhead.

Fortunately, if appropriate care is taken while deriving the power model, many sources of approximation can be controlled for, or incorporated into the model. For instance, the software configuration, such as compiler version or optimisation level, will impact the power characteristics of a given benchmark. Such variations could be incorporated into the model by treating alternative compiler/benchmark configurations as separate benchmarks used during training, instead of a single benchmark run multiple times. Temperature is another commonly considered factor influencing power consumption, which can be difficult to reliably measure. This is primarily due to the poor quality of standard measurement equipment and significant latencies in observable thermal effects within a system. A simple, but apparently rarely considered, approach to mitigating some of the impact is to ensure that execution times are sufficiently long so that a stable operating temperature is reached before data collection. Short, periodic executions can lead to below-average power use for a given workload, as the temperature will be below the typical operating point given insufficient time for processor warm-up. These two factors, among others, illustrate the importance of taking care in setting up a model’s experimental configuration.

However, some sources of estimation error are beyond measurement and are unable to be incorporated into the model. The most common source of these errors is hardware power saving features that operate independently of the operating system, and are therefore transparent to the system. The best example of this is the anecdotal evidence of aggressive clock gating in modern processors (McCullough et al., 2011), where functional units within the processor are shut off while inactive, in order to save power. Given the lack of understanding as to how this feature operates, it is even difficult to trigger the occurrence of certain actions during model training in an attempt to incorporate the respective power variations. A further source of inconsistent
hardware power measurements may be due to manufacturing variability. McCullough et al. (2011) observed an 11% difference in power use between two Intel Core i5-540M processors, in identical configurations. This can exacerbate estimation errors where a model is trained on one processor and is used on other processors which are thought to be identical. This illustrates the unlikely nature of eliminating all sources of estimation error. Therefore, the question should be, what is an acceptable estimation error?

6.4.2 Identifying an acceptable modelling error

There is no single criteria for determining if a given power estimation model is accurate enough, as each use-case will have different tolerances of estimation errors. For instance, using the power model to evaluate how best to consolidate workloads in a data center has a rather high tolerance, suggested to be 5–10% by McCullough et al. (2011), due to the coarse-grained nature of per-server power estimation. In contrast, if the same data center were to use this model to charge clients for the power each submitted job uses during execution, an error this high for the entire system would not be tolerated. Such a high error would possibly lead to charging clients for resources not actually used.

The most commonly proposed use for power models is in making power-aware task schedulers. Power-aware task scheduling is a rather generic term, giving rise to many different scheduling policies. For the discussion here, we define “power-aware task scheduling” as using a scheduler that prioritises resource allocation to the most power efficient uses, while maintaining all strict resource requirements and workload deadlines. For instance, to determine which task should be allocated additional processor cores, the power model can be used to determine the change in power in response to a given allocation. The power efficiency for each task is then evaluated as the size of the relative power change, thereby indicating which task will use the least additional power. A fair evaluation has two accuracy requirements. First, the estimation error needs to remain stable across workloads to prevent any bias in evaluation. Second, the estimation error needs to be less than the power difference of any two cores, otherwise the error may hide the real change in power. These requirements mean the allowable estimation error will be system-specific, as different architectures will have processors with significantly varying operating ranges for power use.

Unfortunately, this means that no single rule exists for determining a specific threshold for modelling accuracy. Therefore, a power model with a 10% power estimation error will be usable for many use-cases, despite the perceived inaccuracy. However, the simple fact remains, the greater the estimation accuracy, the better.
6.5 Power model use-cases

The previous section discussed in general terms what an acceptable model accuracy would be. However, it was concluded that this will often depend upon the eventual use-case of the power model. Therefore, to provide some context, this section presents a selection of different use-cases.

6.5.1 Improved user feedback for effective power management

A common problem in many widely deployed systems is the use of coarse-grained, whole system, power management policies that often result in overzealous power savings, sacrificing run-time performance. Consequently, users commonly disable such power management policies in order to restore the desired performance level, negating all potential power savings. Therefore, fine-grained power measurements enable more effective power management policies, enacted on a per-component or per-application basis. Moreover the detailed power measurements can improve user feedback, supporting additional user configurability of power management policies in achieving the desired balance between power and performance.

Without sufficient feedback, the user will be incapable of adequately evaluating the tradeoff between power consumption and execution performance for different configurations. This will result in overly conservative policies due to the common user preference for system performance. A challenge in evaluating such power policies arises from the disconnect between the immediate change in performance and the corresponding impact of power savings for a given configuration selection (Turner, 2013). That is, taking action now, such as lowering screen brightness, directly impacts the user while providing no immediate benefits. Instead such actions are taken for a future benefit (Turner, 2013). However, evaluating the tradeoff will be difficult if the potential benefits cannot be quantified and presented to the user when selecting the management policy.

User feedback can additionally be used to provide insights into long-run power consumption, such as the impact of leaving systems powered on overnight. Without easily interpretable power consumption statistics, users will be incapable of making proactive decisions on saving power, instead, opting to maintain the default configuration. Moreover, the perceived complexity of changing the configuration will determine the willingness of a user to alter the current configuration. For instance, a large number of sliders and fine-grained controls will be daunting to most users. Alternatively, initially presenting a user with a set of predefined user profiles, with the additional option of
advanced controls, will be more usable. Such an approach was adopted by the FatBatt\textsuperscript{1} battery analysis tool for laptops, by providing simplified user controls and accompanying power feedback. This enabled the power consumption of specific applications to be tracked over time, aiding the user in determining the application-specific power policies to be enacted.

While portable consumer electronics currently provide coarse-grained power details, such as remaining battery power, these metrics are incapable of providing meaningful feedback to users regarding the sources of power consumption. In response, operating systems are beginning to incorporate additional methods of reporting detailed power metrics to users. For example, OS X Mavericks\textsuperscript{2} has added the energy impact factor for each application, determined as a measure of system utilisation, which is unfortunately not a direct measure of power use. The forthcoming ‘L’ release of Android will include the battery historian\textsuperscript{3} power analysis tool, which will provide the ability to plot a time series of the battery discharge on a per-application basis.

Improved feedback of power consumption allows the user to make informed and adequately evaluated decisions when selecting the most appropriate power management policy. It is only with a configurable policy and a truly informed selection procedure that effective power management can be widely achieved.

6.5.2 Increase system deployment efficiency

Data center deployment densities are typically determined by the rated maximum power (nameplate value) of a given server (Giri, 2010), where the number of servers deployed within a rack is determined by the limit of the power supply. This will result in a potentially sizeable power buffer, as the servers typically operate below their rated maximum power consumption. Such power buffers ensure sufficient spare capacity is available to safely handle any sudden spikes in system resource utilisation and the corresponding rise in power consumption.

Alternatively, this power headroom can be leveraged to increase the hardware deployment density of a data center, where the sum of system components peak power consumption exceeds the data center’s total power budget (Barroso and Hölzle, 2009).\textsuperscript{4}

\textsuperscript{1}FatBatt was produced by MiserWare, which has subsequently stopped trading and shutdown all services \url{http://www.fatbatt.com/}

\textsuperscript{2}\url{http://support.apple.com/kb/HT5873}

\textsuperscript{3}A preview of the Battery Historian tool is described in the developer API overview \url{https://developer.android.com/preview/api-overview.html}
The over-provisioning of resources is possible due to the tendency for execution power to not reach the corresponding peak power. For instance, Fan et al. (2007) observed that for a large cluster, consisting of 5,000 machines, the actual power use never exceeded 72% of peak power while executing a mixed workload. This indicates the headroom in power that can be utilised by additional compute nodes. This is based upon the peak power observed for a single node, which was found to be less than 60% of the nameplate power, specified by the manufacturer as a worst case power level for each component to be handled by the power supply.

Software power management policies can be used to enforce power caps within a system or data center in the rare occurrence of power demand exceeding the available power budget. This allows for resource over-provisioning while under normal operating conditions, and additionally prevents any damage from occurring during any unusual spikes in demand. The increase in deployment densities will depend on whether the measured peak power or nameplate power are used to determine the required power infrastructure. For a deployment based upon peak power, capacity can increase by 40%. Alternatively, the overestimation of nameplate values provides significantly more spare power capacity allowing for 80–130% more machines to be deployed (Fan et al., 2007).

6.5.3 Meeting the exascale challenge

Power consumption has become a primary constraint on the scale of high-performance computer deployments, where the lifetime operating costs are now equivalent to the purchase cost. The impact of power on the design of future large-scale systems was first introduced in Chapter 1 with a discussion of the challenges facing the development of an exascale supercomputer. However, like other work (Castro, Francesquini, Nguélé, and Méhaut, 2013), the initial calculation of 50 gigaFLOPS/W for the minimum power efficiency is based upon some simplifying assumptions. A practical exascale supercomputer requires a significantly higher minimum efficiency to overcome the power constraint on performance.

By relaxing the key assumption that the full 20MW power budget is solely consumed by the compute resources, the required efficiency significantly increases as the allowable power use decreases. For instance, if the compute resources (processors) consume 50% of system power, and the data center has a power usage effectiveness (PUE) of 1.25, a minimum efficiency of 125 gigaFLOPS/W is required. Where the PUE provides a measure of the data center’s efficiency, calculated by dividing the total data center
power by the total power for the compute equipment (Azevedo, French, and Power, 2012). Taking other sources of power consumption into account significantly increases the required power efficiency to achieve an exascale system deployment.

A commonly proposed alternative to high power/performance processors is to use low-power embedded ARM processors. The Mont-Blanc research project has developed a prototype blade consisting of 15 Cortex-A15 compute boards (Ramirez, 2014). Despite the low power consumption, at scale, significant power may still be used. For example, Table 6.1 presents the hardware specification details for power and performance of a variety of processors and GPGPUs (General Purpose Graphics Processing Units). For illustrative purposes, the table additionally presents the number of each respective processor to achieve a theoretical exaFLOP and the corresponding power consumption. It can be seen in Table 6.1 that even the Raspberry Pi, which is the most power efficient compute resource, requires an impractical scale of deployment. The 41 million Raspberry Pis would have a peak power consumption of 144 megaWatts in achieving a theoretical exaFLOP. Despite the high power consumption of the NVIDIA GPUs, they are able to achieve a higher overall efficiency than commodity server processors, such as the AMD Opteron.

From these observations, it may be concluded that no single modern architecture has a distinct advantage over all others, where advances in hardware alone may be insufficient to meet the exascale challenge. Instead, power management is set to play an increasingly diverse role in managing power consumption. For instance, the United States Department of Energy (DOE) are driving advances in the management of system resources in addition to the development of more efficient hardware (Harrod, 2012). Power models are able to play a key role in the development and evaluation of such management techniques, which includes:

System middleware: Power-aware middleware can dynamically manage system resource utilisation at run-time, leveraging hardware power saving mechanisms to more effectively manage power consumption.

Data management: Improving data locality awareness can help reduce power consumption by minimising data movements.

Scalable software: Creating scalable, power-aware algorithms.

Power optimisation: Developing techniques for improved software power analysis

\footnotetext{All processor details were taken from the corresponding data sheets}
aiding software development, that will potentially enable compiler power optimisations.

**Workload allocation:** Enabling a broad view to be taken in managing data center resources with load balancing.

The variety of different sources of power improvements and analysis provides a broader perspective of data center power consumption. This importantly allows cross component power effects to be considered, where a change in execution power for one system component will have knock-on effects for other components (Carter and Rajamani, 2010). For example, raising data center ambient temperature may increase the work of server level fans, resulting in higher overall power use. Therefore, a diverse perspective of power consumption is required for effective power management, necessitating the use of a general and adaptable power model.

### 6.6 Summary

In this chapter I evaluated the W-Classifier power model on the NAS Parallel Benchmark suite, comparing it to a typical multi-variable power model. W-Classifier was able to better track changes in run-time power, which was reflected in the smaller average MAE across all benchmarks at 6.95%. In comparison, the multi-variable model had an average MAE of 8.18% for the compute-intensive benchmarks, and a significantly higher average MAE of 40.74% with the inclusion of the DC and SP benchmarks.

I then discussed the common misconception of making direct comparisons between different modelling results, where each model uses different statistical evaluation methods and system configurations. Finally, I presented a selection of non-scheduling use-cases for power models, emphasising the requirements of a general and adaptable power modelling methodology. In the next chapter I will discuss the misconceptions that impact power model accuracy and subsequent model evaluation.
Table 6.1: Power and performance values for a selection of processors used to evaluate the power efficiency and the scale required to calculate a theoretical exaFLOP with peak performance.

<table>
<thead>
<tr>
<th></th>
<th>GFLOPS</th>
<th>Thermal Dynamic Power (watts)</th>
<th>Number required for exascale (million)</th>
<th>Total Power (MW)</th>
<th>GFLOPS/W</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Embedded processors</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raspberry Pi</td>
<td>24.3</td>
<td>3.5</td>
<td>41</td>
<td>144</td>
<td>6.94</td>
</tr>
<tr>
<td>Samsung Exynos 5</td>
<td>32.3</td>
<td>10</td>
<td>31</td>
<td>310</td>
<td>3.23</td>
</tr>
<tr>
<td>Mont-blanc blade</td>
<td>485</td>
<td>175</td>
<td>2</td>
<td>361</td>
<td>2.77</td>
</tr>
<tr>
<td><strong>Commodity processors</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Xeon</td>
<td>185.6</td>
<td>135</td>
<td>5.4</td>
<td>727</td>
<td>1.37</td>
</tr>
<tr>
<td>AMD Opteron</td>
<td>180</td>
<td>140</td>
<td>5.5</td>
<td>778</td>
<td>1.28</td>
</tr>
<tr>
<td><strong>GPGPU accelerators</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVIDIA Tesla K40</td>
<td>1430</td>
<td>245</td>
<td>0.7</td>
<td>171</td>
<td>5.83</td>
</tr>
<tr>
<td>NVIDIA Tesla K20</td>
<td>1170</td>
<td>225</td>
<td>0.8</td>
<td>192</td>
<td>5.0</td>
</tr>
<tr>
<td>Intel Xeon Phi</td>
<td>1208</td>
<td>300</td>
<td>0.8</td>
<td>248</td>
<td>4.02</td>
</tr>
<tr>
<td>AMD FirePro S10000</td>
<td>1478.4</td>
<td>375</td>
<td>0.7</td>
<td>254</td>
<td>3.94</td>
</tr>
<tr>
<td>AMD FirePro S7000</td>
<td>152</td>
<td>150</td>
<td>6.5</td>
<td>987</td>
<td>1.01</td>
</tr>
</tbody>
</table>
Chapter 7

Improvements in power modelling

In this chapter I present a discussion of the common misconceptions found in previous power modelling methodologies, that were highlighted throughout this dissertation. The discussion regarding each misconception is based upon experimental observations made during the thorough analysis of each step of the development and evaluation of W-Classifier.

The modelling accuracy of W-Classifier was improved by addressing each of these misconceptions, thereby mitigating various sources of modelling inaccuracy. Each Section discusses a separate misconception, beginning with those impacting modelling accuracy:

- Sampling rate and execution time can be left unspecified—Section 7.1
- Thermal effects are insubstantial—Section 7.2
- Memory events correlate well with power consumption—Section 7.3

In addition to impacting modelling accuracy, a number of misconceptions have a detrimental impact on model evaluation. These misconceptions include:

- Compilation configuration does not need to be reported—Section 7.4
- Experimental setup can remain unspecified—Section 7.5

The experimental observations discussed in this chapter have informed the good practises adopted in the modelling methodology implementation and evaluation presented in this dissertation.
7.1 Impact of sample rate and execution time

While performance event sample rates and benchmark execution times are often left unspecified in much of the existing literature, changes in such execution parameters will have a potentially detrimental impact on modelling accuracy. The use of either coarse-grained samples or short execution times will have a direct impact on the strength of correlation and the noise within a data set. This may result in erroneous conclusions being drawn, where the relationship between power and performance is mischaracterised.

7.1.1 Various sample rates

The first step in deriving a power model will consist of collecting some preliminary insights into key performance events for a range of application workloads. For this, monitoring tools such as perf (2014) are commonly used to collect aggregate performance events over the execution period, providing coarse-grained performance values. This is equivalent to sampling the performance events once at the beginning and again upon completion of each micro-benchmark execution.

For example, Figure 7.1 presents the aggregate performance values for multiple iterations of the micro-benchmark, introduced in Section 4.2, configured with an FPU-dominant workload. The x-axis is a measure of the intensity of FPU operations, calculated as the difference between two adjacent FPU\_perf values, divided by the difference in the corresponding TSC values. The y-axis is the calculated average power use, measured by the power meter, over the same execution period. In this example, the micro-benchmark’s FPU\_weight, previously shown in Figure 4.1, is adjusted for different iterations, providing the spread of data clusters along the x- and y-axis. The data points form a series of tight clusters along a linear path, indicating a strong correlation between power and performance, where power use increases in response to a corresponding change in FPU\_perf intensity.

From the initial coarse-grained values, it is evident that a relationship exists between FPU\_perf intensity and power use. To gain further insights into this relationship, the sample rate was increased by periodically collecting event samples. This rate was chosen to be once every second, in order to match the available data from the power meter. Each performance event and power measurement was logged to a file during execution, with a corresponding timestamp, to allow synchronisation and post-processing. The results of periodic sampling is shown in Figure 7.2, where the results vary from what
Figure 7.1: Intensity of FPU operations correlated with power, measured over the entire execution of the micro-benchmark running an FPU workload.

was expected. Instead of the points forming a tight linear path, as was previously seen, the values within each cluster are spread vertically, creating a striping pattern. The modest horizontal variation within each cluster can be attributed to the pseudo-random numbers from the micro-benchmark control loops. However, the vertical spread is less readily explained. The presence of such a pattern indicates that the power use varies over the execution period, which was previously obscured through the use of aggregate values.

Given the consistent execution characteristics of the micro-benchmark, the vertical striping was suspected to be caused by an inherent system latency occurring at the beginning of each execution. If such a latency exists, the trend in the data should become more linear with increased execution times. Therefore each micro-benchmark configuration was re-run, increasing the execution time from 3 to 25 minutes. The results in Figure 7.3 show that when sampled over a longer period with the same sampling rate, data points return to lying on a linear path, though there is a long vertical tail in each cluster. The increased spread of data points within each cluster, along the x-axis, is a result of the increased range of pseudo-random numbers used during the significantly increased execution time. Moreover, with the longer execution period, it can be seen in the raw data that the vertical striping only occurs at the beginning of each execution.
Figure 7.2: Intensity of FPU operations correlated with power, sampled every second for \( \approx 3 \) minute execution of the micro-benchmark running FPU workload. Vertical stripes are a result of a warm-up effect.

From these experimental results, two general observations can be made. First, the granularity of performance and power samples have a direct impact on the accuracy of a power model, where coarse-grained samples can have a detrimental impact. Despite the lack of variation in the micro-benchmark’s execution characteristics, the use of aggregate values obscured some important trends within the measured power. Identifying such trends will be increasingly important when modelling general-purpose applications, consisting of multiple distinct workloads, where each execution phase has unique power characteristics requiring modelling. Therefore, the sample rate should adequately reflect the rate of variation in execution characteristics. Second, execution time also plays an important role when deriving a power model, where short execution times will obscure long-run power trends for a given benchmark. This is an issue with much of the existing work, where either short execution times, of less than a minute, are used, or the relevant execution details are left unspecified.

7.1.2 Importance of sample rate on proposed power model

The rate at which performance events are sampled during run-time power estimation is often not considered when using a multi-variable model, as it will not have an impact on the mean power estimation. This is because the model uses the intensity values
for a constant set of performance events, where the sum of each performance event’s intensity values will remain the same for a given time, regardless of the number of samples collected. However, this is not the case for W-Classifier due to the selective use of workload-specific linear functions.

Instead, a classification-based power model leverages workload-specific functions, requiring the sample rate to be responsive to any changes in execution workload. Otherwise, an incorrect power function will be used to model power for an extended period, potentially resulting in a decreased modelling accuracy. For instance, Figure 7.4 shows the power use of an instance of the NPB DC benchmark, where the x-axis is execution time and the y-axis is the corresponding power consumption. In the figure, it can be seen that DC consists of two workload types, modelled by different performance events and linear functions. For some periods of high processor utilisation, with power consumption around 400W, a compute-intensive workload classification is used for power estimation. For other time periods when the utilisation level is much lower, the workload is classified as being memory-dominant, allowing a different power estimation function to be used.

To accurately model power for the DC benchmark, the sample rate has to be of a high enough frequency to detect the periodic changes in utilisation that are strongly
associated with a change in workload. If coarse-grained performance values, aggregated
over the entire execution period, are used, only the dominant memory workload will
be detected. This will result in the memory-specific power model being used for the
entire execution, having a detrimental impact on power estimation as the periodic
spikes in power use will not be modelled. However, the significance of this impact is
dependent on the power and workload characteristics of a given application, where each
application will have a different level of workload variation. While the DC benchmark
has some execution variation, many of the other scientific benchmarks consist of a single
workload type, requiring a very low sample rate. Alternatively, general-purpose, user
applications may consist of a larger number of distinct workload types, necessitating
the use of higher sample rates.

For the purposes of the power model and accompanying discussion, a sample period
of one second was used. It is this periodic sampling in power estimation that enables
W-Classifier to detect all of these workload changes at run-time, resulting in the es-
timation trend matching the measured real power use. A decrease in the sampling
rate can result in modest increases in the estimation error. The sampling rate does
not have a very significant impact on the results for NPB benchmarks, because the
inherent nature of these benchmarks exhibits few detected workload changes if any.
Furthermore, a significantly higher sample rate was not practically feasible given the
hardware limitations of power metering, where a higher sample rate may additionally

Figure 7.4: Power use of the NPB DC benchmark.
be expected to increase execution overhead by requiring substantially higher rates of processing. Further exploration of the sample rate is left for future work.

7.2 Thermal effects

While it is widely recognised that temperature has a direct impact on power use, such effects are rarely considered or discussed when modelling power use. For those who are aware of the thermal effects on power consumption, it is commonly perceived that the thermal effects can be negated by locking the fan speed. This is due to the belief that the change of fan speed, in response to changes in thermal load, is the main cause of the variation in power use. An alternative technique to locking the fan speed is the use of a processor warm-up phase before the start of each execution.

While these policies may seem reasonable, they are based upon some key assumptions that oversimplify the relationship between system temperature and power. Neither of these policies can sufficiently negate the thermal effects, resulting in potentially erroneous conclusions when used.

7.2.1 Dynamic response to processor temperatures

It was observed in Section 7.1 that a latency in the power response occurred at the beginning of each micro-benchmark execution, shown in Figure 7.3 as the vertical spread of data points in the tail of each cluster. Given the consistent execution characteristics of the micro-benchmark, the most probable cause of the progressive change in power use is temperature related. To explore the potential impact of thermal effects, the external monitoring system was configured with IPMI to periodically log processor temperatures to a log file, along with the corresponding power measurements, for each micro-benchmark execution.

The results of an FPU-dominant micro-benchmark execution are shown in Figure 7.5, where the y-axis plots the normalised values for processor temperatures and system power, and the x-axis is the execution time in seconds. The power curve continuously rises for the first 400 seconds, before a stable power level is reached. Similarly, the temperature of each processor continues to increase from the beginning of execution until about the same time. This illustrates the presence of a strong relationship between processor temperatures and power use, as anticipated. However, the time required to reach a stable operating temperature was more surprising, at over six minutes! This is an important observation, as six minutes is longer than many benchmark
Figure 7.5: Normalised CPU temperatures and power meter readings for a long-run execution of the micro-benchmark running an FPU workload.

execution times used while deriving or evaluating power models. Therefore, temperature may play an important role in determining model accuracy, which is currently unacknowledged.

Furthermore, to explore the power response to thermal load, IPMI was additionally used to monitor the speed of each processor’s fan. Despite the operating temperatures nearly doubling, the fan speeds remain constant at 3600rpm for the entire execution. The same fan speed is even maintained while under an extreme thermal load, applied by the CPUburn (2014) benchmark. This observation dispels the perception that the power response to changes in thermal load is primarily caused by a change in the power used for cooling, as the power changed irrespective of the fan speeds. Therefore, policies that intentionally lock the fan speed through the BIOS to mitigate thermal effects, as adopted by Dhiman et al. (2010), are incapable of negating all of the variation in dynamic power.

7.2.2 Effectiveness of processor warm-up

Given that the power latency can be attributed to a delayed thermal response of the processor, it has previously been proposed that such effects can be negated by initially running a processor warm-up phase prior to benchmark execution. To explore the
warm-up policy, a 15-minute cool-down period is run before each micro-benchmark iteration, where the system remains idle. This is intended to ensure that each execution, and subsequent warm-up phase, begins from a consistent initial temperature. Following the cool-down, CPUburn is run concurrently on all processor cores for a set duration, depending on the intended level of processor warm-up. Figures 7.6 and 7.7 present two alternate warm-up times, 60 and 90 seconds respectively, for an INT weight micro-benchmark. The figures are similar to those previously presented, where the x-axis is the $FPU_{perf}$ intensity, and the y-axis is the corresponding power use.

![FPU perf intensity for INT weight micro-benchmark](image)

Figure 7.6: Intensity of FPU correlated with power, sampled every second for the micro-benchmark running an INT workload. Sampling starts after a 60-second period of processor warm-up.

Running a short warm-up of only 60 seconds, as shown in Figure 7.6, is sufficient to remove much of the thermal effects, resulting in a stronger linear correlation. Alternatively, a slightly longer 90 second warm-up period begins to over-heat the processor, resulting in a cool-down effect at the beginning of execution. This is shown in Figure 7.7 by the presence of data points above the main cluster, essentially inverting the previously observed tail for each cluster.

However, different types of workload in the benchmark need different warm-up periods. A 60 second processor warm-up phase provides the best results for a workload with lots of integer calculations, as shown in Figure 7.6. However, a workload with lots of floating point calculations requires a 90 second processor warm-up. These results
are illustrated by the Pearson’s Correlation Coefficient for the two workload types in Table 7.1. While differences in correlations are not significant they do illustrate the point that no single processor warm-up policy is sufficient for all workload types. Therefore, without prior knowledge of the specific workload type, the use of an initial processor warm-up phase may be prone to over-warming the processor.

![Figure 7.7: Intensity of FPU correlated with power, sampled every second for the micro-benchmark running an INT workload. Sampling starts after a 90-second period of processor warm-up.](image)

**Table 7.1:** Pearson’s Correlations for different workload types with different warm-up times.

<table>
<thead>
<tr>
<th>Processor warm-up time (seconds)</th>
<th>FPU&lt;sub&gt;weight&lt;/sub&gt;</th>
<th>INT&lt;sub&gt;weight&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.969611</td>
<td>0.977467</td>
</tr>
<tr>
<td>60</td>
<td>0.992794</td>
<td>0.997928</td>
</tr>
<tr>
<td>90</td>
<td>0.996615</td>
<td>0.992321</td>
</tr>
<tr>
<td>120</td>
<td>0.993053</td>
<td>0.985091</td>
</tr>
</tbody>
</table>
7.2.3 The challenges in accounting for thermal effects

While it may be desirable to enact run-time policies intended to mitigate system thermal effects on power, the observations made in this section have highlighted some common misconceptions made by such policies. Unfortunately, it is not likely that a single policy exists to reliably remove all warm-up effects on power consumption. The most likely cause of the observed thermal effects, illustrated by the delayed power response due to processor warm-up, is static power leakage from the processor. The leakage is caused by a reduction in the power efficiency of system components, such as voltage regulators, due to higher operating temperatures, resulting in increased power use (Chin, 2014). For example, it was noted in an Anandtech (2013) article, that lowering the processor temperature can reduce power by 12%, with the operating voltage and frequency remaining unchanged at 4.6GHz.

Despite leakage power having previously been incorporated into thermal models, the accuracy of such models is limited by the use of simplifying assumptions. For instance, Skadron, Stan, Huang, Velusamy, Sankaranarayanan, and Tarjan (2003) modelled leakage power as a percentage of total power, where the percentage is determined by the current operating temperature. However, this relies on the availability of accurate, run-time temperature values. Moreover, developing an accurate model of leakage power is complicated by the dependence on microarchitectural features, such as transistor sizes and densities. This will likely result in a system-specific model that is incapable of accurately modelling leakage power for a diverse set of alternate systems.

It might seem that the only way to reliably monitor thermal effects is through embedded temperature sensors. However, since their placement inside the socket is some distance away from the top of the processor, embedded sensors do not provide reliable temperature data (Citarella, 2013). Also such sensors were not designed for high precision temperature reading; their primary purpose is to provide an early warning system to prevent hardware damage.

In summary, two key observations have been made. First, benchmarks experience a significant warm-up period at the beginning of execution, resulting in a delayed power response that is not attributable to changes in fan speed. Second, a fixed duration processor warm-up is incapable of adequately pre-heating processors for a diverse set of application workloads, where each workload type requires a different processor warm-up duration. These findings are contrary to some widely held beliefs regarding processor thermal effects and the corresponding impact on power use.
7.3 Correlation of memory events and power

Memory-bound and CPU-bound workloads exhibit quite different power characteristics and in most cases are therefore treated differently. For power estimation, it is also common to use those performance events with a direct logical connection to the respective workload type. This intuitively makes sense, and is commonly expected to be the case. In modelling the power use of memory workloads, performance events directly related to memory were used, such as, instruction cache miss, data cache miss, L2-cache miss, L3-cache miss, DTLB miss and DRAM accesses.

Surprisingly, none of these performance events provided a strong correlation between memory use and power consumption. To illustrate this, the results for L3-cache misses are plotted in Figure 7.8. The micro-benchmark has been configured to execute a large number of memory accesses by increasing the \texttt{memory\_weight}, leaving all other micro-benchmark workload weights at the default value of one. The x-axis is the intensity of cache misses, which is calculated by taking the difference of two L3-cache miss performance values and dividing by the elapsed TSC value. The y-axis is the measured power. The cache intensity and power samples are collected once every second. The warm-up effect, seen by the vertical striping within each cluster, is present since there is no processor warm-up phase before data collection.

![L3-cache-miss for memory weight micro-benchmark](image)

Figure 7.8: Intensity of L3-cache misses correlated with power, sampled every second for the micro-benchmark running a memory workload.
The most notable observation to be made in Figure 7.8 is the distinct lack of any vertical offset between clusters. It seems that power is not functionally determined by L3-cache misses. The same results have been found for all other memory-related performance events mentioned, so we do not repeatedly show the results.

7.3.1 Potential reasons for observed relationship

A common approach taken in modelling power is to decompose the processor and the expected workload type into several key components, such as FPU and memory. Each component requires a specific, strongly correlated performance event to represent its power consumption. For instance, Singh et al. (2009) decomposed the processor into four primary categories; FPU, memory accesses, processor stalls and instructions retired. A single, strongly correlated performance event is then selected for each category from a small set of logically related events. In the case of memory, a performance event such as ‘cache misses’ is expected to correlate well with the activities of the memory subsystem. While this approach has worked in other power models, it was not able to be reproduced on the experimental system, in which there is much less of a link between the aforementioned memory-related performance events and power consumption.

This difference of results can likely be attributed to the architectural differences, although it is unknown which components actually have caused the difference, as there are several components which could possibly be playing a part. The first component could be the memory architecture. The experimental system uses a non-uniform memory access (NUMA) architecture, where, unlike the system architectures used in previous work, there is no single memory bank shared between all processors. The memory is arranged in 4GiB blocks beside each of the four processors. Given a workload of random memory accesses, extra overhead may be incurred if memory accesses are shifted to a remote processor’s memory block.

Also the processors are incapable of entering the sleep states, even at low levels. That means the processor maintains a busy loop or executes some other work while waiting for requests from the memory subsystem. Given the high thermal latencies, temperatures will remain high, despite the lower utilisation.

From this experimental observation, it can be concluded that for a modelling methodology to remain architecture independent, performance events should not be explicitly determined a priori. This equally applies to overzealous restrictions on the scope of performance events. It is primarily due to this observation that the proposed classification-based power model does not provide explicit guidance for workload-
specific performance event selection. Instead, the potential scope for such events is restricted with empirically supported, domain-specific knowledge, as such guidance does not determine the final power model implementation.

7.4 Exploring software configurations

Many of the existing modelling methodologies were presented without a detailed specification of the software configuration for either the training or evaluation benchmarks. Such configuration details can include benchmark configuration, problem size, compiler version and compiler optimisations. The absence of such specification details can likely be attributed to the perception that the software configuration has a minimal impact, if any, on the accuracy of the power model. However, this can be misleading, where this section will show how variations in software configuration can impact the power characteristics of benchmarks, and the corresponding model accuracy.

7.4.1 Benchmark configuration

The previous observations not only illustrated the importance of choosing an appropriate rate at which to sample a benchmark, but also the importance of ensuring a sufficiently long execution time. However, the execution time is dependent upon the benchmark configuration, requiring the publication of configuration details for an experiment to be reproducible. The absence of such details becomes a problem when benchmarks have multiple, configurable execution parameters, such as the problem size or iteration count, where multiple configurations will have varied execution characteristics, but are capable of achieving similar execution times.

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

This illustrates how a similar execution time can be achieved by two different benchmark configurations, with each having quite different power characteristics. Therefore, it can be concluded that execution time alone does not provide sufficient detail to
enable accurate model evaluation, potentially inhibiting fair comparisons between different modelling methodologies. This further highlights the need for a full disclosure of experimental configuration details in the literature, preventing erroneous conclusions from being drawn.

7.4.2 Effects of optimisation during compilation

A further source of variability in execution characteristics arises from the use of compiler optimisations. When deriving the power model, both benchmarks and micro-benchmarks are often configured to use no optimisations in order to ensure consistent performance events, thereby improving regression analysis. This results in a model that provides a much stronger fit to the training data. While this is desirable, it implicitly acknowledges that the use of optimisation flags will change execution behaviour.

To explore the potential impact, two optimisation levels on two different types of workload were evaluated on benchmarks from the NAS Parallel Benchmark suite. For the optimisation levels, -O0 and -O3 were chosen. This is because these are the two most commonly used configurations. -O0 is used to provide no compiler optimisations during the use of micro-benchmarks when deriving the model. Alternatively, -O3 is one of the most commonly used configurations for standard benchmark execution, providing a reasonable performance improvement in most situations. For the benchmarks, a computation-intensive workload is represented by fast Fourier Transform (FT). In contrast, the Data Cube (DC) benchmark provides a more memory-intensive workload.

The use of optimisation when compiling the benchmarks contributes a large amount of noise to the performance data, with much more variation in the power used. An example of this can be seen in Figure 7.9, where two instances of the FT benchmark are shown, each using a different optimisation argument. The x-axis plots the intensity of retired-\(\mu\)ops, while the power for each corresponding one second sample is given on the y-axis. From this it can be seen that the use of no optimisation has a much narrower spread of power use for each corresponding performance value. Alternatively, the use of compiler optimisation has a much greater spread of power values, making it much harder to predict power. Overall, the use of optimisation causes a shift of the main cluster down the x-axis, indicating lower retired-\(\mu\)ops than that of the instance using no optimisation. From this initial view, it is clear that the use of optimisation for a benchmark changes the run-time utilisation levels and power, meaning a power model unaware of optimisation may end up being inaccurate.

While the optimisation reduced the retired-\(\mu\)ops intensity, it did not have an impact
on the FPU intensity. The overall impact of optimisation can be seen in Table 7.2, where a much greater variation in power is seen, both from the standard deviation and maximum value, however optimisation results in lower power usage overall.

<table>
<thead>
<tr>
<th>Table 7.2: Statistics for different optimisation levels.</th>
</tr>
</thead>
<tbody>
<tr>
<td>power</td>
</tr>
<tr>
<td>mean</td>
</tr>
<tr>
<td>standard deviation</td>
</tr>
<tr>
<td>median</td>
</tr>
<tr>
<td>maximum</td>
</tr>
</tbody>
</table>

For the more memory-intensive DC benchmark, the use of optimisation similarly results in lower power overall when more optimisation is performed. However in contrast to FT, this time a much lower maximum power leads to the standard deviation being smaller with the use of optimisation. This can most likely be attributed to the optimiser’s ability to reduce the impact on power of the periodic, compute-intensive phases on the otherwise memory-intensive workload.
7.4.3 Impact of compiler version

Similar to the use of compiler optimisation, different compiler versions could be expected to cause variations in execution power characteristics. This results from newer compilers including additional functionality and execution optimisations that are intended to improve execution performance. However, unlike the effects caused by compiler optimisations, the execution characteristics of each benchmark does not appear to change significantly with different compiler versions. Note this may just be due to limited testing regarding both compiler versions and benchmarks.

While the overall variation may not be significant, changing the version of gcc from 4.6.3 to the older 4.4.7 did result in an increased maximum power level reached. This can be seen in Figure 7.10, which plots the power measurements once a second for an execution of the FT benchmark. The main point of difference seen between the two compiler versions is the peak power, where gcc-4.6.3 has a peak power of 449.4W, while gcc-4.4.7 is 8.5W higher at 457.9W. Interestingly, there is little variation in power at other stable points, such as the period of approximately 410W power consumption towards the end of execution.

![Impact of gcc version on NPB FT](image)

Figure 7.10: Power versus time for two executions of the FT benchmark, that use gcc-4.4.7 and gcc-4.6.3 respectively.
7.4.4 Challenges posed by the software configuration dependence

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

Furthermore, the significance of the software configuration will be architecture-specific, meaning that other systems may suffer much larger variations in power. For instance, Cameron (2013) observed a significant impact of different MPI libraries, where power increased from 120W to 140W, which is a large percentage variation.

Some of this impact could potentially be mitigated by incorporating some compilation and configuration-derived variation into both the training data sets as well as with benchmarks used for evaluating and tuning the model. Moreover, the specification of software configuration details allows fair and reliable comparisons to be made between different modelling methodologies, preventing potentially misleading or erroneous conclusions from being drawn.

7.5 Impact of hardware selection and configuration

Despite the architectural dependence of performance-based power models, the impact of hardware selection and configuration is rarely explicitly considered in much of the existing literature. While the hardware selection will not directly alter the estimated power, such details influence the perceived accuracy, especially in making comparisons with other power models. This can be attributed to differences in the dynamic power, relative to static power, for alternate hardware configurations. This section will demonstrate the potential of a bias towards low-power systems, that can misrepresent models if sufficient configuration details are not explicitly considered.
7.5.1 Potential bias of hardware selection

Power use can vary significantly between systems, where different tradeoffs are made between power and performance to meet a variety of system use-cases. For instance, consumer electronics, such as laptops and desktops, place a high emphasis on a low power consumption, at the cost of higher system performance. This use of power efficient hardware results in a smaller dynamic power consumption, relative to static power. Alternatively, a more performance driven, power hungry enterprise server will sacrifice power savings for increased system performance. This tradeoff typically results in a higher dynamic power range.

Given such varied power characteristics, the selection of an experimental system can influence the perceived modelling accuracy, where an inaccurate model will have a lower absolute estimation error on a more energy efficient system than a high performance system. This is a result of the relative differences in dynamic power, where a large static power can obscure the modelling estimation error. To illustrate the potential of an architecture selection bias, Table 7.3 presents a comparison of the high power and performance Dell PowerEdge, previously discussed in Chapter 5, and an Apple iMac\(^1\) desktop. The two systems have a respective peak execution power of 125W and 500W, and a maximum dynamic power of 56% and 100% of static power, respectively.

For the system comparison, the median power value is used as a naïve power model, providing a constant power estimate for each system, calculated as the median operating power for a suite of equally distributed workloads, i.e. \(\text{static} + \frac{\text{dynamic}}{2}\). While such a model is overly simplistic, it serves the intended purpose of providing a constant estimation error for dynamic power across systems, enabling the impact of hardware power characteristics to be evaluated. This results in a maximum estimation error of 22.5W or 18% for the iMac. Alternatively, the PowerEdge server has a more sizeable maximum estimation error of 125W, 25%.

From these results, it could be concluded that the power model is able to achieve a higher estimation accuracy on the lower-power iMac than the high-power server. However, since the same power model is used on both systems, such a result is clearly misleading. Instead, if the error is presented relative to dynamic power, both systems have the same worst case estimation error of 50%. The only difference between the two systems is their relative proportion of dynamic power. Therefore, the static power obscures the true modelling accuracy, where the use of a low-power system makes

---

\(^1\)The 27” Apple iMac is configured with a 3.2GHz Intel Core i5 processor, 8GB DDR3 memory, and an NVIDIA GeForce GT755M graphics card.
Table 7.3: Bias of architecture selection.

<table>
<thead>
<tr>
<th></th>
<th>iMac desktop</th>
<th>PowerEdge server</th>
</tr>
</thead>
<tbody>
<tr>
<td>static power</td>
<td>80W</td>
<td>250W</td>
</tr>
<tr>
<td>dynamic power</td>
<td>45W</td>
<td>250W</td>
</tr>
<tr>
<td>total peak power</td>
<td>125W</td>
<td>500W</td>
</tr>
<tr>
<td>power estimate (constant median value)</td>
<td>102.5W</td>
<td>375W</td>
</tr>
<tr>
<td>max error (watts)</td>
<td>22.5W</td>
<td>125W</td>
</tr>
<tr>
<td>error (% total power)</td>
<td>18%</td>
<td>25%</td>
</tr>
<tr>
<td>error (% dynamic power)</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>

the model appear more accurate than it is. Without the explicit context provided by dynamic power, any presented results may be misinterpreted, resulting in erroneous conclusions.

This observation has shown the potential for a hardware selection bias, where the hardware chosen by the experimenter can influence the perceived modelling accuracy, further inhibiting direct comparisons between different power models.

7.5.2 Impact of hardware configuration

A further source of influence on the dynamic power range is the chosen hardware configuration. Similar to the selection bias of low-power hardware, many standard system components can be configured to operate in special-purpose, low-power states, replicating the previous scenario of power efficient hardware. The most prevalent example of this is processor dynamic voltage and frequency scaling (DVFS), which allows the processor’s operating frequency and voltage to be adjusted to a set of predetermined levels. Such changes in the hardware configuration directly impact both the power and performance characteristics of a system.

To illustrate the impact of alternate operating frequencies, Figure 7.11 presents the average power for several benchmarks, running at four different processor frequencies. System idle is additionally included to represent the base (static) power of the experimental system, as power is not capable of dropping below this level for each configuration. It can be seen in Figure 7.11 that the idle power does not vary much between each of the operating frequencies, with a maximum difference of 26W. However, more sizeable variations occur during benchmark execution. For example, executing
the Mandelbrot benchmark at 0.8GHz, increases power use by 34W from idle to 254W. At the highest frequency, 2.5GHz, a more significant 186W power increase occurs, rising from 249W to 435W. This is an increase in power by 175% from idle, compared to only 115% while operating at 0.8GHz. The results for all other benchmarks demonstrate the same trend, where lowering the processor’s operating frequency decreases the dynamic power range.

Figure 7.11: Average power use of various parallel benchmarks, utilising all 16 processing cores, illustrating alternate dynamic power ranges.

Therefore, the hardware configuration can be expected to have an impact on the perceived modelling accuracy. Systems that enable power saving features intended to minimise the processor’s operating frequency will have a lower, more predictable, dynamic power. In comparison, more performance-orientated system configurations that set the operating frequency to the highest possible will have less predictable power consumption. Despite such hardware configuration details having a potentially significant impact on the presented modelling accuracy, much of the existing work leaves such details unspecified, preventing fair and reliable comparisons between power models.

7.6 Summary

In this chapter I discussed a number of misconceptions faced when deriving or evaluating the W-Classifier power model. The implementation misconceptions have arisen
from differences in the experimental configuration and the handling of external factors that impact modelling accuracy. This is of particular importance for performance-based power estimation as the derived models are largely architecture dependent. However, these observations are not limited to processor performance events, but equally apply to many other potential performance events. It was shown that changes in the hardware or software configuration can adversely impact the resulting power model. Therefore, a suitably general modelling methodology is important for deriving a robust power model.

While failing to disclose all relevant configuration details would not affect the correctness of the previous work, it can lead to erroneous conclusions being drawn from the presented results. Therefore, it is only with the detailed documentation and discussion of the experimental setup that such misconceptions can be overcome by good modelling and evaluation practises.
Chapter 8

Conclusions and Future Work

In this dissertation I have presented an application-independent methodology called W-Classifier for deriving a collection of workload-specific power models with a parameterised micro-benchmark. Experimental results have shown that W-Classifier is able to estimate power use significantly better than a conventional multi-variable power model. In this chapter I summarise my contributions and discuss potential future research.

8.1 Contributions

While it is common for a single benchmark suite to be used as the source of model training data, the scope of performance values may be insufficient to derive a broadly representative model. Therefore, W-Classifier uses a parameterised micro-benchmark, designed to reproduce a selection of representative, synthetic workloads. This design provides a configurable method of generating various workload-specific utilisation levels that encompass a diverse range of application workloads.

W-Classifier leverages this parameterised design to derive a collection of workload-specific linear functions for power estimation, where a separate power model is used for each of the representative workload types. This enables more accurate power estimation through the use of modelling functions tailored to the current execution context. A single data set from the micro-benchmark provides sufficient detail to be used for performance event selection and workload classification, and to derive the workload-specific power models.

The modelling accuracy of W-Classifier was further improved by addressing a number of common implementation misconceptions that were observed in the existing lit-
erature. Firstly, experimental results found the execution time to be more important than existing work would indicate. Therefore, long execution times were used for the training micro-benchmark and evaluation benchmarks. Secondly, previously proposed methods for mitigating thermal effects were found to be insufficient, so the modelling methodology removes several minutes from the beginning of each training data set to remove the impact of processor warm-up. This is made possible with the long execution times, as sufficient data points remain to still allow regression analysis. Finally, experimentally supported domain-specific knowledge was used to limit the number of performance events collected for model training, thereby avoiding any assumed event correlations. By not imposing strict constraints on the design of the power model a priori, the modelling methodology is less prone to the misconception of previously assumed relationships, where all performance events are evaluated for a given workload type.

The experimental results have shown that W-Classifier can better track run-time power changes than the multi-variable model. This enables W-Classifier to achieve more accurate result across a broad range of application workloads. In comparison, the multi-variable model was able to achieve good modelling accuracy for the compute-intensive workloads, but the inherent workload bias in the power model resulted in high estimation errors for the non-compute-intensive benchmarks such as DC and SP.

In evaluating W-Classifier, direct comparisons were not made with previously proposed power models. It was shown with experimental results, that making such comparisons can be misleading, resulting in erroneous conclusions, because such comparisons will not be able to account for the differences in evaluation metrics, software configuration, compiler optimisations, selected system hardware, hardware configuration, and system static/dynamic power. Each of these factors were found to impact the perceived modelling accuracy. Therefore, the evaluation of W-Classifier did not make direct comparisons with different modelling methodologies in order to avoid such evaluation misconceptions. However, since previous power models were in general based on multi-variable linear regression, W-Classifier should perform better than them according to our comparison to a general multi-variable power model.

In conclusion, my thesis has shown that workload classification can be used to improve modelling accuracy, while simplifying the resulting power model. Workload classification enables accurate power estimates for a wide range of applications, through the use of workload-specific power models. This has been shown to increase the precision of power estimation compared to typical multi-variable approaches that do not
perform workload classification. Furthermore, W-Classifier was able to be derived from a training data set gathered from a single parameterised micro-benchmark, thereby providing a simpler model training procedure. Finally, power models based on workload classification are able to provide additional benefits, not possible with traditional metering or modelling, such as the workload context for the power use. This aids the broader objectives of power management policies by enabling more informed actions, based upon system power consumption.

8.2 Future research

The future work may include further efforts to determine the cause of the remaining power estimation error, and attempts to provide bounds on such errors. This may be achieved by including additional system metrics into the power model, such as determining how temperature affects base power consumption, and how the base power operates across multicore configurations that are operating with different per-core utilisation levels. Processor temperature metrics can additionally be incorporated into the power model. Much of the previous work that has used thermal sensors, such as Jarus et al. (2014), did not use temperature to correct the warm-up effects, but instead used the values from stable operating temperatures. Both uses of temperature can be used to further improve modelling accuracy.

Furthermore, machine learning techniques, such as support vector machines or decision trees, can be used for the classification algorithm. This will allow for part of the modelling procedure to be automated, enabling more convenient deployments. Alternatively, clustering techniques, such as k-means, can be explored for dividing the power and performance values into clusters, performing workload classification or power estimation. These techniques will likely be required when performing an exhaustive analysis on significantly larger data sets that may be a result of increases in the number of performance events or types of workload.

This automated model training will allow the power model to be more readily deployed on a diverse range of system architectures, enabling the development of new software-based power management policies. For instance, the power model can be used to supplement existing user feedback methods, such as the OS X energy impact factor, by providing quantitative values for power consumption. Using general metrics, such as power consumption (Watts), will improve usability as such metrics are generally better understood than more novel evaluation methods. Moreover, this facilitates improved
user analysis of power consumption, where illustrating historic power values can provide the user with insights into long-run trends.

With this improved user feedback, more advanced power-management policies can be implemented, allowing power management to be enacted on a per-application basis. This may then enable user configurable policies, where power saving techniques can better evaluate the power/performance tradeoff on a per-application basis, where high priority applications can receive an increased performance priority, while all other applications emphasise power savings. While such a policy may have lower overall power savings, it will practically achieve higher power savings due to its better applicability, since there is currently the tendency for users to disable power saving features due to the unacceptable impact of performance on those high priority applications.
References


Appendix A

Performance Events

A.1 Performance event list

A full list of the 164 performance events used in perf are presented here. The first 41 performance events are the available events found in perf tool, which includes both kernel software events and processor hardware performance events. The remaining 123 are processor-specific performance counter events for the AMD Family 10h Processors (AMD, 2013).

Perf tool events:
1. cycles
2. instructions
3. cache-references
4. cache-misses
5. branches
6. branch-misses
7. bus-cycles
8. cpu-clock-msecs
9. task-clock-msecs
10. page-faults
11. minor-faults
12. major-faults
13. context-switches
14. CPU-migrations
15. alignment-faults
16. emulation-faults
17. L1-dcache-loads
18. L1-dcache-load-misses
19. L1-dcache-stores
20. L1-dcache-store-misses
21. L1-dcache-prefetches
22. L1-dcache-prefetch-misses
23. L1-icache-loads
24. L1-icache-load-misses
25. L1-icache-prefetches
26. L1-icache-prefetch-misses
27. LLC-loads
28. LLC-load-misses
29. LLC-stores
30. LLC-store-misses
31. LLC-prefetch-misses
32. dTLB-loads
33. dTLB-load-misses
34. dTLB-stores
35. dTLB-store-misses
36. dTLB-prefetches
37. dTLB-prefetch-misses
38. iTLB-loads
39. iTLB-load-misses
40. branch-loads
41. branch-load-misses

AMD 10h processor performance counter events:
42. Dispatched FPU Operations
43. Cycles in which the FPU is Empty
44. Dispatched Fast Flag FPU Operations
45. Retired SSE Operations
46. Retired Move Ops
47. Retired Serializing Ops
48. Number of Cycles that a Serializing uop is in the FP Scheduler
49. Segment Register Loads
50. Pipeline Restart Due to Self-Modifying Code
51. Pipeline Restart Due to Probe Hit
52. LS Buffer 2 Full
53. Locked Operations
54. Retired CLFLUSH Instructions
55. Retired CPUID Instructions
56. Cancelled Store to Load Forward Operations
57. SMIs Received
58. Data Cache Accesses
59. Data Cache Misses
60. Data Cache Refills from L2 or Northbridge
61. Data Cache Refills from the Northbridge
62. Data Cache Lines Evicted
63. L1 DTLB Miss and L2 DTLB Hit
64. L1 DTLB and L2 DTLB Miss
65. Misaligned Accesses
66. Microarchitectural Late Cancel of an Access
67. Microarchitectural Early Cancel of an Access
68. Single-bit ECC Errors Recorded by Scrubber
69. Prefetch Instructions Dispatched
70. DCACHE Misses by Locked Instructions
71. L1 DTLB Hit
72. Ineffective Software Prefetches
73. Global TLB Flushes
74. Memory Requests by Type
75. Data Prefetcher
76. MAB Requests
77. MAB Wait Cycles
78. Northbridge Read Responses by Coherency State
79. Octwords Written to System
80. CPU Clocks not Halted
81. Requests to L2 Cache
82. L2 Cache Misses
83. L2 Fill/Writeback
84. Page Size Mismatches
85. Instruction Cache Fetches
86. Instruction Cache Misses
87. Instruction Cache Refills from L2
88. Instruction Cache Refills from System
89. L1 ITLB Miss, L2 ITLB Hit
90. L1 ITLB Miss, L2 ITLB Miss
91. Pipeline Restart Due to Instruction Stream Probe
92. Instruction Fetch Stall
93. Return Stack Hits
94. Return Stack Overflows
95. Instruction Cache Victims
96. Instruction Cache Lines Invalidated
97. ITLB Reloads
98. ITLB Reloads Aborted
99. Retired Instructions
100. Retired uops
101. Retired Branch Instructions
102. Retired Mispredicted Branch Instructions
103. Retired Taken Branch Instructions
104. Retired Taken Branch Instructions Mispredicted
105. Retired Far Control Transfers
106. Retired Branch Resyncs
107. Retired Near Returns
108. Retired Near Returns Mispredicted
109. Retired Indirect Branches Mispredicted
110. Retired MMXTM/FP Instructions
111. Retired Fastpath Double Op Instructions
112. Interrupts-Masked Cycles
113. Interrupts-Masked Cycles with Interrupt Pending
114. Interrupts Taken
115. Decoder Empty
116. Dispatch Stalls
117. Dispatch Stall for Branch Abort to Retire
118. Dispatch Stall for Serialization
119. Dispatch Stall for Segment Load
120. Dispatch Stall for Reorder Buffer Full
121. Dispatch Stall for Reservation Station Full
122. Dispatch Stall for FPU Full
123. Dispatch Stall for LS Full
124. Dispatch Stall Waiting for All Quiet
125. Dispatch Stall for Far Transfer or Resync to Retire
126. FPU Exceptions
127. DR0 Breakpoint Matches
128. DR1 Breakpoint Matches
129. DR2 Breakpoint Matches
130. DR3 Breakpoint Matches
131. Retired x87 Floating Point Operations
132. IBS Ops Tagged
133. LFENCE Instructions Retired
134. SFENCE Instructions Retired
135. MFENCE Instructions Retired
136. DRAM Accesses
137. DRAM Controller Page Table Overflows
138. Memory Controller DRAM Command Slots Missed
139. Memory Controller Turnarounds
140. Memory Controller Bypass Counter Saturation
141. Thermal Status
142. CPU/IO Requests to Memory/IO
143. Cache Block Commands
144. Sized Commands
145. Probe Responses and Upstream Requests
146. GART Events
147. Memory Controller Requests
148. CPU to DRAM Requests to Target Node
149. IO to DRAM Requests to Target Node
150. CPU Read Command Latency to Target Node 0-3
151. CPU Read Command Requests to Target Node 0-3
152. CPU Read Command Latency to Target Node 4-7
153. CPU Read Command Requests to Target Node 4-7
154. CPU Command Latency to Target Node 0-3/4-7
155. CPU Requests to Target Node 0-3/4-7
156. HyperTransport™ Link 0 Transmit Bandwidth
157. HyperTransport™ Link 1 Transmit Bandwidth
158. HyperTransport™ Link 2 Transmit Bandwidth
159. HyperTransport™ Link 3 Transmit Bandwidth
160. Read Request to L3 Cache
161. L3 Cache Misses
162. L3 Fills caused by L2 Evictions
163. L3 Evictions
164. Non-cancelled L3 Read Requests

Table A.1: The 32 performance events found to have a weak correlation with power for the Mandelbrot benchmark.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instructions</td>
<td>Dispatched Fast Flag FPU Operations</td>
</tr>
<tr>
<td>cache-references</td>
<td>Page Size Mismatches</td>
</tr>
<tr>
<td>cache-misses</td>
<td>Instruction Cache Refills from System</td>
</tr>
<tr>
<td>branches</td>
<td>Instruction Cache Victims</td>
</tr>
<tr>
<td>branch-misses</td>
<td>ITLB Reloads</td>
</tr>
<tr>
<td>cpu-clock-msecs</td>
<td>ITLB Reloads Aborted</td>
</tr>
<tr>
<td>task-clock-msecs</td>
<td>Retired Instructions</td>
</tr>
<tr>
<td>page-faults</td>
<td>Retired Taken Branch Instructions</td>
</tr>
<tr>
<td>minor-faults</td>
<td>Retired Taken Branch Instructions Mispredicted</td>
</tr>
<tr>
<td>dTLB-loads</td>
<td>Retired Indirect Branches Mispredicted</td>
</tr>
<tr>
<td>iTLB-loads</td>
<td>Interrupts-Masked Cycles</td>
</tr>
<tr>
<td>branch-load-misses</td>
<td>Dispatch Stall for Serialization</td>
</tr>
<tr>
<td>MAB Requests</td>
<td>Dispatch Stall for LS Full</td>
</tr>
<tr>
<td>LS Buffer 2 Full</td>
<td>Dispatch Stall Waiting for All Quiet</td>
</tr>
<tr>
<td>Data Cache Misses</td>
<td>CPU Requests to Target Node 0-3/4-7</td>
</tr>
<tr>
<td>L3 Evictions</td>
<td>Microarchitectural Early Cancel of an Access</td>
</tr>
</tbody>
</table>
Table A.2: The 15 most strongly correlated performance events with the Mandelbrot benchmark, calculated using Spearman’s rank correlation.

<table>
<thead>
<tr>
<th>Event</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>0.2059</td>
</tr>
<tr>
<td>Cycles in which the FPU is Empty</td>
<td>-0.3291</td>
</tr>
<tr>
<td>Pipeline Restart Due to Probe Hit</td>
<td>-0.2076</td>
</tr>
<tr>
<td>Instruction Fetch Stall</td>
<td>-0.1903</td>
</tr>
<tr>
<td>Return Stack Hits</td>
<td>-0.192</td>
</tr>
<tr>
<td>Retired Branch Instructions</td>
<td>0.1901</td>
</tr>
<tr>
<td>Retired Mispredicted Branch Instructions</td>
<td>0.1954</td>
</tr>
<tr>
<td>Retired Far Control Transfers</td>
<td>-0.1881</td>
</tr>
<tr>
<td>Retired Branch Resyncs</td>
<td>-0.3206</td>
</tr>
<tr>
<td>Retired Near Returns</td>
<td>-0.3057</td>
</tr>
<tr>
<td>Interrupts Taken</td>
<td>-0.237</td>
</tr>
<tr>
<td>Dispatch Stall for Segment Load</td>
<td>-0.243</td>
</tr>
<tr>
<td>Dispatch Stall for Far Transfer or Resync to Retire</td>
<td>-0.2178</td>
</tr>
<tr>
<td>CPU Read Command Requests to Target Node 4-7</td>
<td>-0.2459</td>
</tr>
<tr>
<td>CPU Command Latency to Target Node 0-3/4-7</td>
<td>-0.1831</td>
</tr>
</tbody>
</table>
Appendix B

Source Code

B.1 Micro-benchmark

In this section we present the code used for the micro-benchmark. Code segments were discussed in Chapter 5.

```c
/* ***********************************************************
* File: micro.c
* Author: Jason Mair
*
* The main file for the micro-benchmark execution.
* *********************************************************** */
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <inttypes.h>
#include <signal.h>
#include <time.h>
#include "micro.h"

/*
* Function to read the current processor TSC (Time Stamp Counter)
*/
inline unsigned long tick(){
  unsigned int a, d;
  __asm __volatile("rdtsc" : "=a" (a), "=d" (d) : "c" (0x10) );
  return ((unsigned long) a) | (((unsigned long) d) << 32);
```
/*
 * Function to read the PMC value for the given PMC register index.
 *
 * c - the index of the PMC register to be read
 */
inline unsigned long perf(unsigned int c){
    unsigned int a, d;
    __asm __volatile("rdpmc": "a" (a), "d" (d): "c" (c));
    return ((unsigned long) a) | (((unsigned long) d) << 32);
}

/*
 * Return an index for writing to the mixed memory array.
 */
inline unsigned long long write_mem(){
    write_cur = (write_cur * write_step) % mixed_array_size;
    return write_cur;
}

/*
 * Return an index for reading the mixed memory array.
 */
inline unsigned long long read_mem(){
    read_cur = (read_cur * read_step) % mixed_array_size;
    return read_cur;
}

/*
 * Alarm handling function. Records the PMC events and TSC for each event trigger.
 *
 * signum - alarm signal
 */
void handle_alarm(int signum){
alarm(alarm_timer);

counter[count][0] = perf(0);
counter[count][1] = perf(1);
counter[count][2] = perf(2);
counter[count][3] = perf(3);
counter[count++][4] = tick();

/*
 * Return the next workload execution time.
 */
int get_time(){
    pos = (pos * step) % time_array_size;
    return time_array[pos];
}

/*
 * The main function that initialises all of the execution values.
 * Once complete
 * execution of the micro-benchmark workload kernels begins.
 * Finally all of the
 * logged performance values are written to stdout.
 * *
 * argc - the number of commandline arguments
 * argv - array of commandline arguments
 */
int main(int argc, char *argv[]){
    long num_iters = 40000000;
    long iter = 0;
    long i = 0;

    int c;
    long start_time;

    // Initialise the workload ratios for each workload type
    int fpu_ratio = 1;

    // ...
int int_ratio = 1;
int memory_ratio = 1;
int nop_ratio = 1;
int cache_ratio = 1;

// Process the input parameters
while ((c = getopt (argc, argv, "f:i:m:n:c:g:r:w:t:x:")) != -1) {
    switch (c) {
    case 'f':
        fpu_ratio = atoi(optarg);
        break;
    case 'i':
        int_ratio = atoi(optarg);
        break;
    case 'm':
        memory_ratio = atoi(optarg);
        break;
    case 'n':
        nop_ratio = atoi(optarg);
        break;
    case 'c':
        cache_ratio = atoi(optarg);
        break;
    case 'g':
        num_iters = atoi(optarg);
        break;
    case 'r':
        read_cur = atoi(optarg);
        break;
    case 'w':
        write_cur = atoi(optarg);
        break;
    case 't':
        start_time = atoi(optarg);
        break;
    case 'x':
        core = atoi(optarg);
        break;
    }
}
time_array = (int*)malloc(time_array_size * sizeof(int));

if(time_array == NULL){
    printf("Failed to allocate time_array\n");
    exit(1);
}

// alloc memory
unsigned long* mixed_array = (unsigned long*)malloc(
    mixed_array_size * sizeof(unsigned long));

if(mixed_array == NULL){
    printf("Failed to allocate mixed_array\n");
    exit(1);
}

// alloc cache
int* cache_array = (int*)malloc(cache_array_size * sizeof(int));

if(cache_array == NULL){
    printf("Failed to allocate cache_array\n");
    exit(1);
}

// init FPU
double fv = 2.271;
double fz = 1.0024;
double fs = 1.1234;
double ft = fs * fv * fz;

// init INT
unsigned long is = 123456;
unsigned long iz = 37;
unsigned long it = 7284354168;
unsigned long iv = 35914;

// init time_array
for(i = 0; i < time_array_size; i++){
    time_array[i] = i*20;
}
// init mixed_array
for (i = 0; i < mixed_array_size; i++) {
    mixed_array[i] = i;
}

// init cache_array
for (i = 0; i < cache_array_size; i++) {
    cache_array[i] = i;
}

// start execution at the signalled time
while (time(NULL) < start_time) { sleep(1); }

// set and arm the pmc alarm
signal(SIGALRM, handle_alarm);
alarm(alarm_timer);

printf("start: %ld\n", time(NULL));

int temp;
unsigned long strt;
unsigned long next_time = tick();

for (iter = 0; iter < num_iters; iter++) {
    // FPU
    next_time = tick() + (get_time() * fpu_ratio);
    while (tick() < next_time) {
        for (i = 0; i < loop_count; i++) {
            fs = fmod((fs * fz), FPU_MAX_LOWER);
            ft = fmod((ft * fv), FPU_MAX_UPPER);
        }
    }

    // INT
    next_time = tick() + (get_time() * int_ratio);
while (tick() < next_time) {
    for (i = 0; i < loop_count; i++) {
        is = (is * iz) % INT_MAX_LOWER;
        it = (it * iv) % INT_MAX_UPPER;
    }
}

// Memory
next_time = tick() + (get_time() * memory_ratio);
while (tick() < next_time) {
    for (i = 0; i < loop_count; i++) {
        mixed_array[write_mem()] = mixed_array[read_mem()];
    }
}

// NOP
next_time = tick() + (get_time() * nop_ratio);
while (tick() < next_time) {
    for (i = 0; i < loop_count; i++) {
        __asm __volatile("nop");
    }
}

// Cache
next_time = tick() + (get_time() * cache_ratio);
while (tick() < next_time) {
    for (i = 0; i < loop_count; i++) {
        temp = cache_array[cache_pos % cache_array_size];
        cache_array[cache_pos % cache_array_size] = cache_array[((cache_pos + cache_array_size) % cache_array_size);
        cache_array[((cache_pos += cache_array_size) % cache_array_size) = temp;
    }
}

// stop alarm going off again
alarm(0);
time_t stop = time(NULL);

for (i = 0; i < count; i++) {
    // Print each performance counter
    printf(" PMC\n 0 \t%lu
 1 \t%lu
 2 \t%lu
 3 \t%lu", counter[i][0], counter[i][1], counter[i][2], counter[i][3]);

    // Print the time tick
    printf(" TSC\t%lu\n", counter[i][4]);
}

printf(" stop: %ld\n", stop);

return 0;

Listing B.1: micro.c

#ifndef MICRO_H_
#define MICRO_H_

// FPU
double FPU_MAX_LOWER = 1233369.896725;
double FPU_MAX_UPPER =
    5924408747168683944341877186346157142588554313875317373846869270180891238283146929852434379565481240582380411450058976973435280296941357947935505384475543375367469984876420795182528593920.094287;

// INT
unsigned long INT_MAX_LOWER = 12345678910;
unsigned long long INT_MAX_UPPER = 5924408874716868394;

// Memory
unsigned long mixed_array_size = 125829371;
unsigned long long read_cur = 10001;
unsigned long read_step = 31;
unsigned long long write_cur = 472103;
unsigned long write_step = 953;
B.2 W-Classifier Algorithm

This is the code for the W-Classifier on the experimental system, discussed in Chapter 5. The regression model values are therefore implementation specific, and are derived from the execution of the micro-benchmark on the experimental system. This code is presented as a demonstration of how the performance events can be used in implementing a W-Classifier power model.

```c
/* ***********************************************************
 * The power estimation algorithm for W-Classifier
 ***********************************************************/

if(perf.fpu > 1.0){
    // Cache workload
    if(perf.dispatch > 10.0){
        if(perf.retired > 1.0){
            // Mixed FPU and cache workload
            power = 424.028 + (-4.80427 * perf.retired);
        } else{
        }
    } else{
    }
}
```
// Mixed INT and cache workload
power = 475.223 + (-10.7538 * perf.dispatch);
#else
  // FPU workload
  power = 383.771 + (2.93496 * perf.fpu);
#else if(perf.retired > 1.0){
  // INT workload
  power = 394.235 + (-0.594378 * perf.dispatch);
} else{
  // Memory or idle workload
  power = 222.205 + (10 * perf.cpu);
}
Appendix C

Model Accuracy

C.1 Power Model

Figure C.1: A comparison of model accuracy for the multi-variable and W-Classifier power models.